SECTION 6

Real-Valued Input FFT Algorithm

"Data acquisition on the DSP96002 is truly parallel with CPU instruction execution."

A real-valued input FFT is a special case of the complex FFT where all imaginary components in the input are zero. Under this condition, input sequence is real, and the time sequence has a symmetric Fourier transform in the frequency domain. Only half of the frequency sequence needs to be computed for real-valued input FFTs or real FFTs. Recall the definition of the DFT:

$$X(k) = \sum_{r=0}^{N-1} x(r)e^{-j(2\pi rk)/N} \qquad k = 0, 1, \ldots N-1 \qquad \text{Eqn. 6-1}$$

If x(r) is real,

$$X^*(-k) = \sum_{r=0}^{N-1} x(r)e^{j(2\pi rk)/N} = \sum_{r=0}^{N-1} x(r)e^{-j(2\pi rk)/N} = X(k) \qquad \text{Eqn. 6-2}$$

and

$$X^*(N-k) = \sum_{r=0}^{N-1} x^*(r)e^{*(-j)(2\pi r(N-k))/N} = \sum_{r=0}^{N-1} x(r)e^{-j(2\pi rk)/N} = X(k) \qquad \text{Eqn. 6-3}$$

6.1 Real-Valued Input FFT Algorithm 1

6.1.1 Bergland Algorithm

This algorithm was developed by Glenn D. Bergland in 1968 (see reference 15). To derive this algorithm, we assume that readers are familiar with the Cooly-Tukey radix-2 DIT complex FFT shown in Figure 3-8.

Bergland's algorithm is based on the observation of the symmetry of the FFT to the real input, $X_N(k) = X_N^*(N-k)$. Calculating the second half of the FFT is not necessary. By checking for redundancy in the Cooly-Tukey radix-2 decimation in time complex FFT when input is a real sequence, one may discover that when the twiddle factors equal $W(N/4) = -j$, only a negation and a re-labeling need be performed. This so called re-labeling simply exchanges real and imaginary data indexed by address registers. All odd index outputs in Figure 3-8 are the second half of the transform, which can be obtained from the symmetry. Bergland's algorithm uses those memory locations for storing imaginary values. A direct map from the Cooly-Tukey algorithm to Bergland's algorithm is diagrammatically shown in Figure 6-1. Note that all inputs are real and all intermediate results are stored in N and only N locations. The calculation can be done in-place, however, the indices of each butterfly outputs are not in bit-reversed order as in the Cooly-Tukey algorithm. The following discussion refers to this order as the Bergland order.

