10 Variations On IIR Biquad Filters

10.1 OVERVIEW

Digital Signal Processing Applications Using the ADSP-2100 Family, Volume 1, contains a chapter about digital filters. That chapter (Chapter 5) includes information about second-order sections of Infinite Impulse Response, or IIR, filters. The particular second-order sections discussed in Volume 1, are commonly referred to as biquads.

This chapter includes the following variations on the basic IIR biquad filter and the filter subroutines described in Volume 1:

- Multiprecision filters
- Optimized filter subroutines

10.1.1 IIR Biquad Filter

Figure 10.1 shows the structure of a second-order biquad IIR filter section. You can design IIR filters of an order greater than two by cascading multiple second-order biquad IIR filter sections. The filter sums the products of the current input, x(n), the previous two inputs, x(n-1) and x(n-2), the past two results, y(n-1) and y(n-2), and their respective coefficients, b0, b1, b2, a1 and a2. The biquad has a necessary coefficient scaling factor when one or more of the coefficients are greater than 1.0.

10.1.2 Biquad Filter Subroutine

Listing 10.1 is the subroutine from *Digital Signal Processing Applications Using the ADSP-2100 Family*, Volume 1, for a basic biquad filter. This filter has a 16-bit input, 16-bit output, and 16-bit coefficients. This code lets the ADSP-2100 Family DSP perform a Nth-order IIR filter by performing N / 2 biquads. The execution time is [8*(N / 2) + 10] instruction cycles. For example, a tenth-order filter executes in 50 instruction cycles, or in 3 µs using a DSP with 60 ns cycle time. The DSP can perform a tenth-order IIR filter on a signal sampled at more than 300 kSa / s.

10 Variations On IIR Biquad Filters

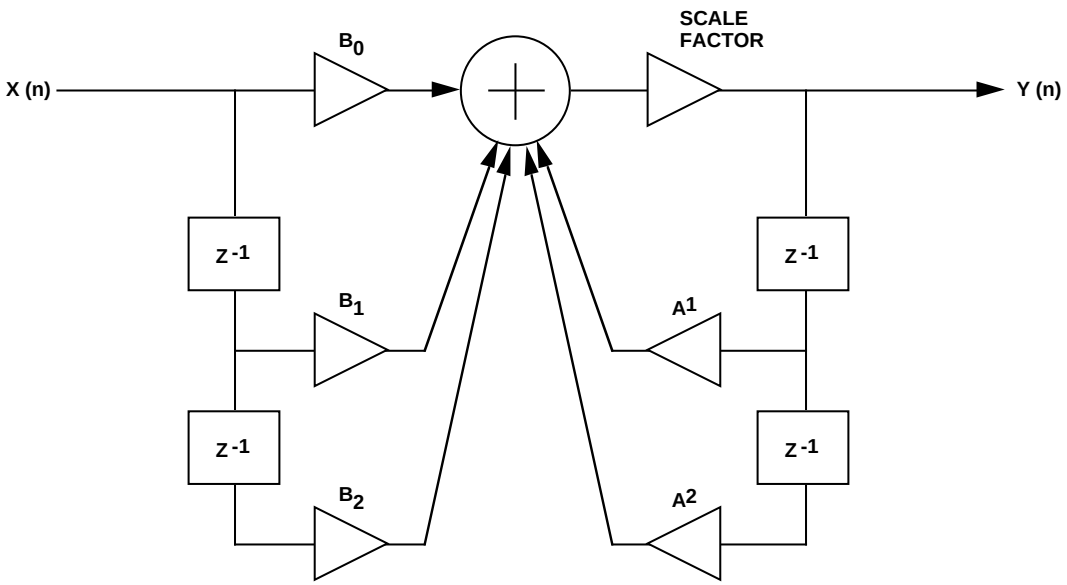

Figure 10.1 Second-Order Biquad IIR Filter Section

```
.MODULE biquad_sub;
{     Cascaded Biquad IIR Filter Subroutine
      Calling Parameters
            SR1 = input sample
            I0 –> delay line buffer
            L0 = 0
            I1 –> list of scale factors for each biquad section
            L1 = 0 (in the case of a single biquad)
            L1 = N/2 where N is the filter order
            I4 –> scaled coefficients b2,b1,b0,a2,a1, b2,b1,b0,a2,a1,
            L4 = 5 * N/2
            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)
            CNTR = number of biquad sections
      Return Values
            SR1 = output sample
            I0 –> inside delay line buffer
            I1 –> top of scale factor list
            I4 –> top of coefficients
```

```
      Altered Registers
            MX0,MX1,MY0,MR,SE,SR
      Computation Time
            8 * number of biquad sections + 10 cycles
      All coefficients and data values are assumed to be in 1.15 format. }

.CONST N = 3;                 {number of biquad sections, example: 3}
.CONST N_x_5 = 15;            {number of biquad sections times five}

.VAR/DM delayline[4];         {this is scratchpad memory}
.VAR/DM scalelist[N];         {initialize scale factor for each biquad}
.VAR/PM coefflist[N_x_5];     {init with filter coefficients for each biquad}

.ENTRY biquad;
biquad: I0=^delayline;
      DO sections UNTIL CE;
            SE=DM(I1,M2);
            MX0=DM(I0,M0), MY0=PM(I4,M4);                  {get x(n-2), b2}
            MR=MX0*MY0(SS), MX1=DM(I0,M0), MY0=PM(I4,M4);  {get x(n-1), b1}
            MR=MR+MX1*MY0(SS), MY0=PM(I4,M4);              {get b0}
            MR=MR+SR1*MY0(SS), MX0=DM(I0,M0), MY0=PM(I4,M4);{get y(n-2), a2}
            MR=MR+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M4);{get y(n-1), a1}
            DM(I0,M0)=MX1, MR=MR+MX0*MY0(RND);         {store x(n-1) as new x(n-2)}
sections: DM(I0,M0)=SR1, SR=ASHIFT MR1 (HI);           {store x(n) as new x(n-1)}
      DM(I0,M0)=MX0;
      DM(I0,M3)=SR1;
      RTS;
.ENDMOD;
```

Listing 10.1 Basic Biquad Filter Subroutine

10.2 MULTIPRECISION FILTERS

When your calculations require more precision than 16-bit arithmetic provides, you can still use ADSP-2100 family 16-bit DSPs. These DSPs have architectural features that make multiprecision calculations possible. Unlike other DSPs, the ADSP-2100 family processors let you multiply mixed-mode numbers (signed numbers x unsigned numbers). This chapter shows you how to take advantage of the multiprecision features of the ADSP-2100 Family.

