6 One-Dimensional FFTs

6.3 BLOCK FLOATING-POINT SCALING

Block floating-point scaling is used to maximize the dynamic range of a fixed-point data field. The block floating-point system is a hybrid between fixed-point and floating-point systems. Instead of each data word having its own exponent, block floating-point format assumes the same exponent for an entire block of data.

The initial input data contains enough guard bits to ensure that no overflow occurs in the first FFT stage. During each stage of the FFT calculation, bit growth can occur. This bit growth can result in magnitude bits replacing guard bits. Because the stage output data is used as input data for the next stage, these guard bits must be replaced; otherwise, an output of the next stage might overflow. In a conditional block floating-point FFT, bit growth is monitored in each stage calculation. When the stage is complete, the output data of the entire stage is shifted to replace any lost guard bits.

Because a radix-2 butterfly calculation has the potential for two bits of growth, SB (the block floating-point exponent register) is initialized to –2. This sets up the ADSP-2100 block floating-point compare logic to detect any data with more than 13 bits of magnitude (or fewer than three sign bits). After each butterfly calculation, the EXPADJ instruction determines if bit growth occurred by checking the number of guard bits. For example, the value 1111 0000 0000 0000 has an exponent of –3. The value 0111 1111 1111 1111 has an exponent of zero (no guard bits). If a butterfly result has an exponent larger than the value in SB, bit growth into the guard bits has occurred, and SB is assigned the larger exponent (if it has not already been changed by bit growth in a previous butterfly of the same stage). Therefore, at the end of each stage, SB contains the exponent of the largest butterfly result(s). If no bit growth occurred, SB is not changed. An example of how the EXPADJ instruction affects SB is shown below (assume SB = –2 initially).

Butterfly Output Data	*Value of SB After EXPADJ Performed on Data*
1111 0000 0000 0000	SB= –2
1110 0000 0000 0000	SB= –2
1100 0000 0000 0000	SB= –1
1110 0000 0000 0000	SB= –1
1000 0000 0000 0000	SB= 0

The *dit_radix-2_bfp_adjust* routine is shown in Listing 6.15, on the next page. This routine performs block floating-point scaling on the outputs of each stage except the last of the DIT FFT. It can be modified for the DIF FFT subroutine by replacing *inplacereal* references with *inplacedata.*

Because guard bits only need to be replaced to ensure that an output of the next stage does not overflow, the subroutine first checks to see if the block of data is the output of the last stage. If it is, no shifting is needed and the subroutine returns. If the data block is not the output of the last stage, shifting is necessary only if SB is not –2 (indicating that bit growth into guard bits occurred). If SB is –2, no bit growth occurred, so the subroutine returns.

If bit growth occurred, shifting is needed. The subroutine determines the amount to shift from the value of SB. The data can grow by either one or two bits for each stage; therefore, if bit growth occurred, SB must be either –1 or zero. If SB is –1, the data block is shifted right one bit. If SB is not –1, it must be zero. In this case, the data block is shifted right two bits. When shifting is complete, the block exponent is updated by the shifted amount (one or two).

In this routine, shifting to the right is performed through multiplication rather than shift instructions. Multiplication by an appropriate power of two gives a shifted result. For example, to shift a number two bits to the right, the number is multiplied by H#0200. In multiplication, the product can be rounded to preserve LSB information, whereas in shifting, this information is merely lost. Multiplication thus minimizes noise.

```
.MODULE     dit_radix_2_bfp_adjust;

{           Calling Parameters
               Radix-2 DIT FFT stage results in inplacereal and inplaceimag

            Return Parameters
               inplacereal and inplaceimag adjusted for bit growth

            Altered Registers
               I0,I1,AX0,AY0,AR,MX0,MY0,MR,CNTR

            Altered Memory
               inplacereal, inplaceimag, blk_exponent
}

.CONST      Ntimes2 = 2048;
.EXTERNAL   inplacereal, blk_exponent;    {Begin declaration section}

.ENTRY      bfp_adj;

bfp_adj:    AY0=CNTR;                     {Check for last stage}
            AR=AY0-1
            IF EQ RTS;                    {If last stage, return}
            AY0=-2;
            AX0=SB;
            AR=AX0-AY0;                   {Check for SB=-2}
            IF EQ RTS;                    {IF SB=-2, no bit growth, return}
            I0=^inplacereal;              {I0=read pointer}
            I1=^inplacereal;              {I1=write pointer}
            AY0=-1;
            MY0=H#4000;                   {Set MY0 to shift 1 bit right}
            AR=AX0-AY0,MX0=DM(I0,M1);     {Check if SB=-1; Get first sample}
            IF EQ JUMP strt_shift;        {If SB=-1, shift block data 1 bit}
            AX0=-2;                       {Set AX0 for block exponent update}
            MY0=H#2000;                   {Set MY0 to shift 2 bits right}
strt_shift: CNTR=Ntimes2 - 1;             {initialize loop counter}
            DO shift_loop UNTIL CE;           {Shift block of data}
               MR=MX0*MY0(RND),MX0=DM(I0,M1); {MR=shifted data,MX0=next value}
shift_loop:    DM(I1,M1)=MR1;                 {Unshifted data=shifted data}
            MR=MX0*MY0(RND);              {Shift last data word}
            AY0=DM(blk_exponent);         {Update block exponent and}
            DM(I1,M1)=MR1,AR=AY0-AX0;     {store last shifted sample}
            DM(blk_exponent)=AR;
            RTS;
.ENDMOD;
```

Listing 6.15 Radix-2 Block Floating-Point Scaling Routine