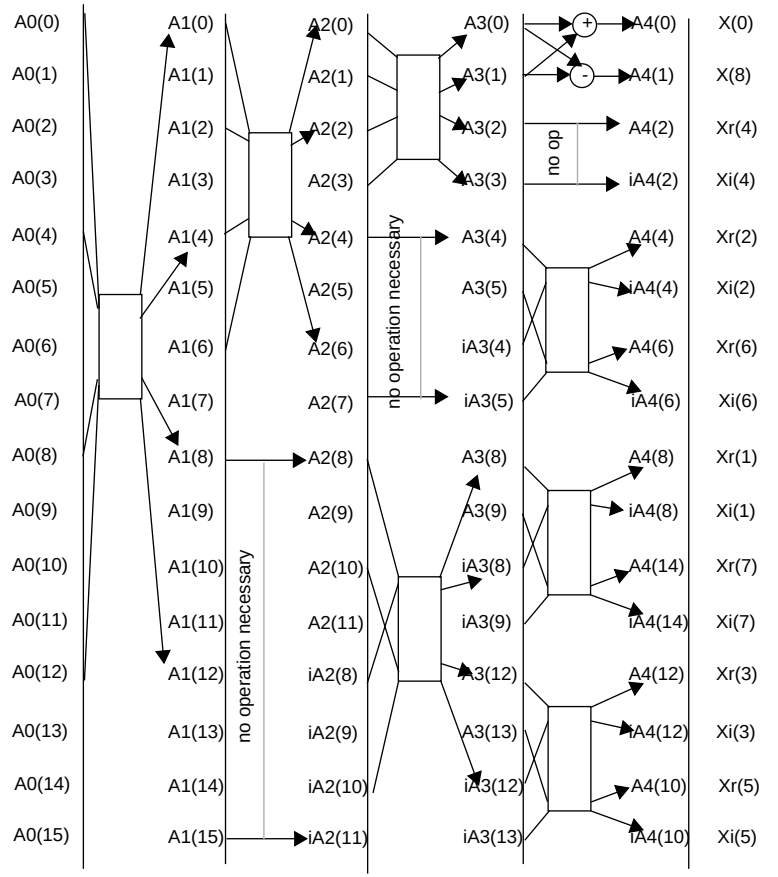

Figure 6-1 Non-redundancy calculation of the Cooly-Tukey radix-2 DIT FFT with real inputs

The twiddle factors appear to be in the Bergland order also, as shown Figure 6-1, if more than 16 points of real FFT are carried out. The next section explains how to convert a normal order of twiddle factors to the Bergland order and how to convert the Bergland ordered outputs to normal order. The only operation performed for multiplying by -j is a re-labeling of half of the current outputs as imaginary inputs for the next stage. Thus, in Figure 4-2 all butterflies, except one with W^0, have the crossed inputs to the butterfly, even though the butterfly in each group is identical. An additional benefit of 'no operation' is the reduction of the number of passes, $\log_2(N)-1$, except for one addition and one subtraction. The final algorithm is shown in Figure 6-2.

The Bergland butterfly differs from the Cooly-Tukey butterfly simply in that the Bergland requires two more conjugate operations, which are done by re-labeling (see Figure 6-3). Essentially, the number of arithmetic operations required by both algorithms is the same. Although re-labeling can be implemented in parallel with other arithmetic operations without consuming instruction cycle time, it does require a data move. This extra traffic may have an impact on the implementation later on. Figure 6-3 depicts the Bergland butterfly. Butterfly (a) is a simplified version of (b) since no complex multiplication is carried out when w=1. Note that the inputs in (b) have been re-labeled to reflect a multiplying -j operation. To calculate the butterfly (a) two additions and two subtractions are needed along with four real multiplications, three real additions, and three real subtractions.

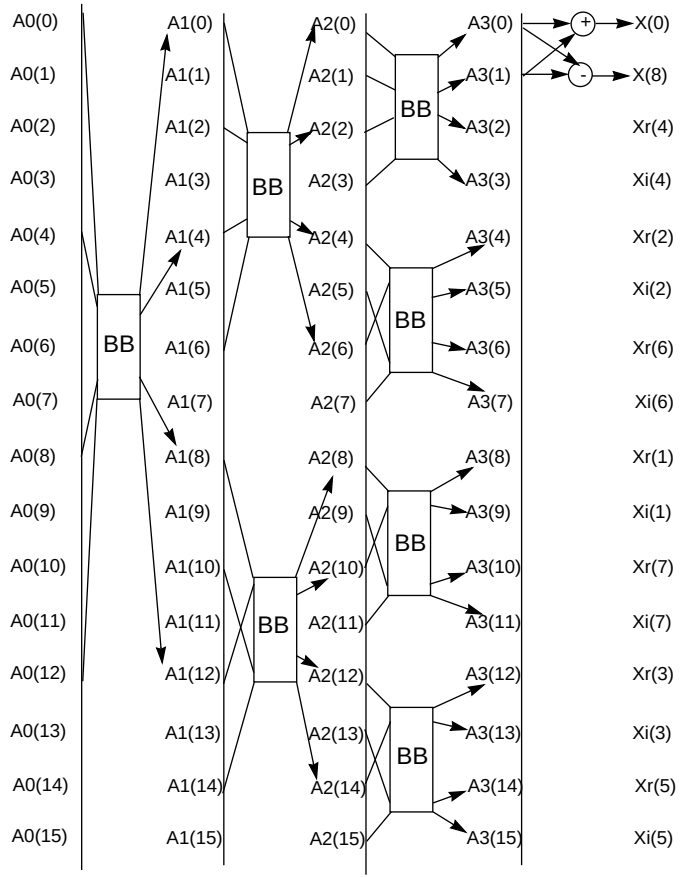

Figure 6-2 Bergland algorithm has only $\log_2(N)$-1 passes and one more addition and subtraction