10 Variations On IIR Biquad Filters

Multiprecision filters can have input data, output data, delay line data, or coefficients greater than 16-bits. While a 16-bit IIR filter is adequate for most applications, use multiprecision filters under the following conditions:

- The filter has a small passband or stopband relative to the sample rate
- You require more than 16-bit precision on the delay line, or the delay line and the coefficients

For example, consider a graphic equalizer where the sample rate is 44.1 kHz. The lowest equalizer band is centered at 31 Hz with stop bands at 0 Hz and 62 Hz. Although this filter is second-order, both the delay line and the coefficients require 32-bit arithmetic for the desired accuracy.

ADC converters, like the ones in the example graphic equalizer, are now available with 18-bit and 20-bit resolution. A digital filter with 32-bit accuracy preserves the arithmetic precision of the filter algorithm. This lets the DSP programmer maintain the signal-to-noise ratio delivered by the ADC.

10.2.1 Multiprecision Multiplication On ADSP-2100 Family DSPs

Multiprecision filters require multiprecision multiplication. ADSP-2100 Family DSPs include a 16-bit multiplier / accumulator with a 40-bit result register that is most efficient when working with 16-bit inputs. When you multiply 32-bit or 48-bit inputs, multiplication is accomplished by breaking the inputs into 16-bit components.

The code segment in Listing 10.2 is an example of multiplying two 32-bit inputs. Normally, when you multiply two 32-bit values, you get a 64-bit result. In this example, only 32 bits are required. To save instruction cycles and properly scale the partial products of each multiplication, the code shifts the contents of the multiply results registers “on-the-fly” to overwrite the lower 16 bits. This is more efficient than using processor cycles to write the contents to the barrel shifter.

Registers MX0 and MX1 contain the least significant word (lsw) and most significant word (msw) of one input. Registers MY0 and MY1 contain the least and most significant words of the second input.

```
MR=MX0*MY0(UU);   {multiply unsigned lsws}
MR0=MR1;          {shift product 16-bits right}
MR1=MR2;
MR=MR+MX1*MY0(SU);{multiply signed msws times unsigned lsws}
MR=MR+MX0*MY1(US);{and accumulate with shifted lsw product}
MR0=MR1;          {shift product 16-bits right}
MR1=MR2;
MR=MR+MX1*MY1(SS);{multiply signed msws and accumulate with}
                  {shifted intermediate product}
```

Listing 10.2 Double-Precision Multiply Routine

To generate the product, the code in Listing 10.2 performs the following operations, which are also shown in Figure 10.2.

1. It multiplies the unsigned contents of registers MX0 (lower 16 bits) and MY0 (lower 16 bits). The 32-bit product is put in the multiplier / accumulator results register MR (MR0, MR1, and MR2).

2. The contents of MR1 are shifted right 16 bits to MR0, this overwrites, or eliminates, the lower 16 bits. The contents of MR2 are shifted right 16 bits to MR1.

3. Next, it multiplies the signed contents of register MX1 (upper 16 bits) and unsigned contents of register MY0 (lower 16 bits), and accumulates this product with the contents of MR.

4. It then multiplies the unsigned contents of register MX0 (lower 16 bits) and the signed contents of register MY1 (upper 16 bits), and accumulates this product with the contents of MR.

5. This product is shifted 16 bits right, as described in step 2.

6. Finally, it multiplies the signed contents of registers MX1 (upper 16 bits) and MY1 (upper 16 bits), and accumulates this 32-bit product with the contents of MR.

This method assumes that the inputs to the filter are twos complement fractional where the most significant bit weighting is -2^0, or -1. Multiplying two 32-bit numbers generates a 64-bit product. With fractional products, the most significant 32 bits are saved; the lower 32 bits are used to properly scale the partial products and then are overwritten by MR1 (MR0=MR1) during the 16-bit right shifts. The 32-bit fractional product is in the accumulator with the msw in register MR1 and the lsw in register MR0.

