SECTION 4

Complex FFT on the Motorola DSP Family

4.1 Required Hardware Support for FFT Calculation

"In general, doubling the points in butterflies of FFT reduces the number of groups in each pass and the number of passes."

The basic building block of the DIT FFT routine is the butterfly computation shown in Figure 3-6. Consequently, the architecture and instruction set of a DSP device should allow efficient computation of this basic butterfly. Since the butterfly consists of additions and multiplications, a hardware adder/subtracter and multiplier is crucial. The DSP56001/2 and the DSP56156 provide a multiplication and addition instruction, or MAC, which is beneficial to most DSP applications including FFT, with no increase in silicon cost. The DSP96002 supports FFT calculation capability by adding subtraction to the MAC function, which provides the multiplication, addition and subtraction instruction, FMPY||ADD||SUB.

Since the butterfly calculation requires complex data, the architecture must easily support complex arithmetic. The input and output data to the butterflies are moved between the processor's arithmetic unit and memory. Consequently, efficient moves are needed.

DSP56001/2 and DSP96002 hardware feature two data memory modules; X and Y. The real component and imaginary component of a complex number can be stored in the X and Y memory modules respectively. Also, the DSP56001/2 and the DSP96002 can perform dual reads and dual writes in one instruction cycle. In contrast, the DSP56156 has only one data memory module, X, where both real and imaginary components of the complex data are stored. To support complex number fetch, the DSP56156 provides dual memory read, where in one instruction, it reads the X memory twice if the specified address registers are used.

The overall FFT algorithm is an array of many such butterflies, and the size of the array depends upon the number of points (N) in the FFT. In order to write general FFT routines (for any N of the power of 2), efficient implementation of the repetitive execution of the basic butterfly element is important. Although FFTs may be calculated on general-purpose microprocessors, typically, a great deal of software overhead is involved. A hardware solution, using hardware designed to efficiently implement the calculation of FFTs, would be generally preferred in a real-time system.The DSP56001/2, DSP96002, and DSP56156 feature a zero-overhead DO loop instruction. After the loop is set up (three instruction cycle time), each iteration takes no additional cost in overhead.

In real-life applications, time as well as frequency data is used in normal order, even though the diagram of Figure 3-7 delivers the frequency data in bit-reversed order. Thus, an efficient method for bit-reversed addressing is needed while avoiding time-consuming

software solutions that modify the addressing order. The DSP56001/2, DSP96002, and DSP56156 all feature a bit-reversed addressing mode.

Some FFT algorithms, (for example, radix-4 FFT) require several registers to hold immediate results. The number of registers available on the DSPs is critical for computation intensive applications since storing and restoring intermediate results to and from memory will take more processing time than if the results are available in on-chip registers.

The input data (time samples) of the FFT is usually obtained from an external source such as an A/D converter. This data collection must occur in parallel with the FFT computation to make real-time performance possible. Consequently, a DSP device must provide easy interface with a variety of A/D converters, and must support low-overhead interrupt schemes which can load data from an external device with minimal impact on the FFT computation. The DSP56001/2, DSP96002, and DSP56156 all feature a variety of peripherals on chip. More details about real-time data acquisition are discussed in ***SECTION 7***.

The key points to implementing efficient FFT calculation using programmable DSPs are summarized below.

FFT calculation requires:

1. MAC or, ideally, FMPY||ADD||SUB instruction
2. Dual memory read and write in one instruction cycle
3. Zero-overhead loop instruction

4. Bit-reversed addressing mode
5. Sufficient number of registers
6. Fast I/O to provide real time data (in real-time applications)

4.2 Radix-2 DIT and DIF Butterflies

Theoretically, radix-2 decimation in time (DIT) butterflies and decimation in frequency (DIF) butterflies have the same computational complexity: three additions, three subtractions, and four multiplications. Since most DSPs have only one hardware multiplier, the minimum cycle time for multiplication for one DIT or DIF butterfly is four instruction cycles. However, on the DSP56001, a MAC instruction can implement one multiplication and one addition in parallel in a single instruction cycle. Four of the six additions or subtractions in a DIT butterfly can be executed in parallel with four multiplications, and two more additions are required to finish the DIT butterfly calculation. Due to data dependence, a DIF butterfly can implement only two additions in parallel with two multiplications. Thus, one DIF butterfly calculation requires four multiplications plus four additions (see Figure 4-3).

The DSP96002 features a special instruction, FMAY||ADD||SUB, which can implement either a DIT or a DIF butterfly in four instruction cycles. Although the DSP56156 has a MAC instruction, the lack of a dual memory write operation plus constraints on ad-

dress pointer updates in dual memory read operations, causes the DIT butterfly and the DIF butterfly to both take eight instruction cycles.

In short, the Motorola DSP architecture implements the more efficient DIT butterfly, since it generates shorter cycle time than the DIF. The following discussions assume a radix-2 DIT, extending to radix-4 DIT in later sections.

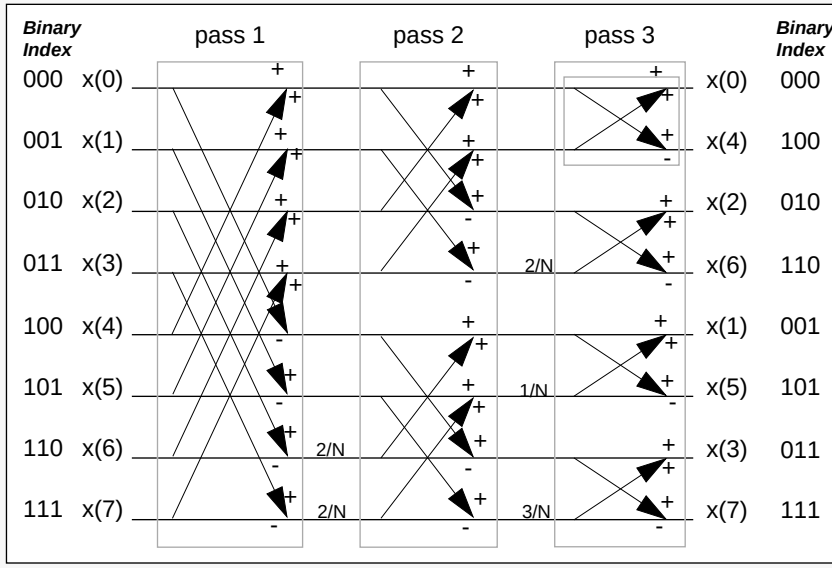


Figure 4-1 Grouping of Butterflies in the FFT Calculation

4.3 Complexity of a Radix-2 DIT FFT

The number of instructions required in a radix-2 DIT FFT is determined by the number of instructions in the butterfly core and the structural overhead of the DSP. If only arithmetic operations are counted in term of the multiplications and additions, a triple-nested implementation of the FFT (see next sections) requires the following number of instruction cycles for $N = 2^m$:

$$m \times N/2 \times BFLY \qquad \text{Eqn. 4-1}$$

where BFLY is number of instructions for calculating a complex input butterfly. For the DSP56001/2, the DSP96002 and the DSP56156, BFLY is 6, 4, and 8 respectively. On the DSP96002, for example, a 1024-point complex FFT needs 10 x 512 x 4 = 20,480 instruction cycles.