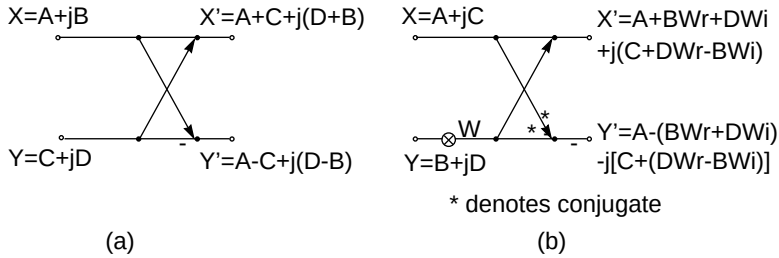

Figure 6-3 (a) Butterfly of Bergland Algorithm with W = 1
(b) Butterfly of Bergland Algorithm with W ≠ 1

6.1.2 Reordering

The output order of the Bergland algorithm is slightly different than the bit-reversed order, and the twiddle factor required in the calculation is also in Bergland order. To get this special order, one may use the following algorithm for doubling the length of each number sequence:

1. Multiply the second entry of the sequence by two, and make this product the second entry of the new sequence
2. Subtract each nonzero entry of the sequence from twice the product formed in step 1 (these differences form the rest of the even entries of the new sequence)
3. Take the odd entries of the new sequence as the numbers of the original sequence

The algorithm in Figure 6-3 can be translated to the following C language code:

```
void
bildberg(bergtabl,buf_size)
short *bergtabl,buf_size;

{
 register int i,j,k;
 i = buf_size / 4;
 k = 4;

 bergtabl[0] = 0;                    /* seed values for start    */
 bergtabl[i] = 2;
 bergtabl[2*i] = 1;
 bergtabl[3*i] = 3;

while(i>1)
 {
 i = i/2;                            /* increments drop by half  */
 k = k*2;                            /* new sequence size doubles*/

 bergtabl[i] = k / 2;
 for (j=i+i;       j<buf_size; j = j+i+i)
 bergtabl[j+i] = k - bergtabl[j];
}
}
```

Figure 6-4 C language code that generates Bergland order tables

Also note that the size of the twiddle factors required in Bergland FFT is N/4, while the size of the output data is N/2. Two tables must be generated before the FFT computation.

6.1.3 Performance Estimation

For $N=2^m$, it has been shown that the pass or stage number in Bergland algorithm is $\log_2(N)-1=m-1$. In each pass there is one (and only one) type (a) butterfly group. The Bergland algorithm

takes four points in and four points out. The number of butterflies in each pass is N/4. Each butterfly uses four multiplications, three additions, and three subtractions, except that the type (a) butterfly uses only two additions or two subtractions. For $N=2^m$, Bergland algorithm may need

$$4 \times N/4 + \sum_{i=2}^{m-1} [4 + BB(2^{i-1} - 1)]N/(2^{i+1}) \qquad \text{Eqn. 6-4}$$

instruction cycles to perform a N-point real FFT, where BB is the number of instructions for the Bergland butterfly. Theoretically, for the DSP96002 and the DSP56001, BB should be 4 and 6, respectively. If the normal order output is desired, then converting Bergland order data to the normal order data must be included in the performance estimation. At least two more instructions have to be added to the last pass for accessing the Bergland order table. The performance of the Bergland algorithm including unscrambling could be:

$$N + \sum_{i=2}^{m-1} [4 + BB(2^{i-1} - 1)](N/2^{i+1}) + (N/2) - 1$$

Eqn. 6-5

Eventually, the real performance of an FFT is determined by the architecture of the DSP on which the FFT runs. As described in **SECTION 4.4**, the actual performance of the FFT is determined by the number of data paths, the number of registers, the instruction type, the cycle time of DO loop, and the memory organization. In other words, a good

or relatively low complexity algorithm may not generate good performance if the microprocessor's architecture does not provide hardware support for that algorithm. Due to the memory structure and instruction type, the number of instructions for a Bergland butterfly, (BB), actually are 5 and 7 on the DSP96002 and the DSP56001,respectively. (See program RFFT96B.ASM and RFFT56B.ASM in **APPENDIX A.**) Due to this compromise in the implementation, the next algorithm may be preferable because of the number of instructions.