10 Variations On IIR Biquad Filters

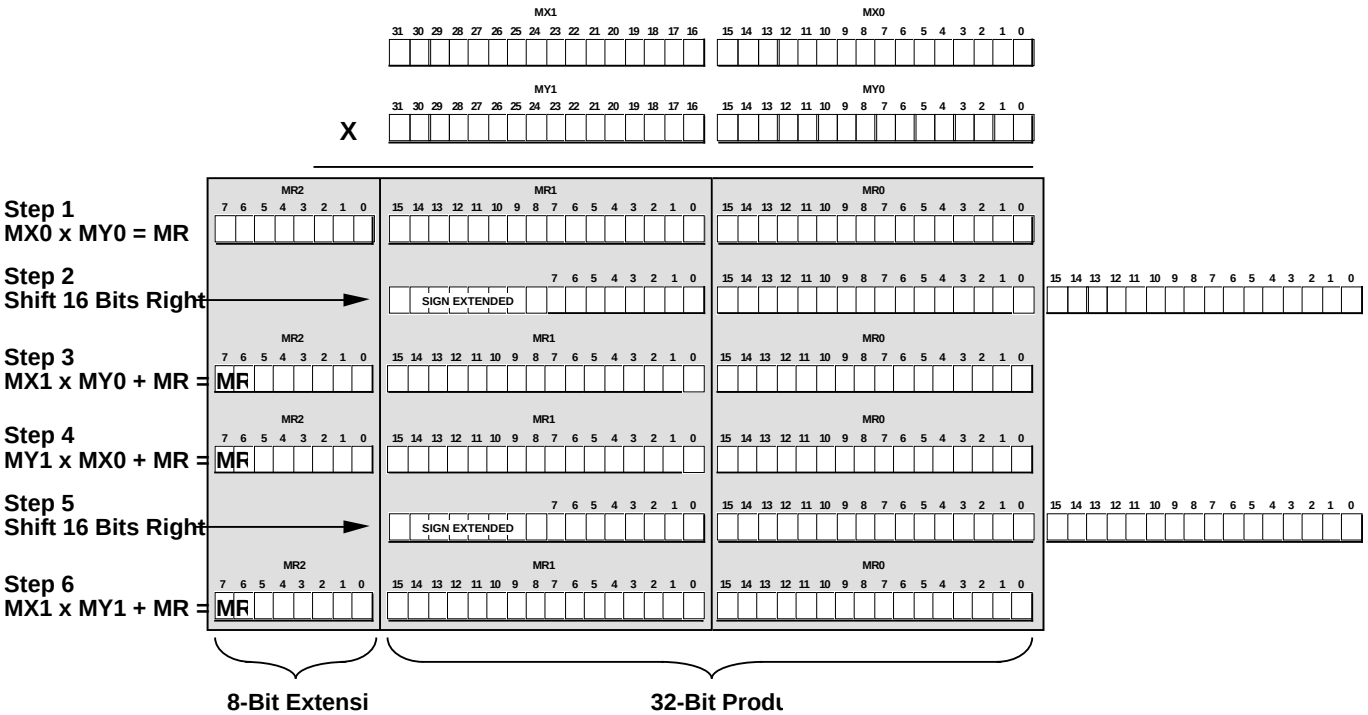

Figure 10.2 Multiprecision Multiplication Of 32-Bit Numbers

The technique used in Listing 10.2 can be applied to multiprecision IIR biquads. For example, an IIR biquad has five product terms that are accumulated:

```
y(n) = b0*x(n) + b1*x(n-1) + b2*x(n-2) + a1*y(n-1) + a2*y(n-2)
```

Instead of multiplying each 32-bit delay line input and coefficient pair one at a time, follow these steps to calculate a 32-bit delay line / coefficient biquad:

1. First, accumulate the lsw products of the five delay line / coefficient pairs.

2. Then, shift the accumulator 16-bits right.

3. Next, accumulate the msw / lsw and lsw / msw products.

4. Again, shift the accumulator 16-bits right.

5. Finally, accumulate the msw products.

In this method, the accumulator is shifted only twice for each biquad. As a result, this technique is more efficient than generating the full product for each pair and accumulating the full products.

10.2.2 Double-Precision Biquad

Listing 10.3 shows a double-precision IIR biquad subroutine with a 32-bit delay line and 32-bit coefficients. The calling routine initializes registers with the following items:

- 32-bit input sample
- Start addresses of the delay line
- Coefficient
- Scaling factor buffers
- Number of sections

Also, the calling routine sets the data address generator (DAG) length and modify registers to support modulo addressing. When you initialize registers in the calling routine, the biquad subroutine is reusable.

The code only clears the delay line buffer once before the first filter call. After the filter subroutine is called, the code stores delay line data in the buffer (lsw first, then msw). The filter coefficients are stored in the reverse order of the delay line data (msw first). After completing the calculations, the routine returns a 32-bit result in the shifter result registers SR1 and SR0.

This routine uses the circular addressing mode to retrieve and store data efficiently into the delay line. The starting address of the buffer in register I0 changes each time the filter is called, therefore, this address must be saved to memory if other routines use I0.

The core loop of Listing 10.3, starting at "`DO biq UNTIL CE:`", groups line of code according to function. The comment above each group describes its function.

The execution time for this code is [28*(N / 2) + 10] instruction cycles. A tenth-order filter executes in 150 instruction cycles, or in 9 µs using a DSP with 60 ns cycle time. The DSP can perform a tenth-order IIR filter on a signal sampled at more than 100 kSa / s.

10 Variations On IIR Biquad Filters

```
.MODULE/RAM/BOOT=0            dpiir_2p2z;
{
      Nth Order IIR Filter Constructed From N/2 Biquads of the Form:
            y(n)=b0*x(n)+b1*x(n-1)+b2*x(n-2)+a1*y(n-1)+a2*x(n-2)

      Where:
            x(n), x(n-1), x(n-2), y(n-1), y(n-2) are 32-bits
            b0, b1, b2, a1, a2 are 32-bits

      Calling Parameters:
            AX0=least significant word (LSW) of input
            AX1=most significant word (MSW) of input
            I0=start address of delay line in data memory stored in
                  LSW,MSW order [organized x(n-1),x(n-2),y(n-1),y(n-2)]
                  buffer length = (4*N/2)+4 where N is the order of the
                  filter - must be declared circular
            I4=start address of coefficients in program memory stored
                  in MSW,LSW order [organized b0,b1,b2,a1,a2]
                  buffer length = 10*N/2 - circular declaration not required
            I5=start address of coefficient scaling factors in program
                  memory (one/section)
                  buffer length = N/2 - circular declaration not required
            L0=(4*N/2)+4
            L4=0
            L5=0
            M0=-1, M1=1, M2=2, M3=-6
            M5=1, M6=2, M7=-9
            CNTR=N/2

      Return Values:
            I0=new start address of delay line must be saved to memory
            SR1=MSW result
            SR0=LSW result

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

      Computation Time:
            10 + (28 * N/2)

      Coefficients, delay line, input sample in 1.31 (Q.31) format
}
```

```
.ENTRY                  biq32;

biq32: SR1=AX1;                     {transfer input to SR}
            SR0=AX0;
            MODIFY(I4,M5);
            MY0=PM(I4,M6);          {read first coefficient}
            SE=PM(I5,M5);           {read first scaling factor}

            DO biq UNTIL CE;        {set up biquad loop}

                  {multiply/accumulate LSW*LSW}

                MR=SR0*MY0(UU),    MX0=DM(I0,M2), MY0=PM(I4,M6);
                MR=MR+MX0*MY0(UU), MX0=DM(I0,M2), MY0=PM(I4,M6);
                MR=MR+MX0*MY0(UU), MX0=DM(I0,M2), MY0=PM(I4,M6);
                MR=MR+MX0*MY0(UU), MX0=DM(I0,M3), MY0=PM(I4,M7);
                MR=MR+MX0*MY0(UU), MY0=PM(I4,M5);

                MR0=MR1;           {16-bit right shift}
                MR1=MR2;

                      {multiply/accumulate LSW*MSW, MSW*LSW}

                MR=MR+SR0*MY0(US), MY0=PM(I4,M5);
                MR=MR+SR1*MY0(SU), MX0=DM(I0,M1), MY0=PM(I4,M5);
                MR=MR+MX0*MY0(US), MX0=DM(I0,M1), MY0=PM(I4,M5);
                MR=MR+MX0*MY0(SU), MX0=DM(I0,M1), MY0=PM(I4,M5);
                MR=MR+MX0*MY0(US), MX0=DM(I0,M1), MY0=PM(I4,M5);
                MR=MR+MX0*MY0(SU), MX0=DM(I0,M1), MY0=PM(I4,M5);
                MR=MR+MX0*MY0(US), MX0=DM(I0,M1), MY0=PM(I4,M5);
                MR=MR+MX0*MY0(SU), MX0=DM(I0,M1), MY0=PM(I4,M5);
                MR=MR+MX0*MY0(US), MX0=DM(I0,M3), MY0=PM(I4,M7);
                MR=MR+MX0*MY0(SU), MY0=PM(I4,M6);

                MR0=MR1;           {16-bit right shift}
                MR1=MR2;
```

