MOTOROLA

APR4/D
Rev. 3

Implementation on Fast Fourier Transforms on Motorola's Digital Signal Processors

Motorola's High-Performance DSP Technology

Implementation of Fast Fourier Transforms on Motorola's Digital Signal Processors

by
Guy R. L. Sohie and Wei Chen
Digital Signal Processing Division

Preface

The human body has inherently slow perception mechanisms. For instance, when listening to music, or speech; we do not hear individual pressure variations of the sound as they occur so quickly in time. Instead, we hear a changing pitch, or frequency. Similarly, our eyes do not “see” individual oscillations of electromagnetic fields (light); rather, we see colors. In fact, we do not directly perceive any fluctuations (or oscillations) which change faster than approximately 20 times per second. Any faster changes manifest themselves in terms of the frequency or rate of change, rather than the change itself. Thus, the concept of frequency is as important and fundamental as the concept of time.

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Illustrations

Illustrations

Illustrations

Tables

SECTION 1

Introduction to the Fourier Integral

1.1 Definition and History

". . . a digital signal processor can efficiently compute the Fourier transform and perform specific frequency-domain tasks. . ."

The scientific and engineering communities have attempted to represent changing signals in two fundamental domains: time and frequency. Temporal changes are easily shown on oscilloscopes, for instance, where change in time is directly proportional to distance across a screen. Representation of signals in terms of frequencies falls under the general category of "spectrum analysis", and has generated a lot of attention recently, due to the increased availability of hardware which makes such representations possible. The first formal approach to spectrum analysis probably dates back to the work of Fourier, who showed how to mathematically represent a general class of time-varying phenomena in terms of sine and cosine functions of particular frequencies. His work is best known as the Fourier Integral (inverse Fourier transform) (see Reference 1):

$$\chi(t) = \int_{+\infty}^{+\infty} X(f)e^{j2\pi ft}dt \qquad \text{Eqn. 1-1}$$

where: $j = \sqrt{-1}$ and $e^{j2\pi ft} = \cos(2\pi ft) + j\sin(2\pi ft)$

When interpreted as an infinite summation, the previous integral is simply a linear combination of a number of sine and cosine functions (expressed by the complex exponential), each one of which is weighted by the complex amplitude X(f). Conversely, the complex frequency function X(f) can be derived from the time-varying signal $\chi(t)$ by the Fourier Transform:

$$X(f) = \int_{+\infty}^{+\infty} \chi(t)e^{-j2\pi ft}dt \qquad \text{Eqn. 1-2}$$

The two expressions shown in Eqn. 1-1 and Eqn. 1-2 define a Fourier transform pair $\chi(t)$ and X(f). The Fourier transform X(f) determines the frequency content of the signal in question, while $\chi(t)$ shows the way the signal varies as a function of time. Note that, in general, $\chi(t)$ can be directly measured (for instance, displayed on an oscilloscope). X(f) remains a mathematical expression which attempts to express our intuitive perception of frequency.

Unfortunately, it is not always true that the concept of frequency, as defined by the Fourier transform in Eqn. 1-2, and the intuitive concept of frequency as we perceive it, are identical. For instance, music consists of tones (frequencies) which vary over time. Although we can clearly perceive time-varying frequencies, Eqn. 1-2 does not allow for Fourier's concept of frequency to have any time-varying character— X(f) is a function of frequency only.

1.2 Use of the Fourier Transform

Because of the basic nature of the frequency concept, practical applications of the Fourier transform are abundant. As more cost-efficient methods become available to compute the Fourier transform, the number of practical solutions to frequency-based problems will grow even larger. In these frequency-based applications, a digital signal processor can efficiently compute the Fourier transform (as defined in ***SECTION 1.1 Definition And History***), and perform specific frequency-domain tasks such as elimination of certain frequency components, etc.