4.4 Implementation on Motorola's DSP56001

4.4.1 DSP56001 Architecture

The DSP56001 (see Reference 4) was the first member of the Motorola Digital Signal Processor line. It features 16.5 million instructions per second (MIPS) with a 33 MHz clock.

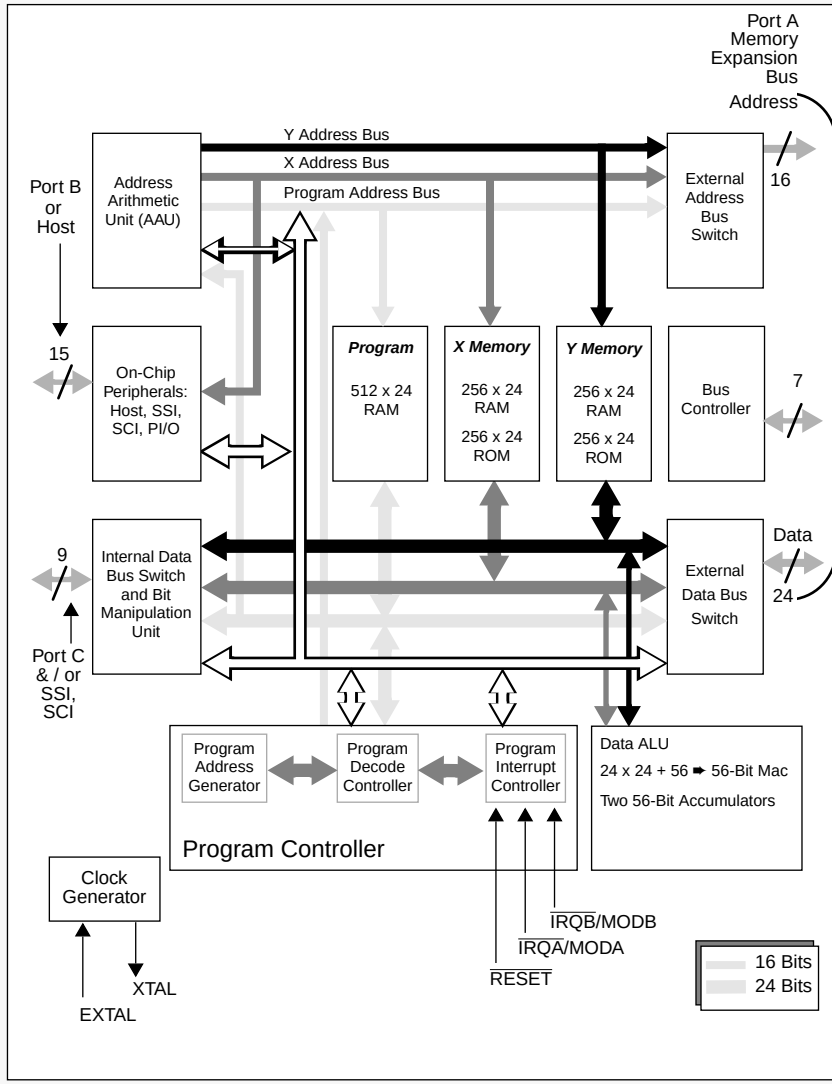
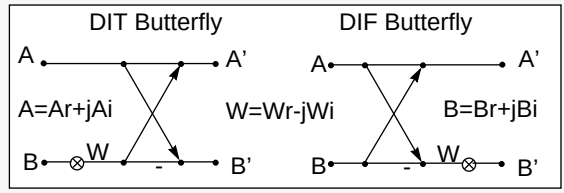


Figure 4-2 DSP56001 Architecture Block Diagram

The data paths are 24 bits wide, thereby providing 144 dB of dynamic range. More importantly, intermediate results are held by a 56-bit accumulator which gives more accuracy in noise sensitive applications. The data ALU, address arithmetic units, and program controller operate in parallel so that an instruction prefetch, a 24x24-bit multiplication, a 56-bit addition, two data moves, and two address pointer updates using one of three types of arithmetic (linear, modulo, or bit-reversed) can be executed in one instruction. Three on-chip peripherals (Serial Communication Interface, Synchronous Serial Interface and Host interface), a clock generator and seven buses (three address, four data) make the overall system functionally complete and powerful. The architecture of DSP56001 is shown in Figure 4-2.



T1=Ar+BrWr	(MAC)	Ar'=Ar+Br	(ADD)
Ar'=T1+BiWi	(MAC)	T1=Ar-Br	(SUB)
Br'=2Ar-Ar'	(SUBL)	Ai'=Ai+Bi	(ADD)
T2=Ai-BrWi	(MAC)	T2=Ai-Bi	(SUB)
Ai'=T2+BiWr	(MAC)	T3=T1Wr	(MPY)
Bi"=2Ai-Ai"	(SUBL)	Br'=T3+T2Wi	(MAC)
		T4=T2Wr	(MPY)

Figure 4-3 A radix-2 DIT butterfly needing less instruction cycles than a radix-2 DIF butterfly

4.4.2 DIT Butterfly Kernel on DSP56001

The parallel architecture and the instruction set of Motorola's DSP56001/2 lend themselves particularly well to the radix-2 DIT FFT computation.The DIT butterfly equations are programmed on Motorola's DSP56001/2 as given below:

$$A'_r = A_r + B_r W_r + B_i W_i \qquad \text{Eqn. 4-2}$$
$$A'_i = A_i + B_i W_r - B_r W_i$$
$$B'_r = 2A_r - A'_r$$
$$B'_i = 2A_i - A'_i$$

where: ***i*** represents an imaginary component
r represents a real component
‘ symbolizes output items