(listing continues on next page)

```
                    {multiply/accumulate MSW*MSW}

                MR=MR+SR1*MY0(SS), MX0=DM(I0,M2), MY0=PM(I4,M6);
                MR=MR+MX0*MY0(SS), MX0=DM(I0,M2), MY0=PM(I4,M6);
                MR=MR+MX0*MY0(SS), MX0=DM(I0,M2), MY0=PM(I4,M6);
                MR=MR+MX0*MY0(SS), MX0=DM(I0,M3), MY0=PM(I4,M6);
                MR=MR+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M5);

                    {apply scale factor, read next scale factor}

                SR=ASHIFT MR1 (HI), MY0=PM(I4,M6);
                SR=SR OR LSHIFT MR0 (LO), SE=PM(I5,M5);

                    {store new y(n) to delay line}

                DM(I0,M1)=SR0;
biq:        DM(I0,M1)=SR1;

                MODIFY(I0,M2);
                DM(I0,M1)=AX0;    {store original input to delay line}
                DM(I0,M0)=AX1;

                RTS;

.ENDMOD;
```

Listing 10.3 Double-Precision IIR Biquad Subroutine

Listing 10.4 is an optimized version of Listing 10.3. In Listing 10.4, the lsw multiply/accumulates are eliminated, reducing the feedback accuracy from 32-bits to 31-bits. In most cases, the code in listing 10.4 provides adequate performance.

The execution time is [20*(N/2) + 9] instruction cycles. A tenth-order filter executes in 109 instruction cycles, or in 6.54 μs using a DSP with 60 ns cycle time. The DSP can perform a tenth-order IIR filter on a signal sampled at more than 150 kSa/s.

```
.MODULE/RAM/BOOT=0            dpiir_2p2z_optimized;
{
      IIR Filter of the Form:

            y(n)=b0*x(n)+b1*x(n-1)+b2*x(n-2)+a1*y(n-1)+a2*x(n-2)

      Where:
            x(n), x(n-1), x(n-2), y(n-1), y(n-2) are 32-bits
            b0, b1, b2, a1, a2 are 32-bits

      Calling Parameters:
            AX0=least significant word (LSW) of input
            AX1=most significant word (MSW) of input
            I0=start address of delay line in data memory stored in
                  LSW,MSW order [organized x(n-1),x(n-2),y(n-1),y(n-2)]
                  buffer length = (4*N/2)+4 where N is the order of the
                  filter - must be declared circular
            I4=start address of coefficients in program memory stored
                  in MSW,LSW order [organized b0,b1,b2,a1,a2]
                  buffer length = 10*N/2 - circular declaration not required
            I5=start address of coefficient scaling factors (one/section)
                  buffer length = N/2 - circular declaration not required
            L0=(4*N/2)+4
            L4=0
            L5=0
            M0=-1, M1=1, M2=2, M3=-6
            M5=1, M6=2, M7=-9
            CNTR=N/2

      Return Values:
            I0=new start address of delay line must be saved to memory
            SR1=MSW result
            SR0=LSW result

      Altered Registers:
            MX0,MY0,MY1,MR,SE,SR

      Computation Time:
            9 + (20 * N/2)

      Coefficients, delay line, input sample in 1.31 (Q.31) format
}
```

(listing continues on next page)

10 Variations On IIR Biquad Filters

```
ENTRY       opt_biq32;

opt_biq32:

      SR1=AX1;                {transfer input to SR}
      SR0=0;
      MY0=PM(I4,M5);          {read first coefficient}
      MY1=PM(I4,M5);          {read second coefficient}
      DO biq UNTIL CE;        {set up biquad loop}


                        {multiply/accumulate LSW*LSW, read scale factor}

                  MR=SR0*MY0(US),    SE=PM(I5,M5);
                  MR=MR+SR1*MY1(SU), MX0=DM(I0,M1), MY0=PM(I4,M5);
                  MR=MR+MX0*MY0(US), MX0=DM(I0,M1), MY0=PM(I4,M5);
                  MR=MR+MX0*MY0(SU), MX0=DM(I0,M1), MY0=PM(I4,M5);
                  MR=MR+MX0*MY0(US), MX0=DM(I0,M1), MY0=PM(I4,M5);
                  MR=MR+MX0*MY0(SU), MX0=DM(I0,M1), MY0=PM(I4,M5);
                  MR=MR+MX0*MY0(US), MX0=DM(I0,M1), MY0=PM(I4,M5);
                  MR=MR+MX0*MY0(SU), MX0=DM(I0,M1), MY0=PM(I4,M5);
                  MR=MR+MX0*MY0(US), MX0=DM(I0,M3), MY0=PM(I4,M7);
                  MR=MR+MX0*MY0(SU), MY0=PM(I4,M6);

                  MR0=MR1;          {16-bit right shift}
                  MR1=MR2;

                        {multiply/accumulate LSW*MSW, MSW*LSW}

                  MR=MR+SR1*MY0(SS), MX0=DM(I0,M2), MY0=PM(I4,M6);
                  MR=MR+MX0*MY0(SS), MX0=DM(I0,M2), MY0=PM(I4,M6);
                  MR=MR+MX0*MY0(SS), MX0=DM(I0,M2), MY0=PM(I4,M6);
                  MR=MR+MX0*MY0(SS), MX0=DM(I0,M3), MY0=PM(I4,M6);
                  MR=MR+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M5);

                        {apply scale factor, store y(n) to delay line}
```

```
                SR=LSHIFT MR0 (LO), MY1=PM(I4,M5);
                DM(I0,M1)=SR0, SR=SR OR LSHIFT MR1 (HI);
biq:        DM(I0,M1)=SR1;

            MODIFY(I0,M2);
            DM(I0,M1)=AX0;          {store inputs to delay line}
            DM(I0,M0)=AX1;

            RTS;

.ENDMOD;
```