Three general types of Fourier transform applications are:

1. Number-Based — Most spectrum analysis applications require the direct evaluation of the Fourier transform as in Eqn. 1-2. Since the Fourier transform is a mathematical expression, these applications are based on numerical computations, and can be termed number-based. Examples range from spectrum analysis laboratory instrumentation and professional audio equipment to velocity estimation in radar. Note that in number-based applications the accuracy of the computed numbers is of vital importance to the performance of the overall system. For instance, the quality-conscious audio industry requires 16-bit or more precision in order to eliminate audible distortion.

2. Pattern-Based — Many problems involve the recognition and detection of signals with a specific frequency content (a predefined spectral pattern). For instance, speech consists of segments of sound with very specific frequency characteristics. In this type of application, the conversion to the frequency domain is often only a single step in the overall task. It is important that this conversion process be as fast as practical, to allow for sufficient time to perform computationally intensive pattern matching techniques. In addition to providing fast Fourier transform computations, the processor in question needs to be fast at general-purpose DSP tasks so that it can perform a variety of frequency-based calculations for pattern matching.

3. Convolution-Based — The third class of applications of Fourier transforms uses the transform as a simple mathematical tool to perform general filtering in a very efficient manner. This concept is based on the property that the Fourier transform of the convolution of two time-signals:

$$y(t) = \int_{+\infty}^{+\infty} \chi(t-\tau)h(\tau)d\tau \qquad \text{Eqn. 1-3}$$

is equal to the product of the individual transforms:

$$Y(f) = X(f)H(f) \qquad \text{Eqn. 1-4}$$

Eqn. 1-3 (better known as the convolution integral) represents the output of a linear filter with impulse response h(t) and input signal x(t). Clearly, in the frequency domain, the output of a filter can be obtained by a simple multiplication, whereas in the time domain, a more complicated convolution integral needs to be solved. The amount of computation involved in evaluating the integral in Eqn. 1-3 becomes particularly large when the impulse response h(t) has a long time duration which often prevents real-time implementation. Clearly, if the Fourier transform X(f) of the signal can be computed efficiently, the filtering operation itself can be achieved by simple multiplications.

The combined number of computations (for computing the Fourier transform, for filtering in the frequency domain, and for obtaining the inverse Fourier Transform) is often less than the total number of calculations required to compute Eqn. 1-3 directly. This is especially true when the filter in question performs a simple frequency discrimination function (lowpass, bandpass, highpass, bandreject, etc.). In this case, the multiplications in the frequency domain can be replaced by a simple masking operation, which removes the stopbands and leaves the passband(s) unchanged.

Although no direct frequency information is extracted from the signal, the Fourier transform is used as a mathematical tool for fast-filtering applications. Note that again, fast Fourier transform and inverse Fourier transform “engines” are needed in order to provide the real-time filtering operation.

In summary, the basic nature of the frequency concept indicates that the number of possible frequency domain applications is as large as more conventional time domain applications. In the past, frequency domain applications were either difficult to implement or could not be realized in a cost-efficient manner because of the lack of low-cost, high-performance hardware. This application note demonstrates that the DSP56001/2 and the DSP96002 Families of digital signal processors fulfill the demanding requirements imposed by frequency domain problems. In addition to providing a fast implementation of high-precision Fourier transform computations, the general-purpose nature of the instruction set allows for a ***complete, single-chip, low-cost*** integrated solution to a wide variety of frequency domain problems. ■

SECTION 2

The Discrete Fourier Transform

2.1 The Discrete-Time Fourier Transform (DTFT)

“. . . the results need to be available within a finite time period, and the infinite summation must somehow be reduced to a finite summation.”

In order to compute the Fourier transform using digital hardware, Eqn. 1-2 needs to be approximated in a manner which makes machine computation feasible. The first step in this process consists of eliminating the theoretical integral symbol, and replacing it by a computable sum:

$$X(f) \approx \tilde{X}(f) = T \sum_{n=-\infty}^{+\infty} \chi(nT)e^{-j2\pi fnT} \qquad \text{Eqn. 2-1}$$