The basic butterfly “core” is implemented by assembly language in Figure 4-4. Note that the previous DSP56001/2 equations are written in this particular form such that the instruction to shift left and subtract accumulators (SUBL) can be used. This SUBL instruction allows efficient implementation of the DIT butterfly in a two-accumulator ALU.

```
;r0➧ A
;r1➧ B
;r4➧ C
;r5➧ D

mac     x1,y0,b     y:(r1)+,y1              ;Ai - BrWi ➧ b,Bi ➧ y1
macr    -x0,y1,b    a,x:(r5)+   y:(r0),a    ;Ai - BrWi + BiWr➧ b,Ai ➧ a
subl    b,a         x: (r0),b   b,y:(r4)    ;2Ai - b ➧ a,Ar ➧ b
mac     -x1,x0,b    x: (r0)+,a  a,y:(r5)    ;Ar + BrWr ➧ b,Ar➧a
macr    -y1,y0,b    x: (r1),x1              ;Ar + BrWr + BiWi➧ b,Br➧x1
subl    b,a         b,x:(r4)+   y:(r0),b    ;2Ar - b ➧ a,Ai ➧ b
```

Figure 4-4 The radix-2, DIT butterfly kernel on the DSP56001/2

The kernel shown in Figure 4-4 executes in six instruction cycles, or a total of 12 clock cycles. This is made possible because of the parallel architecture of the DSP56001/2, which allows up to two data ALU operations (multiply/accumulate) in parallel with two data moves to/from memory and two pointer updates in a single instruction cycle. The dual data spaces X and Y with the appropriate X and Y buses are ideally suited for complex arithmetic; the real components are stored in X memory and the imaginary components are stored in Y memory.

The simplest way of combining all of the butterflies into a complete program is shown in Figure 4-1. The FFT diagram is first divided into FFT passes. On each pass, the data is fetched from memory, the butterfly calculations are done, and the results are moved back out to memory. It is easily shown that there are log2N passes. Within each pass, the butterflies cluster in groups. From one pass to the next, the number of groups doubles, while the number of butterflies per group is divided by two. Note that the twiddle factors are the same for all butterflies within each group, and that the order of the twiddle factors from one group to the next is bit-reversed. This is easily implemented on the DSP56001/2 by setting the appropriate modifier register (m6) equal to zero and the offset register (n6) equal to N/4 (= coefficient table size/2), such that the twiddle factors are addressed in bit-reversed manner.

This gives rise to the simple, triple-nested DO loop program shown in Figure 4-5. The outer DO loop steps through passes, the middle loop goes through all of the groups within a pass, and the inner loop cycles through all of the butterflies inside a group. The

DSP56001/2 is particularly well suited for looped program execution because it has hardware DO-loop capability. Once a loop is entered through the DO instruction, this loop is executed without any time penalty. The resulting program takes 40 words in program memory. This is the most compact implementation of the radix-2 DIT FFT. A 1024-point complex FFT using this code executes in 4.72 ms when using a 27-MHz clock.

```
;This program originally available on the Motorola DSP bulletin board.
;It is provided under a DISCLAIMER OF WARRANTY available from
;Motorola DSP Operation, 6501 Wm. Cannon Drive W., Austin, Tx., 78735.
;
;Radix 2, In-Place, Decimation-In-Time FFT (smallest code size).
;
;Last Update 30 Sept. 86 Version 1.1
;
fftr2a      macro      points, data, coef
fftr2a      ident      1,1
;
;Radix 2 Decimation in Time In-Place Fast Fourier Transform Routine
;   Complex input and output data
;         Real data in X memory
;         Imaginary data in Y memory
;   Normally ordered input data
;   Bit reversed output data
;         Coefficient lookup table
;         -Cosine values in X memory
;         -Sine values in Y memory
;
;Macro Call - fftr2a points,data,coef
;
;   points              number of points (2-32768, power of 2)
;   data                start of data buffer
;   coef                start of sine/cosine table
;
;Alters Data ALU Registers
;   x1    x0    y1    y0
;   a2    a1    a0    a
;   b2    b1    b0    b
;
;Alters Address Registers
;   r0    n0    m0
;   r1    n1    m1
;         n2
;
;   r4    n4    m4
;   r5    n6    m5
;   r6    n6    m6
;
```

Figure 4-5 A Simple, Triple-Nested DO Loop Radix-2 DIT FFT on DSP56001/2 (sheet 1 of 2)

```
;Alters Program Control Registers
;  pc   sr
;Uses 6 locations or System Stack
;Latest Revision     September 30, 1986
;r0 points to A
;r1 points to B
;r4 points to C
;r5 points to D
;r6 points to twiddle factor                                                    move
# points/2,n0;initialize butterflies per group
        move        # 1,n2           ;initialize groups per pass
        move        # points/4,n6    ;initialize C pointer offset
        move        #-1,mo           ;initialize A and B address modifiers
        move        m0,m1            ;for linear addressing
        move        m0,m4
        move        m0,m5
        move        #0,m6            ;initialize C address modifier for
                                     ;reverse carry (bit-reversed) addressing
;
;Perform all FFT passes with triple nested DO loop
;
   d0       #(αcvi(αlog(points)/(αlog(2)+0.5)_end_pass
   move     #data,r0                          ;initialize A input pointer     move
r0,r4   ;initialize A output pointer lua
(r0)+n0,r1[;initialize B input pointer move
#coef,r6;initialize C input pointer
   lua      (r1)-,r5                          ;initialize B output pointer
   move     n0,n1                             ;initialize pointer offsets     move
n0,n4
   move     n0,n5
   d0       n2,_end_grp
   move     x:(r1),x1y:(r6),y0                ;lookup -sine and
                                              ;-cosine values
   move     x:(r5),a    y:(r0),b              ;preload data
   move     x:(r6)+n6,x0                      ;update C pointer

   do       n0,_end_bfy
   mac      x1,y0,b     y:(r1)+,y1            ;Radix 2 DIT
                                              ;butterfly kernel
   macr     -x0,y1,b    a,x:(r5)+   y:(r0),a
   subl     b,a         x:(r0),b    b,y:(r4)
   mac      -x1,x0,b    x:(r0)+,a   a,y:(r5)
   macr     -y1,y0,b    tx:(r1),x1
   subl     b,a         b,x:(r4)+   y:(r0),b
_end_bfy
   move     a,x:(r5)+n5 y:(r1)+n1,y1          ;update A and B pointers
   move     x:(r0)+0,x1 y:(r4)+4,y1
_end_grp
   move     n0,b1
   lsr      b           n2,a1                 ;divide butterflies per group by two
   lsl      a           b1,n0                 ;multiply groups per pass by two
   move     a1,n2
 _end_pass
   endm
```

Figure 4-5 A Simple, Triple-Nested DO Loop Radix-2 DIT FFT on DSP56001/2 (sheet 2 of 2)

4.5 Implementation on Motorola's DSP96002

4.5.1 DSP96002 Architecture

DSP96002 is a 32-bit floating-point digital signal processor with 20 million instructions execution per second using a 40 MHz clock. The data ALU provides full conformance with the IEEE 754-1985 Standard for Single Precision Binary Floating-Point Arithmetic. Single Extended precision with a 32-bit mantissa and 11-bit exponent is also implemented. The data ALU, AGU, and program controller operate in parallel within the CPU so that an instruction pre-fetch, up to three floating point operations, two data moves, and four address pointer updates using one of three types of arithmetic (linear, modulo, and reverse carry) can all be executed in one instruction cycle.