Listing 10.4 Optimized Double-Precision IIR Biquad Subroutine

10.2.3 Half, Double-Precision Biquad

The half, double-precision IIR Biquad subroutine in Listing 10.5 maintains a 32-bit delay line, but reduces the coefficients to 16-bit precision. This routine is useful when a 16-bit biquad (as shown in Listing 10.1) has the desired accuracy, but lacks the necessary stability.

The execution time is [16*(N / 2) + 9] instruction cycles. A tenth-order filter executes in 89 instruction cycles, or in 5.34 µs using a DSP with 60 ns cycle time. The DSP can perform a tenth order IIR filter on a signal sampled at more than 180 kSa / s.

```
.MODULE/RAM/BOOT=0          dpiir_2p2z_32_16;

{
      IIR Filter of the Form:
            y(n)=b0*x(n)+b1*x(n-1)+b2*x(n-2)+a1*y(n-1)+a2*x(n-2)

      Where:
            x(n), x(n-1), x(n-2), y(n-1), y(n-2) are 32-bits
            b0, b1, b2, a1, a2 are 16-bits

      Calling Parameters:
            AX0=least significant word (LSW) of input
            AX1=most significant word (MSW) of input
            I0=start address of delay line in data memory stored in
                  LSW,MSW order [organized x(n-1),x(n-2),y(n-1),y(n-2)]
                  buffer length = (4*N/2)+4 where N is the filter order
                  buffer must be declared circular
```

(listing continues on next page)

```
            I4=start address of coefficients in program memory
                  [organized b0,b1,b2,a1,a2] - buffer length = 5*N/2
                  does not require circular declaration
            I5=start address of coefficient scaling factors in program
                  memory (one/section) - buffer length = N/2
                  does not require circular declaration
            L0=(4*N/2)+4
            L4=0
            L5=0
            M0=-1, M1=1, M2=2, M3=-5
            M5=1, M7=-4
            CNTR=N/2

      Return Values:
            I0=new start address of delay line must be saved to memory
            SR1=MSW result
            SR0=LSW result

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

      Computation Time:
            9 + (16 * number of sections)

      Coefficients in 1.15 (Q.15) format
      Delay line, input sample in 1.31 (Q.31) format
}

.ENTRY                  biq_3216;

biq_3216:   SR1=AX1;                {copy input to SR}
                  SR0=AX0;
                  MY0=PM(I4,M5);    {read first coefficient}
                  SE=PM(I5,M5);     {read first scaling factor}

                  DO biq UNTIL CE;  {set up biquad loop}

                        {multiply/accumulate LSW delay line * coef}
```

```
                MR=SR0*MY0(US),    MX0=DM(I0,M2), MY0=PM(I4,M5);
                MR=MR+MX0*MY0(US), MX0=DM(I0,M2), MY0=PM(I4,M5);
                MR=MR+MX0*MY0(US), MX0=DM(I0,M2), MY0=PM(I4,M5);
                MR=MR+MX0*MY0(US), MX0=DM(I0,M3), MY0=PM(I4,M7);
                MR=MR+MX0*MY0(US), MY0=PM(I4,M5);

                MR0=MR1;          {16-bit right shift}
                MR1=MR2;

                      {multiply/accumulate MSW delay line * coef}

                MR=MR+SR1*MY0(SS), MX0=DM(I0,M2), MY0=PM(I4,M5);
                MR=MR+MX0*MY0(SS), MX0=DM(I0,M2), MY0=PM(I4,M5);
                MR=MR+MX0*MY0(SS), MX0=DM(I0,M2), MY0=PM(I4,M5);
                MR=MR+MX0*MY0(SS), MX0=DM(I0,M3), MY0=PM(I4,M5);
                MR=MR+MX0*MY0(SS);

                      {scale factor correction}

                SR=ASHIFT MR1 (HI), MY0=PM(I4,M5);
                SR=SR OR LSHIFT MR0 (LO), SE=PM(I5,M5);

                DM(I0,M1)=SR0;    {store y(n) to delay line}
biq:        DM(I0,M1)=SR1;

            MODIFY(I0,M2);
            DM(I0,M1)=AX0;          {store input to delay line}
            DM(I0,M0)=AX1;

            RTS;

.ENDMOD;
```

Listing 10.5 Half, Double-Precision IIR Biquad Subroutine

10 Variations On IIR Biquad Filters

10.2.4 Half, Triple-Precision Biquad

The half, triple-precision IIR Biquad subroutine in Listing 10.6 has a 48-bit delay line and increases the coefficients to 32-bit precision. This subroutine provides a filter resolution that is usually reserved for floating-point arithmetic. The code in Listing 10.6 gives you 32-bit, floating-point precision on a 16-bit fixed-point DSP. Filters with extremely narrow pass or reject bands may require this precision.