The above expression uses a sampled signal $\chi(nT)$, where the sampling period T is made as small as possible to reduce approximation errors. Appropriately, $\tilde{X}(f)$ is called the discrete-time Fourier transform (DT-FT). As T (the sampling period) becomes infinitely small, the previous summation approaches the original Fourier transform in Eqn. 1-2. To assess the accuracy of this approximation, note that the resulting expression for $\tilde{X}(f)$ is a periodic function of frequency:

$$\tilde{X}(f) = \tilde{X}\left(f + \frac{1}{T}\right) \qquad \text{Eqn. 2-2}$$

because:

$$e^{-\left(j2\pi fnT+j2\pi n\frac{T}{T}\right)} = e^{-j2\pi fnT}e^{-j2\pi n} = e^{-j2\pi fnT} \qquad \text{Eqn. 2-3}$$

In general, the original spectrum X(f) is not periodic, and the approximation is only justified for a range of small values of f. In Figure 2-1, the DTFT magnitude and the Fourier transform magnitude of a simple rectangular function are shown for several values of the sample rate $f_s = 1/T$. Note the periodic nature of the resulting function, as well as the approximation errors due to the sampling process.

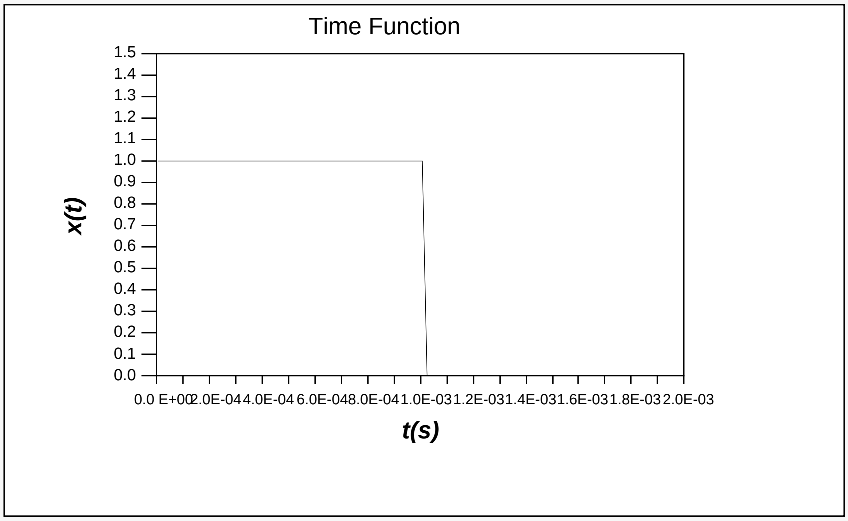
The Nyquist sampling theorem gives a well accepted criterion for the sampling rate. It states that a signal needs to be sampled faster than twice its highest frequency. In other words, if:

$$X(f) = 0 \qquad \text{Eqn. 2-4}$$

for $|f| \geq B$ (B is referred to as the bandwidth of the signal), then the sampling frequency needs to satisfy:

$$f_s \geq 2B \qquad \text{Eqn. 2-5}$$

In practice, signals rarely satisfy Eqn. 2-5, and some error, called the aliasing error, can be expected in the evaluation of X(f). The aliasing error is generated by frequency components at higher frequencies, which manifest themselves at lower frequencies because of the periodic nature of $\tilde{X}(f)$ (aliasing). The aliasing error can be reduced by filtering out the higher-frequency components of the signal using a low-pass anti-aliasing filter and/or by increasing the sampling rate.