Also, an on-chip dual channel DMA controller generates two addresses, using one of the three types of address update arithmetic so that a memory-to-memory or memory-to-peripheral transfer can occur in parallel with the CPU operation during each instruction cycle. Host interface circuitry on each port provides a flexible slave interface to external processors and/or DMA controllers for easy design of a multi-master system. Designed primarily for image processing, real-time data acquisition, sonar signal processing, radar signal processing, medical image analysis, and video compression, the DSP96002 has the widest data bandwidth of any DSP currently on the market. A special FMAY||ADD||SUB instruction makes FFT calculations extremely fast on the DSP96002.

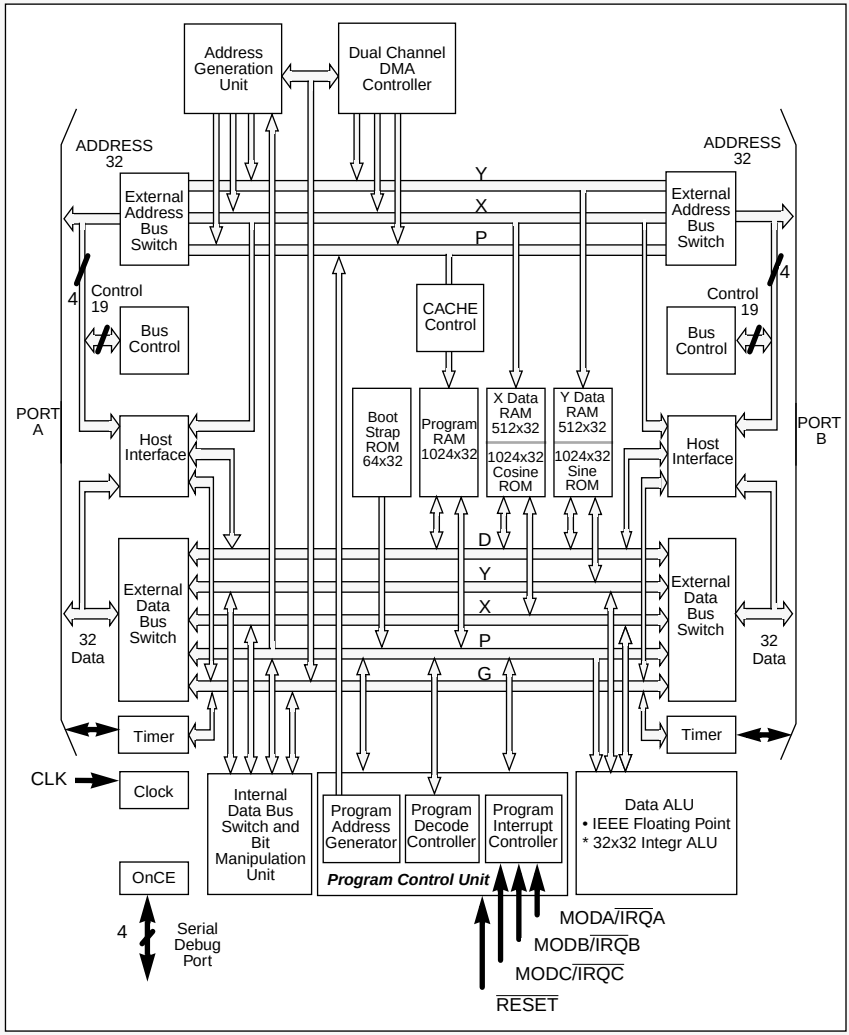


Figure 4-6 DSP96002 Architectural Block Diagram. Two symmetric bus expansion ports with two channel DMA controller that blow away the speed limit on external memory access and data I/O.

4.5.2 DIT Butterfly Kernel on DSP96002

The butterfly equations implemented in the radix-2, DIT FFT on DSP96002 are the following:

$$A'_r = A_r + B_r W_r + B_i W_i$$
$$A'_i = A_i + B_i W_r - B_r W_i \qquad \text{Eqn. 4-3}$$
$$B'_r = A_r - (B_r W_r + B_i W_i)$$
$$B'_i = A_i - (B_i W_r - B_r W_i)$$

where: ***i*** represents an imaginary component

r represents a real component

‘ symbolizes output items

The implementation of this basic butterfly in DSP96002 assembly language code is shown in Figure 4-7. The kernel in Eqn. 4-3 executes in four instruction cycles, or eight clock cycles. Since four real multiplications are needed, and only one real multiplier is available, this is the most efficient implementation possible. In addition to the features available on the DSP56001/2, this efficient execution is obtained by the FADDSUB instruction which delivers the sum and the difference of two operands, in parallel with a multiplication and two data moves. With this feature, a total of three floating-point operations can be executed in one instruction cycle, resulting in a peak performance of 60 million floating-point operations per second (MFLOPS) with a 40-MHz clock.

The triple-nested DO loop routine, which computes the radix-2, DIT FFT on the DSP96002 takes only 30 words in program memory. A 1024-point complex FFT is executed in only 2.31 ms, assuming a 27-MHz clock.

```
;r0 ➡ A
;r1 ➡ B
;r4 ➡ C
;r5 ➡ D

fmpy d8,d6,d   fadd.s   d3,d0   x:(r0),d4.s  d2.s,y:(r5)+  ;Br*sin ➡ d2
                                                          ;Bj*sin + Br*cos ➡ d0
                                                          ;Ar ➡ d4,Dj ➡ mem.

fmpy d8,d7,d3  faddsub.sd4,d0   x:(r1)+,d6.s d5.s,y:(r4)+  ;Bj*sin ➡ d3
                                                          ;Ar + Br1 ➡ d0
                                                          ;Ar - Br1➡ d4
                                                          ;Br ➡d6
                                                          ;Cj ➡ mem.

fmpy d9,d6,d0  fsub.sd1,d2      d0.s,x:(r4)  y:(r0) + d5.s ;Br*cos ➡ d0
                                                          ;Br*sin - Bj*cos ➡ d2
                                                          ;Cr ➡ mem.
                                                          ;Aj ➡ d5

fmpy d9,d7,d1  faddsub.sd5,d2   d4.s,x:(r5)  y:(r1),d7.s   ;Bj*cos ➡ d1
                                                          ;Aj + Bj1 ➡ d2
                                                          ;Aj - Bj1 ➡ d5
                                                          ;Dr ➡mem.
                                                          ;Bj ➡ d7
```