6.2 Real-Valued Input FFT Algorithm 2

The second algorithm treats an N real-valued input array as an N/2 complex array, without extra zeros. Then, an N/2 complex FFT is performed. The trick is to separate the transformation of the complex sequence into two complex sequences, then to obtain the transformation of the real-valued input array.

6.2.1 Separating Two Real FFT from One Complex FFT

If a real-valued input array is z(n), its transform Z(k) has an even real part and an odd imaginary part. If z(n) is packed in such a way that all even index data is in x(n) and all odd index data is in y(n), then,

$$Z(k) = DFT[z(n)] = DFT[x(n) + jy(n)]$$
$$= (DFT[x(n)] + jDFT[y(n)])$$
$$= (X_r(k) + jX_i(k) + j[Y_r(k) + jY_i(k)])$$
$$= ([X_r(k) - Y_i(k)] + j[X_i(k) + Y_r(k)])$$

Eqn. 6-6

Eqn. 6-6 shows that the DFT of a complex time sequence z(n) can be represented by the DFTs of two real time sequences x(n) and y(n), because the DFT is a linear transform.

Also the second half of z(n) can be represented by the DFT of x(n) and y(n)

$$Z(N-k) = [X_r(k) + Y_i(k)] - j[X_i(k) - Y_r(k)] \quad \text{Eqn. 6-7}$$

The goal of the derivation is to find out how to construct the DFT of two real time sequences from the DFT of a complex sequence. By combining Eqn. 6-6 and Eqn. 6-7, it shows:

$$DFT[x(n)] = X_r(k) + jX_i(k)$$
$$= \{[Z_r(k) + Z_r(N-k)] + j[Z_i(k) - Z_i(N-k)]\}/2$$

$$DFT[y(n)] = Y_r(k) + jY_i(k)$$
$$= \{[Z_i(N-k) + Z_i(k)] + j[Z_r(N-k) - Z_r(k)]\}/2$$

Eqn. 6-8

where: $k = 0,1,...,N/2$

According to Eqn. 6-8, two DFTs of two real time sequences can be rebuilt from one complex DFT. This split operation, which separates two DFTs from one, paves the way for the calculation of N real input DFTs done by an N/2 complex DFT.

6.2.2 Rebuilding the DFT of a Real Sequence from Two DFTs

From the previous discussion, DFTs of two real sequences can be constructed from one complex DFT. In this section, we investigate how to rebuild the DFT of a real sequence from two DFTS. To understand this point, recall Eqn. 3-1. It can be rewritten as:

$$F(k) = X(k) + W_N^k Y(k) \qquad k = 0, 1, \ldots N-1$$

Eqn. 6-9

where:

$$X(k) = \sum_{r=0}^{N/2-1} x(2r) W_{N/2}^{rk}$$

$$Y(k) = \sum_{r=0}^{N/2-1} x(2r+1) W_{N/2}^{rk}$$

Note that X(k) is the DFT of the even index sequence and Y(k) is the DFT of the odd index sequence. X(k) and Y(k) in Eqn. 6-9 can be determined from Eqn. 6-8. Furthermore, F(k), the DFT of

a real sequence with N points, can be found according to Eqn. 6-9. Combining Eqn. 6-8 and Eqn. 6-9, we obtain the final equation Eqn. 6-10.

$$F(k) = \frac{[Z(k) + Z^*(N/2 - k)]}{2} - j\frac{[Z(k) - Z^*(N/2 - k)]}{2}W_N^{rk} \qquad \text{Eqn. 6-10}$$

where: k = 0,1,...(N/2)-1,
N = Number of real inputs

Notice that:

- Only 0 to N/2-1 points are saved by the algorithm.
- The values F(0) and F(N/2) are real and independent, to obtain entire spectrum, F(N/2) in the imaginary part of F(0).
- The twiddle factors in the DFT and split complex multiplication have different resolutions. In the DFT, the period of W is N/2; in the split complex multiplication, the period of W is N, though the same number of points (N/2) are needed in both cases. This means the algorithm may use more memory space for twiddle factors.