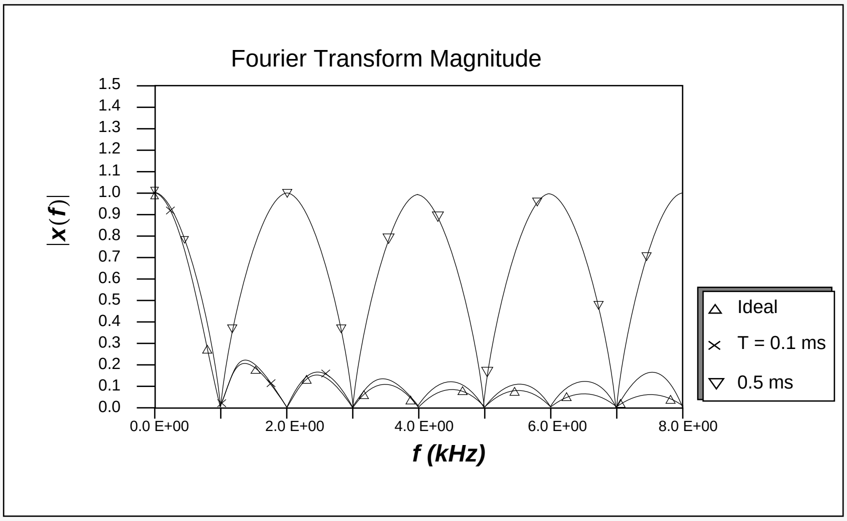


Figure 2-1 Fourier Transform of a Rectangular Function

2.2 Windowing and Windowing Effects

The discussion of aliasing errors illustrates how the Fourier transform can be approximated by an infinite summation. In practice, the results need to be available within a finite time period, and the infinite summation must somehow be reduced to a finite summation. One obvious way to reduce the infinite summation is by simply truncating the sum in Eqn. 2-2 to N terms as:

$$\tilde{X}_W(f) = T \sum_{n=0}^{N-1} \chi(nT) e^{-j2\pi fnT} \qquad \text{Eqn. 2-6}$$

This truncation is frequently referred to as “windowing” because an infinite summation is viewed through a finite window. The resulting transform is called the windowed discrete-time Fourier transform (WDTFT). In mathematical terms, windowing is simply the multiplication of the signal by a window sequence of finite-length, w(n). In the simple case above, w(n)=1 for $0 \le n \le N-1$; otherwise, w(n)=0. Because of its rectangular shape, the window shown above is called the rectangular window.

Unless the signal in question is of finite duration, this truncation will introduce other errors, resulting in a number of artifacts in the spectrum. To assess the effect of the windowing operation, a simple sine wave of the form:

$$\chi(t) = \sin(2\pi 1000t) \qquad \text{Eqn. 2-7}$$

is sampled with a sampling frequency of 4000 Hz, and the windowed DTFT is computed with N=20.

Figure 2-2 shows the result of windowing a sine wave by a rectangular window. Windowing causes the following errors:

1. Leakage — Even though the input signal consists of a single-frequency component at 1000 Hz, the result clearly shows components at frequencies other than 1000 Hz. This is called the leakage effect: it appears as if energy has “leaked” from 1000 Hz to the rest of the spectrum.
2. Smoothing — Although the theoretical transform exhibits an infinitely narrow, and infinitely large peak at 1000 Hz, the actual peak has finite magnitude and exhibits finite width. It appears that the narrow peak has been “smeared” out in the frequency domain as a result of the windowing function in the time domain. This effect is appropriately termed the smoothing effect.
3. Ripple — The overall magnitude plot in Figure 2-2 shows an oscillatory character not present in the original Fourier transform: this is called the ripple effect. The origin of the ripple effect lies in the discontinuity (abrupt start and end) introduced in the signal by the window. Windows with more gradual transitions generally have lower sidelobes and less ripple.

In general, a tradeoff exists between these different effects, and the advantages of an appropriate windowing function can be chosen for a specific application. For an excellent summary of existing windowing functions and their properties, see Reference 2.