The execution time is [45*(N / 2) + 12] instruction cycles. A tenth-order filter executes in 237 instruction cycles, or in 14.22 μs using a DSP with 60 ns cycle time. The DSP can perform a tenth-order IIR filter on a signal sampled at more than 70 kSa / s.

```
.MODULE/RAM/BOOT=0            half_triple_precision_iir_biquad;

{
      IIR Filter of the Form:
            y(n)=b0*x(n)+b1*x(n-1)+b2*x(n-2)+a1*y(n-1)+a2*x(n-2)

      Where:
            x(n), x(n-1), x(n-2), y(n-1), y(n-2) are 48-bits
            b0, b1, b2, a1, a2 are 32-bits

      Calling Parameters:
            AR=lower word of input
            AX0=middle word of input
            AX1=upper word of input
            I0=start address of delay line lower word in data memory stored in
                  [x(n-1),x(n-2),y(n-1),y(n-2)] order buffer length = (2*N/2)+2
                  where N is filter order must be declared circular
            I1=start address of delay line middle word in data memory stored in
                  [x(n-1),x(n-2),y(n-1),y(n-2)] order buffer length = (2*N/2)+2
                  where N is filter order must be declared circular
            I2=start address of delay line upper word in data memory stored in
                  [x(n-1),x(n-2),y(n-1),y(n-2)] order buffer length = (2*N/2)+2
                  where N is filter order must be declared circular
            I4=start address of coefficients in program memory stored in MSW,LSW
                  order [organized b0,b1,b2,a1,a2] buffer length = 10*N/2
                  does not require circular declaration
            I5=start address of coefficient scaling factors in program
                  memory (one/section) - buffer length = N/2
                  does not require circular declaration
```

```
            L0=(2*N/2)+2
            L1=(2*N/2)+2
            L2=(2*N/2)+2
            L4=0
            L5=0
            M0=0, M1=1, M3=-3
            M5=1, M6=2, M7=-9
            CNTR=N/2

      Return Values:
            I0,I1,I2 -> new start address of delay line must be saved to memory
            SR1=upper result
            SR0=middle result
            AR=lower result

      Altered Registers:
            MX0,MY0,MR,SE,SR,SI,AF

      Computation Time:
            12 + (45 * number of sections)

      Coefficients in 1.31 (Q.31) format
      Delay line, input sample, result in 1.47 (Q.47) format
}

.ENTRY                  biq_3P;

biq_3P:

            SR1=AX1;                {SR = upper/middle word}
            AF=PASS AR, SR0=AX0;    {AF = lower word}
            MODIFY(I4,M5);
            MY0=PM(I4,M6);          {read first coefficient}

            DO biq UNTIL CE;        {set up biquad loop}

                              {multiply/accumulate - lower delay*lower coef}
```

(listing continues on next page)

```
            MR=AR*MY0(UU),     MX0=DM(I0,M1), MY0=PM(I4,M6);
            MR=MR+MX0*MY0(UU), MX0=DM(I0,M1), MY0=PM(I4,M6);
            MR=MR+MX0*MY0(UU), MX0=DM(I0,M1), MY0=PM(I4,M6);
            MR=MR+MX0*MY0(UU), MX0=DM(I0,M3), MY0=PM(I4,M7);
            MR=MR+MX0*MY0(UU), MY0=PM(I4,M5);
            MR=MR(RND), SE=PM(I5,M5);

                  {shift accumulator 16-bits right}

            MR0=MR1;
            MR1=MR2;


{multiply/accumulate - middle delay*lower coef, lower delay*upper coef}

            MR=MR+AR*MY0(US), MY0=PM(I4,M5);
            MR=MR+SR0*MY0(UU), MX0=DM(I0,M1), MY0=PM(I4,M5);
            MR=MR+MX0*MY0(US), MX0=DM(I1,M1), MY0=PM(I4,M5);
            MR=MR+MX0*MY0(UU), MX0=DM(I0,M1), MY0=PM(I4,M5);
            MR=MR+MX0*MY0(US), MX0=DM(I1,M1), MY0=PM(I4,M5);
            MR=MR+MX0*MY0(UU), MX0=DM(I0,M1), MY0=PM(I4,M5);
            MR=MR+MX0*MY0(US), MX0=DM(I1,M1), MY0=PM(I4,M5);
            MR=MR+MX0*MY0(UU), MX0=DM(I0,M3), MY0=PM(I4,M5);
            MR=MR+MX0*MY0(US), MX0=DM(I1,M3), MY0=PM(I4,M7);
            MR=MR+MX0*MY0(UU), MX0=DM(I0,M1), MY0=PM(I4,M5);
            MR=MR(RND);

                  {shift accumulator 16-bits right}

            MR0=MR1;
            MR1=MR2;

{multiply/accumulate - upper delay*lower coef, middle delay*upper coef}

            MR=MR+SR0*MY0(US), MY0=PM(I4,M5);
            MR=MR+SR1*MY0(SU), MX0=DM(I1,M1), MY0=PM(I4,M5);
            MR=MR+MX0*MY0(US), MX0=DM(I2,M1), MY0=PM(I4,M5);
            MR=MR+MX0*MY0(SU), MX0=DM(I1,M1), MY0=PM(I4,M5);
            MR=MR+MX0*MY0(US), MX0=DM(I2,M1), MY0=PM(I4,M5);
            MR=MR+MX0*MY0(SU), MX0=DM(I1,M1), MY0=PM(I4,M5);
            MR=MR+MX0*MY0(US), MX0=DM(I2,M1), MY0=PM(I4,M5);
            MR=MR+MX0*MY0(SU), MX0=DM(I1,M3), MY0=PM(I4,M5);
            MR=MR+MX0*MY0(US), MX0=DM(I2,M3), MY0=PM(I4,M7);
            MR=MR+MX0*MY0(SU), MX0=DM(I1,M1), MY0=PM(I4,M6);
```

```
                    {shift accumulator 16-bits right}

                SR0=MR0;
                MR0=MR1;
                MR1=MR2;

                    {multiply/accumulate - upper delay*upper coef}

                MR=MR+SR1*MY0(SS), MX0=DM(I2,M1), MY0=PM(I4,M6);
                MR=MR+MX0*MY0(SS), MX0=DM(I2,M1), MY0=PM(I4,M6);
                MR=MR+MX0*MY0(SS), MX0=DM(I2,M1), MY0=PM(I4,M6);
                MR=MR+MX0*MY0(SS), MX0=DM(I2,M3), MY0=PM(I4,M6);
                MR=MR+MX0*MY0(SS), MX0=DM(I2,M1), MY0=PM(I4,M5);


      {apply scale factor to biquad result, write scaled result to delay line}

scale_it:   SR=LSHIFT SR0 (LO), MY0=PM(I4,M6);
                AR=SR0;
                SR=LSHIFT SR1 BY 0 (LO);
                DM(I0,M1)=AR, SR=SR OR LSHIFT MR0 (LO);
                DM(I1,M1)=SR0, SR=SR OR ASHIFT MR1 (HI);
biq:        DM(I2,M1)=SR1;

                    {store original inputs to delay line}

            MODIFY(I0,M1);
            MODIFY(I1,M1);
            MODIFY(I2,M1);
            DM(I2,M0)=AX1;
            DM(I1,M0)=AX0, AR=PASS AF;
            DM(I0,M0)=AR;
            RTS;        {return}

.ENDMOD;
```