Figure 4-7 The Radix-2, DIT FFT Butterfly Kernel on the DSP96002

4.6 Implementation on Motorola's DSP56156

4.6.1 DSP56156 Architecture

The DSP56156 is the most recent addition to the Motorola DSP line. This 16-bit fixed-point number DSP is designed primarily for speech coding and telecommunication. The on-chip sigma-delta codec functions as a bridge between the analog and digital world. The on-chip phase-locked-loop (PLL) reduces clock noise to a minimum. Operating at 60 MHz, the DSP56156 can execute 30 million instructions per second with two kilowords (2k) on-chip data RAM (which is four times larger than DSP56001's) and four address registers. Since the DSP56156 is designed for the digital cellular phone, its limited instruction operation codes must focus on telecommunication capability, and some of its advanced addressing modes and instructions that accelerates FFT calculation must be compromised due to the smaller instruction words.

Although only one memory module can be accessed in a single instruction cycle, the DSP56156 does support dual memory reads. However, it does not support dual memory writes in a single instruction cycle. Four address registers and a single write per instruction may slow down FFT performance on DSP56156, but having 2k on-chip data memory may compensate for a portion of the performance loss, i.e. dual on-chip memory reads may save time equivalent to four instruction cycles if the number of data points is between 256 and 1024 points.

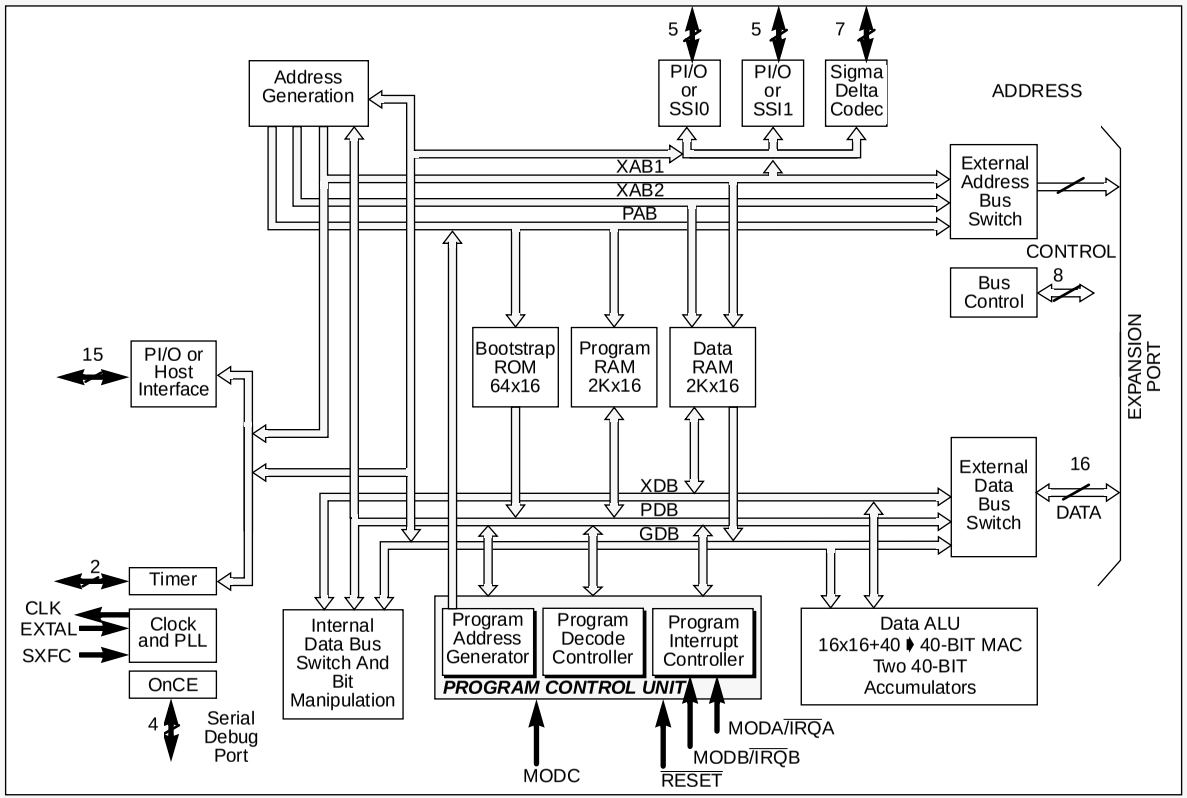


Figure 4-8 DSP56156 architectural block diagram. Only four address registers are available. On the single data memory space, only a dual-read per one instruction is allowed.

4.6.2 DIT Butterfly Kernel on DSP56156

The butterfly equation for the DSP56156 is the same as the DIT butterfly equation for the DSP56001/2 as shown in Eqn. 4-2. However, two more instructions are required in the DSP56156 butterfly than the DSP56001/2 because of its lack of a dual-write operation and its constraints on the address register mode. Figure 4-9 shows the DSP56156 assembly language code of the butterfly core.

```
mpy   x0,y0,b   a,x:(r2)+                ;b=WrBr,save prev. Bi',r2 -> Br
macr  x1,y1,b   x:(r0)+n0,a              ;b=WrBr+WiBi,a=Ar
add   a,b                                ;b=Ar+WrBr+WiBi=Ar'
subl  b,a       b,x:(r0)+                ;a=2Ar-Ar'=Br', save Ar', r0 pt to Ai
mpy   -y1,x0,b  a,x:(r2)+                ;b=-WiBr,       save Br', r2 pt to Bi
macr  y0,x1,b   x:(r0)+n0,a  x:(r3)+,x0  ;b=-WiBr+WrBi,a=Ai, x0=next Br
add   a,b                    x:(r3)+,x1  ;b=Ai-WiBr+WrBi=Ai', x1=next Bi
subl  b,a          b,x:(r0)+             ;a=2Ai-Ai'=Bi', save Ai', r0-> next Ar
```

Figure 4-9 The butterfly core of the DSP56156. Notice that a single write operation paralleling with an instruction always occupies a whole data move field.