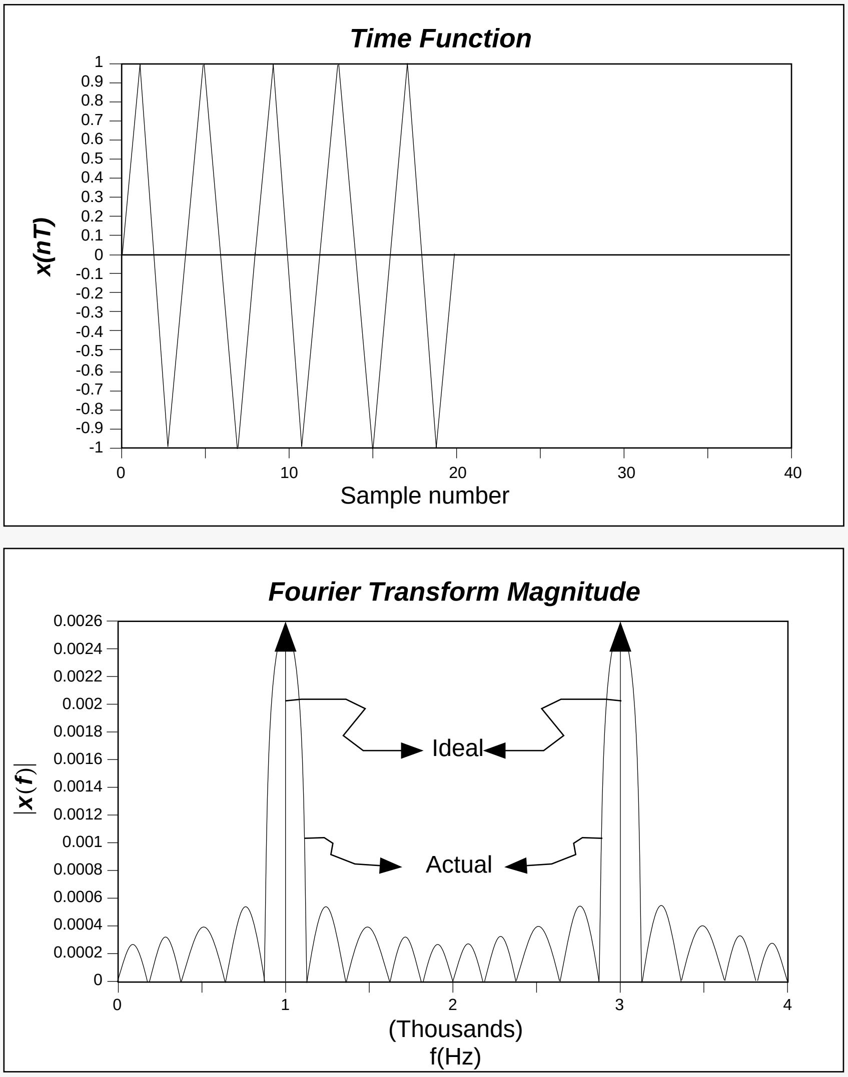


Figure 2-2 Windowing Effects When Windowing a Single Sine

2.3 Sampling the Frequency Function

The windowed DTFT is now ready for machine computation, with one exception: the independent frequency variable f is still a continuous variable, and needs to be captured in discrete intervals, or sampled. Since the DTFT is periodic in the frequency domain with period f_s, only values of f from 0 to f_s (the sampling frequency) need to be computed. Although there are similar arguments concerning the distance between successive frequency samples as in the case of time-sampling, it turns out that when the WDTFT is sampled every f_s/N Hz, fast algorithms for computing the transform can be derived. Note that in this case, the number of samples in the window (N) and the number of samples in the frequency domain (N) are equal. The resulting transform is called the discrete-time Fourier series (DTFS):

$$\tilde{X}_N(k) = T\sum_{n=0}^{N-1}\chi(nT)e^{-j\frac{2\pi}{N}nk} \qquad \text{Eqn. 2-8}$$

The inverse DTFS is given by:

$$\chi_N(k) = \frac{1}{NT}\sum_{k=0}^{N-1}\tilde{X}_N(k)e^{j\frac{2\pi}{N}nk} \qquad \text{Eqn. 2-9}$$

Keep in mind that the values of the frequency samples of f_k are equal to $[f_s/N]$ k.

Note that many textbooks simply define the Discrete Fourier transform (DFT) $X_N(k)$:

$$X_N(k) = \sum_{n=0}^{N-1} \chi(nT) e^{-j\frac{2\pi}{N}nk} \qquad \text{Eqn. 2-10}$$

with inverse transform:

$$\chi_N(n) = \frac{1}{N} \sum_{n=0}^{N-1} X_N(k) e^{j\frac{2\pi}{N}nk} \qquad \text{Eqn. 2-11}$$

Obviously, the DFT and DTFS differ only by a scaling factor of T, making the spectrum independent of the sampling period. Consequently, explicit T dependence can be dropped from Eqn. 2-11.