Listing 10.6 Half, Triple-Precision IIR Biquad Subroutine

10 Variations On IIR Biquad Filters

10.3 OPTIMIZED 16-BIT BIQUADS

If 16-bit accuracy is adequate for your application, you can use the two subroutines included in this section. While these routines are similar to the program in Listing 10.1, they are optimized to require fewer instruction cycles to execute.

The program in Listing 10.7 provides the identical results to the program in Listing 10.1, but, in Listing 10.7, it executes the biquad loop in six instruction cycles rather the seven. This decrease results from the ADSP-2100 Family's modulo addressing capability.

To optimize the filter, the program uses a circular buffer that contains input data and output data. If you carefully arrange the data in the delay line and use the modulo addressing of the data address generators, you will have an efficient addressing scheme that lets you use any address in the circular buffer as the starting address. Modulo addressing only applies to the delay line; it is not required for the coefficient and scale factor buffers. Modulo addressing is also used in the multiprecision IIR filters in this chapter. You must save the delay line address pointer, index register I0, to memory after each call if I0 is used elsewhere in the program.

Figure 10.3 is a memory map that illustrates modulo addressing. This figure shows three time intervals. In the first interval (t), addresses 1, 3, and, 5 contain the oldest data. Because the oldest data is not needed during the second interval (t + 1), new data is written into those locations. Data in addresses 0, 2, and 4 is preserved for the second interval, but it becomes the oldest data. For example, $X_0(n-1)$ from the first interval becomes $X_0(n-2)$ in the second interval. In the third sample interval, the oldest data from the second interval is overwritten. The cycle continues for every new sample interval. This method lets save instruction cycles because you move the pointers to the circular buffer, rather than move the buffer contents. You can also use the modulo addressing advantages for feedback values in the delay line or past results. For example, past result y(n-1) is used as y(n-2) during the next sample interval without moving to another memory location.

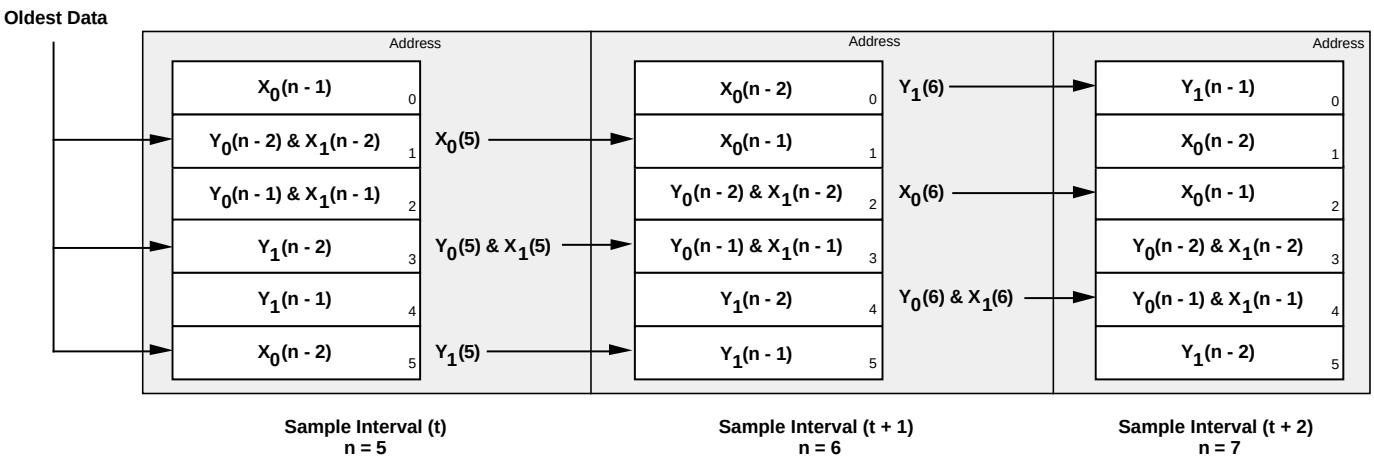

Figure 10.3 Modulo Addressing & Delay Line Data

The execution time is [7*(N / 2) + 6] instruction cycles. A tenth-order filter executes in 41 instruction cycles, or in 2.46 µs using a DSP with 60 ns cycle time. The DSP can perform a tenth-order IIR filter on a signal sampled at more than 400 kSa / s.

```
.MODULE/boot=0 optimized_biquad_sub;

{
      Optimized Cascaded Biquad IIR Filter Subroutine (Direct Form I)
      Calling Parameters
            SR1 = input sample
            I0 –> delay line buffer in data memory
                  x(n-2),x(n-1),y(n-2),y(n-1) order
                  buffer length = 2N+2 where N is the filter order
            L0 = 2N+2 - circular buffer declaration required
            I4 –> scaled coefficients in program memory b2,b1,b0,a2,a1 order
                  buffer length = 5*N/2 - circular buffer not required
            L4 = 0
            I5 –> coefficient scale factors in program memory
                  buffer length = N - circular buffer not required
            L5 = 0
             M0 = 0
            M1,M4 = 1
            M3,M5 = -1
            CNTR = N/2
```