Eqn. 6-10 can be decomposed further to a real multiplication format that can be implemented on DSPs.

$$H1_r = (A_r + B_r)/2$$

$$H1_i = (A_i - B_i)/2$$

$$H2_r = (A_i + B_i)/2$$

$$H2_i = (B_r - A_r)/2 \qquad \text{Eqn. 6-11}$$

$$A'_r = H1_r + (W_r H2_r - W_i H2_i)$$

$$B'_r = H1_r - (W_r H2_r - W_i H2_i)$$

$$A'_i = H1_i + (W_i H2_r - W_r H2_i)$$

$$B'_i = -(H1_i) + (W_i H2_r - W_r H2_i)$$

where: $W_r = \cos((2\pi k)/N)$

$W_i = -\sin((2\pi k)/N)$

and

$A_r = \text{real}Z(k)$ k=0,...(N/4-1)

$B_r = \text{real}Z(N-k)$ k=0,...(N/4-1)

$A_i = \text{imag}Z(k)$ k=0,...(N/4-1)

$B_i = \text{imag}Z(N-k)$ k=0,...(N/4-1)

6.2.3 Performance Estimation

In the following paragraph, we will discuss the computational complexity of Eqn. 6-10 and the implementation constraints on the architecture of Motorola's DSP. For detailed implementation, please refer to the programs RFFT96.ASM and RFFT56.ASM in **APPENDIX A**.

Eight multiplications, five additions, and five subtractions are needed to implement Eqn. 6-10. The minimum requirement for this calculation is eight instructions if one multiplier and the MPY||ADD||SUB is available on the given DSP. Note that there are four special multiplications, and the multiplicands are 1/2 in the calculations of $H1_r$, $H1_i$,$H2_r$, and $H2_i$.

On the DSP56001, the divide-by-2 operation can be automatically implemented by a “scaling down” mode when data moves from the ALU accumulator (A or B) to the X or Y data bus occur. The cost of implementing the division operation, of course, is that one instruction has to be used to turn on the scaling down bit in the Status Register. Apparently, only four multiplications are needed on the DSP56001. But one may find that when the scaling down mode is on, all output data from the accumulator (A or B) to X or Y memory is also divided by 2. Thus, the scaling down mode has to be turned off before data is output to the X or Y memory.

The scaling bit control instructions on the DSP56001 do not allow parallel data moves or any other operations. If the DSP is in the scaling mode, a total of twelve instructions are needed: four MAC instructions, two toggling scaling bit instructions, and six more ADD or SUB instructions. In practice, see program RFFT56.ASM in **APPENDIX A**, where the scaling mode is never turned on because scaling must be done if block floating point is not used. Therefore, the output of the program RFFT56.ASM

is twice as large as true values. Ten instruction cycles is the minimum requirement. In practice, one instruction in the loop for data saving is included.

On the DSP96002, since the FMPY||ADD||SUB instruction is available, eight instructions are enough to perform a computation such as Eqn. 6-10. In **APPENDIX A** more details about implementation such as memory map, program length, twiddle factors, and data size are presented.

The overall performance of the algorithm is determined by the time required to calculate an N/2 complex FFT plus the time for separating manipulations.

$$CFFT(N/2) + S \times N/4$$

Eqn. 6-12

where: S = 11 for the DSP56001
S = 8 for the DSP96002

6.3 Real-Valued Input FFT Algorithm 3

In most practical situations, the data to be analyzed by the FFT is real and is usually obtained from a single analog-to-digital (A/D) converter.This knowledge can be exploited in several ways to increase the speed of the FFT calculation even

further:

1. Since the input data is real, there is no need to multiply, add, or subtract the imaginary parts.
2. Use can be made of symmetries within the FFT:

$$X_N(k) = X_N^*(N-k) \qquad \text{Eqn. 6-13}$$

When x(nT) is real, * denotes complex conjugate.

Clearly, not all of the frequency points need to be calculated, as many of them can be obtained by taking a simple complex conjugate of other, previously computed points. Taking a complex conjugate can be easily achieved by moving the same values to different memory locations, after taking the negative of the value which goes to Y memory (imaginary part). Figure 6-5 shows the procedure for a 16-point, real FFT in greater detail. A real-input FFT routine is available for the DSP56001/2, which executes in 1.01 ms using a 40-MHz clock. This also includes the amount of time necessary to bring in 1024 sampled data points from an external A/D converter. Because of the fast interrupt capability of the DSP56001/2, data sampling creates very little overhead. As a result, the maximum sampling rate at which a 1024-point real FFT can be executed equals:

$$F_{smax} = \frac{1024}{1.01 \times 10^{-3}} = 1.014(MHz)$$

Comparing this with the sampling rate of 3.3 kHz mentioned in **SECTION 3.1 Motivation**, a more than 300-fold improvement is obtained by carefully optimizing the Fourier transform algorithm!