Although the sequence $x_N(n)$ corresponds to the original sampled and windowed sequence $\chi(nT)$ for sampling instants 0 through N-1, the complete sampled sequence $\chi(nT)$ for any n cannot necessarily be recovered from it. Indeed, $x_N(n)$ appears to be periodic with period N due to

the periodicity of $e^{j\frac{2\pi}{N}nk}$, whereas the original sampled signal was not assumed to be periodic.[1]

[1] The error introduced in the time domain by sampling a frequency function is termed “aliasing in time” which is analogous to the “aliasing in frequency” caused by sampling a time function. (See SECTION 2.1 The Discrete-Time Fourier Transform (DTFT)). That is, if a frequency spectrum is not sampled densely or closely enough, the signal constructed in the time domain through the inverse “discrete-frequency Fourier transform” will show some distortion.

This must be kept in mind in convolution-based applications, where the forward as well as inverse transforms are used; the incoming signal stream needs to be segmented, and the computed signal segments need to be pieced together to construct the complete output stream. Most basic text-books on digital signal processing discuss techniques for piecing together the output stream (see Reference 3). ■

SECTION 3

The Fast Fourier Transform

3.1 Motivation

"Since there are two independent variables (time and frequency) in the Fourier transform, dividing (or decimating) the DFT into smaller ones can be done in two ways."

Upon closer examination of Eqn. 2-10, it becomes clear that for every frequency point, N-1 complex summations and N complex multiplications need to be evaluated. Since there are N frequency points to be evaluated, this gives a total of N(N-1) complex sums, and N^2 complex multiplications. Counting two real sums for every complex one, and four real multiplications plus two real summations for every complex multiplication, gives a total of $4N^2$- 2N real summations and 4 N^2 real multiplications.

The above numbers grow rapidly for increasing N. For N=1024 (1024-point DFT), 4,194,304 real multiplications are required. If this is computed on a DSP56001/DSP56002 with a 27-MHz clock, it takes 0.31 seconds just to execute that many real multiplications. Since the DFT computation needs to be completed by the time the next 1024 data points are collected for real-time performance, the sampling rate is limited to a maximum of 3.3 kHz. Obviously, faster solutions are needed.

3.2 Divide and Conquer

A faster algorithm for computing the DFT can easily be derived. The principle behind this is very simple. As illustrated in Figure 3-1, a square of half the linear dimension of a larger square has one-fourth the surface area. This is because the surface area is proportional to the square of the linear dimensions of the square. Similarly, the number of multiplications needed to compute the DFT is proportional to the square of the DFT's length (N). Thus, if we could replace the DFT over N points by two DFTs over N/2 points, computations would be reduced in order of magnitude of 0.5 (=0.25+0.25).

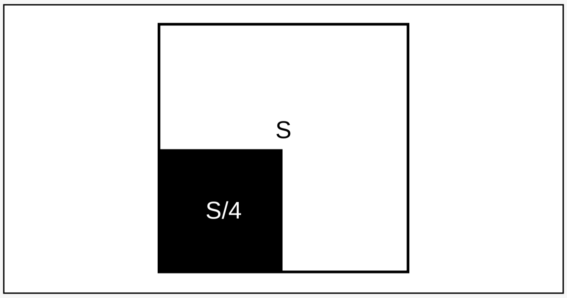


Figure 3-1 The FFT principle in layman's terms

Since there are two independent variables (time and frequency) in the Fourier transform, dividing (or decimating) the DFT into smaller ones can be done in two ways. We can attempt to represent an N-point transform in terms of DFTs over half the number (N/2) of time-samples. This approach is appropriately called the decimation-in-time or DIT approach. Alternatively, the N-point DFT can be represented in terms of DFTs with N/2 frequency samples. This approach is called the decimation-in-frequency or DIF approach.