(listing continues on next page)

```
      Return Values
            SR1 = output sample
            I0 –> new delay line buffer start address must be saved to memory

      Altered Registers

            MX0,MY0,MR,SR,SE

      Computation Time

            7 * number of biquad sections + 6 cycles

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

.ENTRY      optbiq;

optbiq:

            SE=PM(I5,M4);                 {read coefficient sclaing factor}
            MX0=DM(I0,M1), MY0=PM(I4,M4); {x=x(n-2), y=b2}
            DO sections UNTIL CE;
                  MR=MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M4);
                                          {mult, x=x(n-1), y=b1}
                  MR=MR+MX0*MY0(SS), MY0=PM(I4,M4);
                                          {mac, y=b0}
                  MR=MR+SR1*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M4);
                                          {mac, x=y(n-2), y=a2}
                  MR=MR+MX0*MY0(SS), MX0=DM(I0,M3), MY0=PM(I4,M4);
                                          {mac, x=y(n-1), y=a1}
                  MR=MR+MX0*MY0(RND), MX0=DM(I0,M0), MY0=PM(I4,M4);
                                          {mac, x=next x(n-2), y=next b2}
                  DM(I0,M1)=SR1, SR=ASHIFT MR1 (HI);
                                          {x(n)->new x(n-1), scale result}
sections:   SR=SR OR LSHIFT MR0 (LO), SE=PM(I5,M4);
                                          {continue scaling, new scale factor}
            MX0=DM(I0,M1);                {dummy read to modify pointer}
            DM(I0,M1)=SR1;                {store last result into delay line}
                                          {must return new I0}
            RTS;

.ENDMOD;
```

Listing 10.7 Optimized Basic Biquad Filter Subroutine

Listing 10.8 further optimizes the filter because the coefficient scaling factor for each biquad is the same. This eliminates the need to read the scaling factor from program memory.

The biquad loop in Listing 10.7 includes a double-precision shift for scaling factor correction. Listing 10.8 performs a single-precision shift. Since the scaling factor typically implies a one-bit left shift, a single precision shift yields a zero as the least significant bit. Therefore, the result from Listing 10.8 only has 15-bit accuracy. This may change filter performance if the filter was designed for 90 dB or greater stopband attenuation, which may effect the filter's stability.

The execution time is [6*(N / 2) + 5] instruction cycles. A tenth-order filter executes in 35 instruction cycles, or in 2.1 µs using a DSP with 60 ns cycle time. The DSP can perform a tenth-order IIR filter on a signal sampled at more than 470 kSa / s.

```
.MODULE/boot=0 optimized_biquad_sub;
{
      Optimized Cascaded Biquad IIR Filter Subroutine (Direct Form I)

      Calling Parameters
            SR1 = input sample
            I0 –> delay line buffer in data memory
                  x(n-2),x(n-1),y(n-2),y(n-1) order
                  buffer length = 2N+2 where N is the filter order
                  buffer must be declared circular
            L0 = 2N+2
            I4 –> scaled coefficients in program memory
                  b2,b1,b0,a2,a1 order
                  buffer length = 5*N/2 - circular buffer not required
            L4 = 2.5 * filter order –or– 5 * number of biquad sections
            M0 = 0
            M1,M4 = 1
            M3,M5 = -1
            CNTR = number of biquad sections
            SE = shift count (must be same for all biquad sections)

      Return Values
            SR1 = output sample
            I0 –> new delay line buffer start address must be stored to memory
```

(listing continues on next page)

```
      Altered Registers
            MX0,MY0,MR,SR

      Computation Time
            6 * number of biquad sections + 5 cycles

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

.ENTRY optbiq;

optbiq:

            MX0=DM(I0,M1), MY0=PM(I4,M4); {x=x(n-2), y=b2}
            DO sections UNTIL CE;
                  MR=MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M4);
                                                {mult, x=x(n-1), y=b1}

                  MR=MR+MX0*MY0(SS), MY0=PM(I4,M4);
                                                {mac, y=b0}
                  MR=MR+SR1*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M4);
                                                {mac, x=y(n-2), y=a2}
                  MR=MR+MX0*MY0(SS), MX0=DM(I0,M3), MY0=PM(I4,M4);
                                                {mac, x=y(n-1), y=a1}
                  MR=MR+MX0*MY0(RND), MX0=DM(I0,M0), MY0=PM(I4,M4);
                                                {mac, x=next x(n-2), y=next b2}
sections:   DM(I0,M1)=SR1, SR=ASHIFT MR1 (HI);
                                                {x(n)->new x(n-1), scale result}
            MX0=DM(I0,M1), MY0=PM(I4,M5);
            DM(I0,M1)=SR1;                      {store last result into delay line}
                                                {must return new I0}
            RTS;

.ENDMOD;
```

Listing 10.8 Second-Level Optimization Of Basic Biquad Filter

10.4 CONCLUSION

This chapter provides you with precision-related options when designing digital IIR filters. Depending on the filter characteristics, 15-bit, 16-bit, 32-bit or 48-bit, precision may be required in the delay line for an IIR biquad to function correctly. Filter coefficients may also require 32-bit precision to ensure filter stability. Higher precision IIR filters are available at the expense of instruction cycles. Table 10.1 lists the subroutines listed in this chapter, their characteristics, and performance.

Listing #	*Filename*	*delay line*	*coefficients*	*performance*
		# bits	*# bits*	*# cycles*
8	iir1516.dsp	15-bits	16-bits	6*(N / 2) + 5
7	iir1616.dsp	16-bits	16-bits	7*(N / 2) + 6
5	iir3216.dsp	32-bits	16-bits	16*(N / 2) + 9
4	iir3132.dsp	31-bits	32-bits	20*(N / 2) + 9
3	iir3232.dsp	32-bits	32-bits	28*(N / 2) + 10
6	iir4832.dsp	48-bits	32-bits	45*(N / 2) + 12

where N = filter order

Table 10.1 Filter Routine Characteristics Summary

Other variations of IIR biquad sections are possible. You can achieve more precision by increasing the feedback variables or coefficients to 64-bits. Existing 16-bit or 32-bit subroutines can be optimized to exclude those multiplications when the coefficients are known to be zero.

Coefficients for these filters can be generated by many digital filter design software tools. These packages, however, do not determine if a 16-bit or 32-bit filter is required for proper convergence. A few of these companies are listed below.

Momentum Data Systems	(714) 557-6884
Hyperception	(214) 343-8525
The Athena Group	(904) 371-2567
Signalogic	(214) 343-0069