4.7 Scaling for Fixed-Point Processors

(DSP56001/2 and DSP56156)

Whenever mathematical algorithms are implemented in digital hardware, note that results are obtained with finite precision. The precision is generally limited by the number of bits used in the number representation, and depends on how the arithmetic

limits its results to those bits. The user must use care to prevent overflows in the FFT outputs of fixed-point DSPs. Scaling via shifting or dividing can keep input data or intermediate results within the correct range, while maintaining maximum precision on the outputs.

4.7.1 Scaling at the Input – Guard Bits

Since data length grows with each pass, overflow can occur at any pass if there is no scaling in the input of a fixed point number DSP. The magnitude of the output by the DIT butterfly defined in Eqn. 4-2 will grow an average of one bit on the output in each pass. This is based on the observation that output A' (a complex output) can be rewritten as A' = A+ B x W where A', A, B, and W are complex numbers. Since $W = e^{-j\theta}$, it has a unit magnitude.

The complex operation B x W simply rotates B according to θ and causes no magnitude growth. Complex addition is the only chance in a single butterfly calculation to make the output magnitude grow larger than a value of one. One addition can cause growth of one bit. Therefore, for $N = 2^m$ points of the FFT, m passes are required, i.e., m times a potential worst case magnitude doubling. However, the twiddle factor will reach its maximum magnitude when $\theta = \pi/4$. For this case, the maximum magnitude growth is 2.4 bits on real and imaginary components. Fortunately, only two groups of butterflies in each pass will use the maximum twiddle factors. No butterflies use the maximum twiddle factors twice

within an entire FFT calculation. This mutually exclusive characteristic is the base upon which block floating point arithmetic is designed.

To prevent overflows in the FFT calculations, the input data should keep m zeros in the significant part so that growth bits will not get lost during the overflow. The m zeros are called “guard bits”. To obtain sufficient guard bits, divide the input data words by N. For example, if the DSP56001 is implementing a 1024-point complex FFT, 10 guard bits are inserted into the most significant bits of the 24-bit data word, resulting in 14 bits of actual information. But on the 16-bit DSP56156, only 6 bits contain actual information after 10 guard bits are inserted. This may make the signal-to-noise ratio unacceptably low. This method of scaling the input data is simple and effective on a smaller FFT or on a large data word processor like the DSP56001. For a larger FFT or a small data word processor, an alternative method discussed in the next subsection may result in improved signal-to-noise ratio with some trade-offs.

4.7.2 Scaling During the Passes – Auto-Scaling and Block Floating-Point

Scaling in the input truncates valuable information contained in data words by shifting input data right by m-bits. 6.02 x m dB have already been lost before the start of the FFT calculations. As indicated in the last subsection, an average of one bit word growth occurs in each pass. Another way to prevent over-

flow in the FFT calculation is to scale down the output of the butterfly by two at each pass, regardless of whether or not an overflow occurs. Since the scaling down at the output is automatically carried out to the next pass, the amount of scaling down is known before hand. To obtain the true FFT output, simply multiply each output by N. This method is simple and has better signal-to-noise ratio than the scaling in the input method. But some passes may not have bit growth or overflows, so excessive scaling may occur, and automatic scaling may cause some information to be lost.

A more aggressive method treats one pass as one block of data, and assigns an exponent for each block. If bit growth occurs, the method scales down the output by one bit and increases the exponent by one. At the end of the FFT, the same number of scaling up operations must be carried out. In the DSP56156/DSP56002, the scaling bit (bit 7 in the status register) eases implementation of this method. The scaling bit is referred to as a “sticky” bit because once set, it retains its status until the next read of the status register. Five more instructions are added to the end of each pass to check the scaling bit in the DSP56002 and DSP56156, and to update the exponent of the complex FFT. (See program FFTBF.asm on the Motorola DSP bulletin board; Dr. BuB.) Among the methods discussed here, the sticky bit method gives the best signal-to-noise ratio.

4.8 Twiddle Factors and On-Chip ROM

4.8.1 Twiddle Factors for Decimation-in-Time

Twiddle factors, $W_N^k = e^{-j2\pi k/N}$, are coefficients used in FFT calculations. For normal order input radix-2 decimation-in-time FFT, the twiddle factors are always fetched in bit-reversed order, i.e.

$$W_N^0, W_N^{(N/2)-1}, W_N^{N/4}, W_N^{N/8}, W_N^{(3N)/8}, \ldots, W_N^{(N/4)-1}$$

Note that for an N point radix-2 FFT, two input data words share one twiddle factor, and the bit-reversed order of the twiddle factor is based on N/2 points.

4.8.2 Sine Table on the DSP56001/2

When the data-ROM-enable (DE) bit in the OMR register of the DSP56001/2 is set, the Y memory from \$100 to \$1FF contains a 256-point full cycle sine-wave, and each data entry has 24-bit accuracy. As mentioned in the last subsection, for an N point FFT, N/2 complex coefficient twiddle factors are required, and these N/2 twiddle factors are a half cycle of the sine and cosine waveforms. Since only a 256-point full cycle sine-wave is stored in the DSP56001/2 data ROM, the maximum FFT length utilizing only internal twiddle factors is one full cycle

of the sine table, 256 points. However, a FFT larger than 256 points can still be implemented utilizing the on-chip sine table by calling this internal ROM during the first several passes and the first several groups in the last pass. Because DIT and normal input order FFT require bit-reversed sine and cosine tables, the DSP must be in the bit-reversed addressing mode when the on-chip sine table is invoked. A common set up for addressing this table is:

```
r6 = $100
n6 = $40
m6 = 0
```

To address the cosine table in the FFT calculation, the following relation between sine and cosine is utilized:

$$\cos(x) = \sin(x + \pi/2) \qquad \text{Eqn. 4-4}$$

Another address pointer, for example, r2 is used to point to the correct location.

```
r2 = $140
n2 = $40
m0 = 0
```

This set-up can be applied for all FFTs up to 256 points with length equaling a power of two, 2^N.

4.8.3 Sine and Cosine Tables on the DSP96002

The on-chip ROM of the DSP96002 features sine and cosine tables. When the DE bit is set to 1, X and Y memory from $400 to $7FF contain 512-point cosine and sine tables respectively. Therefore, the

maximum data length of the FFT without utilizing external twiddle factors is 512 points. The addressing set-up is similar to that of the DSP56001:

```
r6 = $400
n6 = $100
m6 = 0
```

Only one set of address registers is required on the DSP96002 to access both sine and cosine values.