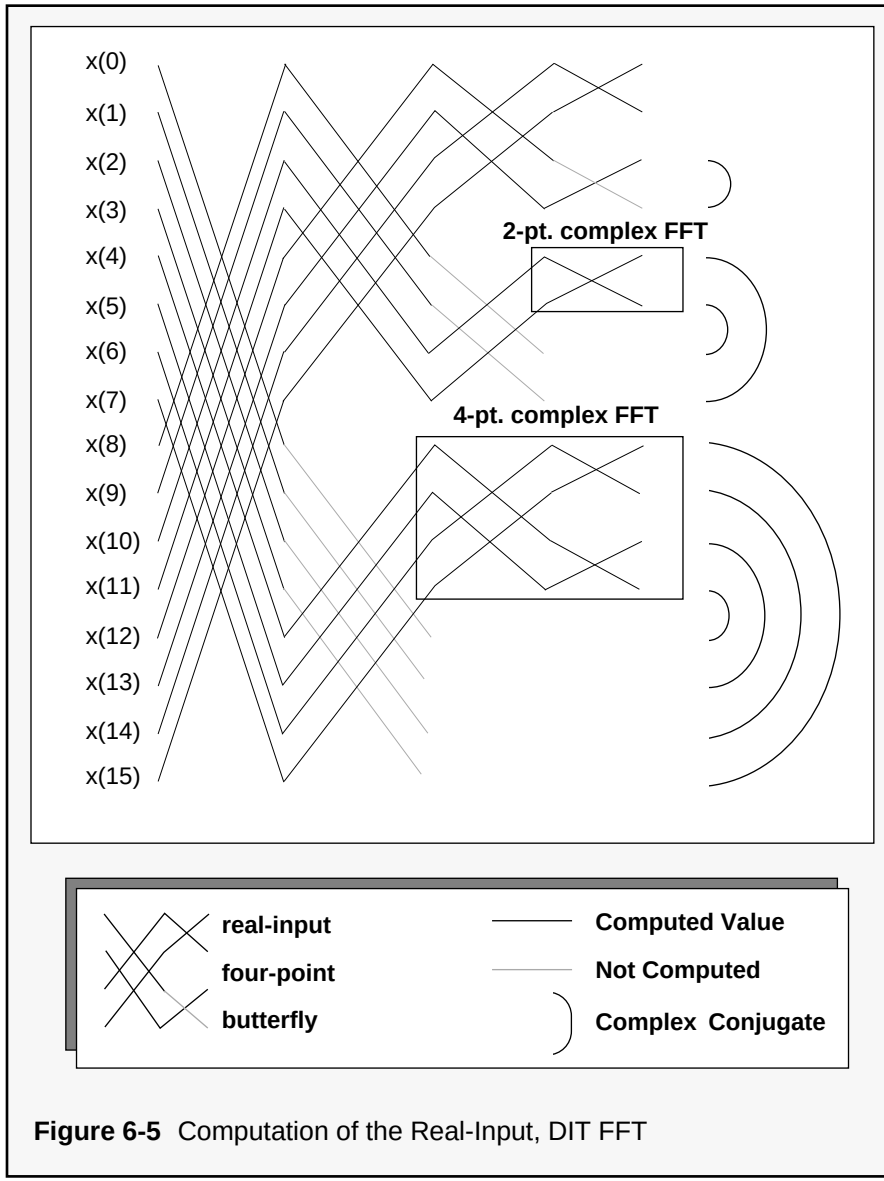

Figure 6-5 Computation of the Real-Input, DIT FFT

6.4 The Goertzel Algorithm

Previous FFT algorithms compute all or half of the frequency points in the range equaling half of the sampling rate. For some applications, such as single frequency detection, only one or several frequency points are of interest. Using FFT to find these frequencies is no longer cost effective in the sense of computational complexity.