4.9 Bit-Reversed Addressing

All Motorola DSPs feature a bit-reversed or inverse-carry addressing mode to accelerate FFT calculations. When bit-reversed addressing is enabled, an additional temporary data buffer is required to hold normal order outputs since bit-reversing on the fly is not an in-place method of FFT calculation. In some situations, the memory space used is more critical than the time used. To reduce the requirement for space in the second buffer, an in-place bit-reversed method is preferred. However, there is a time penalty for space-saving since the in-place bit-reversal must be carried out after the FFT is done. Program BITREVTWD56.asm on the Motorola DSP bulletin board (Dr. BuB) presents an example of in-place bit-reverse for DSP56001/2. The algorithm that performs conversion from bit-reversed order to normal order addressing is presented in Figure 4-10.

```
normal_order=output_pointer;
bitrev_order=data_buffer;
for (i=0;i<N;i++){
        normal_order+;
        bitrev_order+=N/2;
        \* suppose bit reverse address available *\
        if (normal_order< bitrev_order)
        data[normal_order]=data[bitrev_order]
}
```

Figure 4-10 In-place bit-reversed to normal order conversion

4.10 Implementation of a Radix-4 DIT FFT on DSP96002

In general, doubling the points in butterflies of FFT reduces the number of groups in each pass and the number of passes. A radix-4 butterfly accepts four complex inputs, thus, the number of butterflies in a pass is N/4, and the number of passes is $\log_4(N)$. However, the number of instructions required in the radix-4 butterfly is three times that of the radix-2 butterfly. If the number of the instructions used in a radix-4 butterfly is four or more times than that of the radix-2's on a processor, there is really no advantage to adapting the radix-4 FFT on such a processor. Because the outputs or inputs of a radix-4 FFT might be digit-reversed order which is not being supported by any DSPs in the market. A software routine has to be used for converting digit-reversed order data to the normal one.

4.10.1 Radix-4 DIT Butterfly Core

The butterfly equations for a radix-4 DIT FFT can be derived directly from two stages of radix-2 DIT butterflies, which are plotted in Figure 4-11. There are four butterflies with four twiddle factors involved in the calculation. In the first pass, pass x, two butterflies are in the same group (the twiddle factors for a group are identical). In the second pass, pass x+1, two adjacent butterflies share one twiddle factor but differ by -j. (See ***SECTION 5.1 Optimization***).

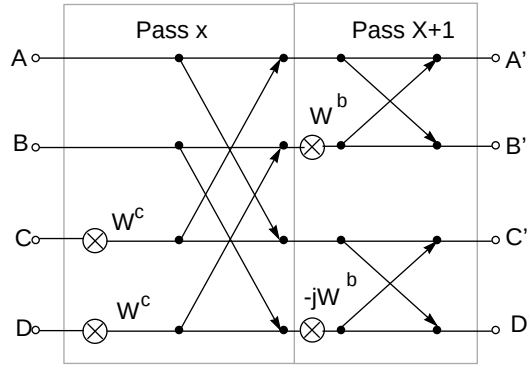


Figure 4-11 A flow diagram of two stages in a radix-2 DIT butterfly —four complex multiplications are involved in the computation.

There are four complex multiplications required which can be reduced to three by combining them into a radix-4 butterfly. Eqn. 4-5 shows two-stage radix-2 butterfly calculations.

$$
\begin{aligned}
A' &= A + CW^{c} + (BW^{b} + DW^{c}W^{b}) \\
B' &= A + CW^{c} - (BW^{b} + DW^{c}W^{b}) \\
C' &= A - CW^{c} - j(BW^{b} - DW^{c}W^{b}) \\
D' &= A - CW^{c} + j(BW^{b} - DW^{c}W^{b})
\end{aligned}
\qquad \text{Eqn. 4-5}
$$

Let $W^{b}W^{c} = W^{d}$, which gives us Eqn. 4-6. A new flow diagram for radix-4 DIT FFT results as shown in Figure 4-12. Three twiddle factors are needed. W_a and W_b originally come from the radix-2 DIT FFT; W_c is new for the radix-4 FFT. Note that the radix-4 DIT butterfly accesses 1/3 more twiddle factors than the radix-2 does.

$$
\begin{aligned}
A' &= A + CW^{c} + (BW^{b} + DW^{d}) \\
B' &= A + CW^{c} - (BW^{b} + DW^{d}) \\
C' &= A - CW^{c} - j(BW^{b} - DW^{d}) \\
D' &= A - CW^{c} + j(BW^{b} - DW^{d})
\end{aligned}
\qquad \text{Eqn. 4-6}
$$

Since each butterfly takes four complex inputs and generates four complex outputs, the number of groups in a pass is reduced to N/4. Also, the number of passes is reduced to $\log_4(N)$. Theoretically, the lower boundary for radix-4 DIT FFT is:

$$TRIV \times N/4 + (\log_4(N) - 1) \times N/4 \times BFLY$$

Twelve multiplications, fourteen additions, and eight subtractions are required for a radix-4 DIT butterfly, as Eqn. 4-7 illustrates.

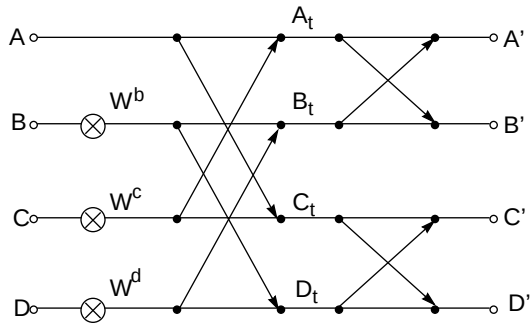


Figure 4-12 A flow diagram of a Radix-4 DIT butterfly. 12 multiplications and 22 additions or subtractions are required.

$$Atr = Ar + CrW_r^c - CiW_i^c$$

$$Ati = Ai + CrW_i^c + CiW_r^c$$

$$Btr = BrW_r^b + DrW_r^d - BiW_i^b - DiW_i^d$$

$$Bti = BrW_i^b + DrW_i^d + BiW_r^b + DW_r^d$$

$$Ctr = Ar - CrW_r^c + CiW_i^c$$

$$Cti = Ar - CiW_r^c - CrW_i^c$$

$$Dtr = BrW_r^b - DrW_r^d - BiW_i^b + DiW_i^d \qquad \text{Eqn. 4-7}$$

$$Dti = BrW_i^b - DrW_i^d + BiW_r^b - DiW_r^d$$

$$Ar' = Atr + Btr$$

$$Ai' = Ati + Bti$$

$$Br' = Atr - Btr$$

$$Bi' = Ati - Bti$$

$$Cr' = Ctr + Dti$$

$$Ci' = Cti - Dtr$$

$$Dr' = Ctr - Dti$$

$$Di' = Cti + Dtr$$

```
;r0->A,r4->B, r1->C, r6->D;
;r1->A', r3->B', r5->C', r7'->D';
;n0=n4=4,n4=2;
;n2=n3=n5=n7=N/8.


                  move                x:(r4)+n4,d3.s  y:,d5.s
                  move                x:(r4)+n4,d1.s  y:,d2.s
                  faddsub.s   d1,d3   x:(r0),d7.s
                  faddsub.s   d5,d2   x:(r1),d0.s     d1.s,y:(r7)
                  faddsub.s   d7,d0   d3.s,d4.s   y:(r1)+n1,d1.s
                  faddsub.s   d7,d5   x:(r4),d6.s y:(r0)+n0,d3.s
                  faddsub.s   d0,d4   d7.s,x:(r3) y:(r4)+n4,d7.s

  do      #N/4,_end_r4
                  faddsub.s   d3,d1   x:(r6)+,d9.sy:,d8.s
  fmpy.s  d6,d9,d5                    d5.s,x:(r7)
  fmpy    d7,d8,d3  faddsub.s   d1,d2   d4.s,x:(r5)     d3.s,d4.s
  fmpy    d6,d8,d1  fadd.s      d5,d3   d0.s,x:(r2)+n2  d1.s,y:
  fmpy.s  d7,d9,d5                    x:(r6)+,d9.s    y:,d8.s
                  fsub.s      d1,d5   x:(r4)+n4,d6.s  y:,d7.s
  fmpy.s  d6,d9,d1                                    y:(r7),d0.s
  fmpy    d7,d8,d2  faddsub.s   d4,d0   d2.s,y:(r5)+n5
  fmpy    d6,d8,d0  fadd.s      d2,d1   x:(r1),d6.s d0.s,y:(r7)+n7
  fmpy    d7,d9,d2  faddsub.s   d1,d3   x:(r6)+,d9.s    y:,d8.s
  fmpy    d6,d9,d0  fsub.s      d0,d2   y:(r1)+n1,d7.s
  fmpy    d7,d8,d3  faddsub.s   d5,d2   d3.s,d4.s d4.s,y:(r3)+n3
  fmpy    d7,d9,d1  fadd.s      d3,d0   x:(r0),d7.s     d1.s,y:(r7)
  fmpy    d6,d8,d3  faddsub.s   d7,d0
                  faddsub.s   d7,d5
                  faddsub.s   d0,d4   d7.s,x:(r3)     y:(r4),d7.s
                  fsub.s      d3,d1   x:(r4)+n4,d6.sy:(r0)+n0,d3.s
_end_r4

                  faddsub.s   d3,d1   d5.s,x:(r7)
                  faddsub.s   d1,d2   y:(r7),d6.s
                  move                d0.s,x:(r2)     d1.s,y:
                  faddsub.s   d3,d6   d4.s,x:(r5)     d2.s,y:
                  move                                d6.s,y:(r7)
                  move                                d3.s,y:(r3)
```

Figure 4-13 Radix-4 DIT Butterfly takes 17 instructions on the DSP96002

For example, if there are 1024-point complex inputs, 8 x 256 + 4 x 256 x 14 =16,384 instructions may be required to improve performance by 11% if compared with 1024-point radix-2 DIT FFT. Here assume, TRIV = 8 and BFLY = 14 since eight ADD||SUB and six ADD instructions are theoretically required for such a butterfly calculation. One important fact is that BFLY, (the number of instruction cycles for butterfly calculation) in a radix-4 DIT FFT must be less than 16, otherwise, there is no advantage for using radix-4 over radix-2. Due to an insufficient number of operations code, FMPY// ADD//SUB instruction only works with destination registers D0 to D3 on the DSP96002.

4.10.2 Radix-4 DIF Butterfly Core

Using the same derivation, a radix-4 DIF butterfly can be obtained. Although the number of multiplications and additions is the same as the radix-4 DIT butterfly, the sequence of data appears differently. Eqn. 4-9 shows an expanded form of the radix-4 DIF butterfly. Eighteen instructions are used to code the radix-4 DIF butterfly.

$$Ar' = Ar + Br + (Dr + Cr)$$

$$Ai' = Ai + Bi + (Di + Ci)$$

$$Cr' = [(Ar - Br) - (Dr - Cr)]W_r^c + [(Ai - Bi) - (Di - Ci)]W_i^c$$

$$Ci' = [(Ai - Bi) - (Di - Ci)]W_r^c - [(Ar - Br) - (Dr - Cr)]W_i^c$$

$$Br' = [(Ar + Bi) - (Di + Cr)]W_r^b + [(Ai - Br) + (Dr - Ci)]W_i^b$$

$$Bi' = [(Ai - Br) + (Dr - Ci)]W_r^b - [(Ar + Bi) - (Di + Cr)]W_i^b$$

$$Dr' = [(Ar - Bi) + (Di - Cr)]W_r^d + [(Ai + Bi) - (Dr + Ci)]W_i^d$$

$$Ci' = [(Ai + Br) - (Di + Cr)]W_r^d - [(Ar - Bi) + (Di - Ci)]W_i^d$$

Eqn. 4-8

4.11 Inverse FFT

The Inverse Fast Fourier Transform (IFFT) is defined in Eqn. 4-9

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j2\pi kn/N} \qquad \text{Eqn. 4-9}$$

The differences between inverse FFTs and forward FFTs are in the scaling factor, N, and the conjugated twiddle factors. A common method of implementing the IFFT is to change the sign of the sine table values and use the FFT subroutine to get the IFFT. Alternatively, one can swap real and imaginary parts, use swapped inputs to the regular

FFT program, and then divide every real and imaginary output by N. Eqn. 4-10 and Eqn. 4-11 show the equality. Eqn. 4-10 shows the inverse FFT.

$$(A_r + jA_i)(W_r + jW_i) = (A_rW_r - A_iW_i) + j(A_iW_r + A_rW_i)$$

Eqn. 4-10

When swapping real and imaginary parts at the input and using forward FFT twiddle factors, we have the relation shown in Eqn. 4-11.

$$(A_i + jA_r)(W_r - jW_i) = j(A_rW_r - A_iW_i) + (A_iW_r + A_rW_i)$$

Eqn. 4-11

Eqn. 4-11 shows that the real part of the IFFT is in the space used for imaginary memory in the forward FFT and the imaginary part of the IFFT is in the real part of the forward FFT. ■