The Goertzel algorithm (see reference 3) can be implemented by a second order IIR filter for each DFT coefficient. The transfer function for the IIR filter is:

$$H_k(Z) = \frac{1 - W_N^k Z^{-1}}{1 - 2\cos(2\pi k/N)Z^{-1} + Z^{-2}}$$

Eqn. 6-14

where: $W_N^k = e^{-2\pi kj/N}$

N = the length of input sequence, which depends on the resolution of two consecutive frequencies to be detected

k = the index of DFT coefficient

Also note that only three real coefficients are required in the IIR filter. Naturally, the IIR filter recursively works on input samples and output results, so no input data buffer is needed; and only two memory locations are used for storing internal states of the IIR filter. Figure 6-6 shows an implementation of the Goertzel algorithm by a second order IIR filter. In contrast, an IIR filter calculates

every output corresponding to every input. In the Goertzel algorithm, only one DFT coefficient X(k) is needed, and $X(k) = y_k(N)$. In other words, the complex multiplication is carried out only once in an entire DFT calculation. In frequency detection, only the power of magnitude of the DFT coefficient is needed. This observation may simplify the computation even more.

```
;Goertzel algorithm to calculate energy of DFT coefficient
;
;
;
data     equ      $100
COEF     equ      $123456
LOOP     equ      256

         org p:$40
         move     #data,r0          ;r0 -> input data
         clr a    #0,b              ;I(n-1)=0,I(n-2)=0
         move     #COEF,y0          ;y0=cos(2pik/N)
         do   #LOOP,_END_GOERT
              neg b y:(r0)+,a a,x1  ;x1=I(n-1),b=-I(n-2),a=x[i]/2
              macr y0,x1,a x1,y1    ;a=x[i]/2 + I(n-1)*COEF,y1=I(n-1)
              addl b,a x1,b         ;a=x[i] + 2*I(n-1)*CEOF - I(n-2),b=I(n-1)
_END_GOERT
         mpy -y0,x1,a  a,x0         ;a=-con(2pik/N)I(n),x0=I(n)
         mpy x1,y1,b                ;b=I(n-1)^2
         mac x0,x0,b   a,y0         ;b=I(n)^2+I(n-1)^2
         mpy x1,y0,a                ;a= -con(2pik/N)I(n)I(n-1)
         addl b,a                   ;a= power of magnitude of DFT
```

Figure 6-6 DSP56001 assembly code that calculates energy of DFT coefficients by single parameter

From Figure 6-6, the last output of the IIR filter is:

$$y_k(N) = I(N) - W_N^k I(N-1) \qquad \text{Eqn. 6-15}$$

The power of magnitude of the DFT coefficient is easy to show:

$$\left|y_k(N)\right|^2 = I^2(N) - 2\cos(2\pi k/N)I(N)I(N-1) + I^2(N-1)$$

Eqn. 6-16

Hence, only one real coefficient is required to compute the energy of the signal. Figure 6-6 shows the DSP56001 assembly language code used to detect the energy of a frequency specified by the Goertzel algorithm. The recursive part of the IIR filter is effectively implemented by three instructions. The total instruction cycles for a N-point input sequence is 3N+8. Only one coefficient cos(2pk/N) is stored in the on-chip memory and two more memory locations are used to store internal states I(N) and I(N-1).

6.5 Real-Time Data Acquisition on Motorola DSPs

A very important feature of a DSP is its capability to carry data in and out in a deterministic amount of time without interfering with the CPU core operations. “Real-time FFT” refers to the sampled data from an A/D converter or other devices that is stored in a buffer. Once this buffer is full, the DSP starts the FFT program execution. In the mean time, the DSP grabs the sampled data and puts it into another buffer. Whichever finishes first, (the

FFT program execution or data acquisition), has to wait for the other one to finish its task. Thus, two data buffers, plus synchronization between the program execution and data acquisition is required to implement the real-time FFT. This is also called double buffering. The following sections present the I/O peripherals on the DSP56001/2 and the DSP96002, and show examples of how to set up these peripherals for real-time data acquisition.

6.5.1 Fast Interrupt on DSP56001 for Real-Time FFT Data Acquisition

Figure 6-7 shows a scheme for double buffering. Two memory spaces are exclusively assigned to an FFT program. The FFT program will not start until one of two buffers is full. The loaded buffer will not be loaded with data again unless the FFT has finished its execution on the buffer.

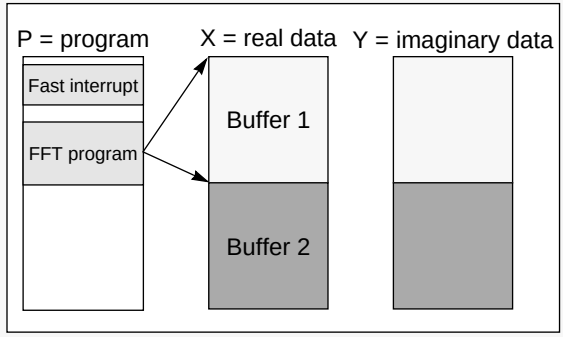

Figure 6-7 Double buffering input data so that data input can work with the FFT program concurrently

The double buffering is implemented by the fast interrupt on the DSP56001/2 (see reference 1). The data received by peripherals such as the SSI or Host Interface (HI) on the DSP56001/2 will be moved into the internal memory by the fast interrupt. The fast interrupt needs only two instruction cycles to move one received data word from a peripheral to a specified memory location without changing the program flow in the CPU.

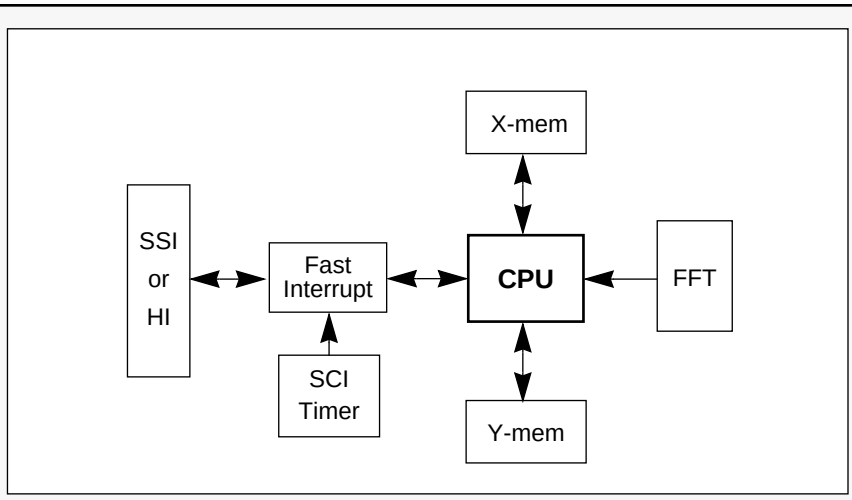

Figure 6-8 Block diagram of the double buffering technique. SSI/HI fast interrupt has higher priority than the MAIN or FFT program. The pointer of buffer is checked by SCI timer interrupt which has highest interrupt priority. The interval of the timer interrupt is set according to data length so that the buffer pointer can be updated accordingly.

The data generation rate is actually much slower than the FFT speed. For example, to generate a set of 1024-point data at 44.1 kHz sampling rate could take 1/44100 x 1024 = 23.2(ms) while a 1024-point real FFT only takes about 1ms at 40 MHz clock on the DSP56001/2. For this reason, the SSI or HI interrupt as shown in Figure 6-8 has been assigned higher priority than the FFT program so that every piece of data received can be sent to internal memory via fast interrupt on the DSP56001/2. The buffer pointer keeps growing by SSI/HI data moves and is being checked by the SCI timer interrupt. Once the buffer is full, the FFT program starts and proceeds to move the buffer pointer to the next buffer so that SSI/HI fast interrupt works with the CPU concurrently.

6.5.2 Real-Time Data Acquisition on DSP96002

The same double buffering technique used on the DSP56001 for real-time data acquisition is also applicable on the DSP96002. Data acquisition on the DSP96002 is truly parallel with CPU instruction execution. Recall the DSP96002 architectural block diagram in Figure 4-4. The double buffering technique guarantees that the two DMA channels directly connected to the internal memory support parallel data access without stretching an instruction cycle if the CPU core and the DMA controller access different internal memory locations. ■