3.3 The Decimation-in-Time and Decimation-in-Frequency Radix-2 Fast Fourier Transforms

It is easily shown that Eqn. 2-10 can be rewritten when N is even as:

$$X_N(k) = \sum_{r=0}^{(N/2)-1} \chi(2rT) e^{-j\frac{2\pi}{(N/2)}rk} + e^{-j\frac{2\pi}{N}} \sum_{r=0}^{(N/2)-1} X\left[(2r+1)\,T\right] e^{-j\frac{2\pi}{(N/2)}rk}$$

Eqn. 3-1

As illustrated in Figure 3-2, this expression shows how two N/2-point DFTs can be combined to obtain one N-point DFT. If N is an integer power of 2, this process can be repeated, as shown in Figure 3-3 and Figure 3-4, until a simple, two-point DFT is obtained. This gives rise to the flow diagram of a DIT fast Fourier transform (FFT) as shown in Figure 3-5, which represents a complete 8-point FFT computation.

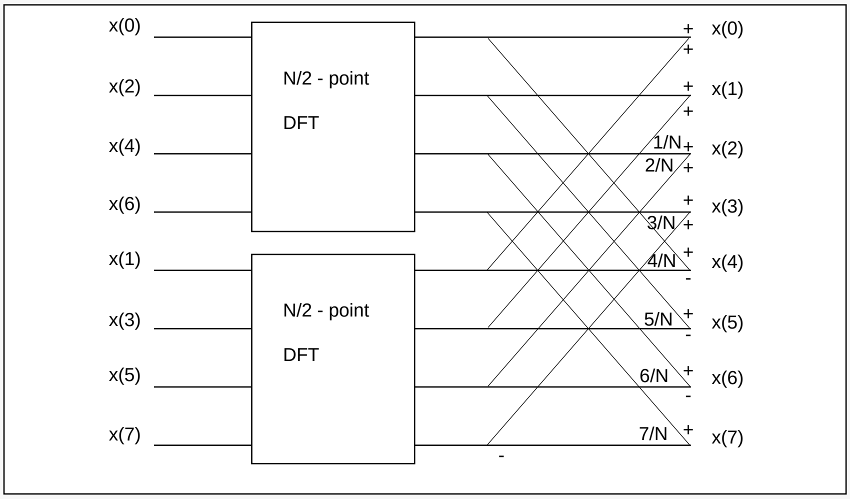


Figure 3-2 Decimation-in-Time of an N-Point

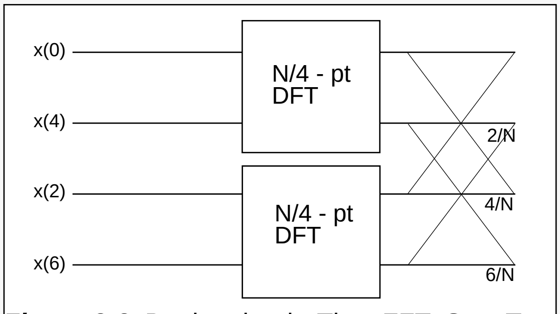


Figure 3-3 Decimation-in-Time FFT: Step Two

NOTE: k/N denotes multiplication by the “twiddle factors” $e^{-j\frac{2\pi}{N}k}$ throughout this document

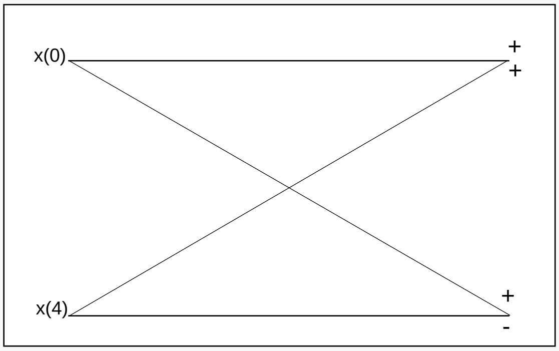


Figure 3-4 Decimation-in-Time FFT: Final Step (2-Point DFT)

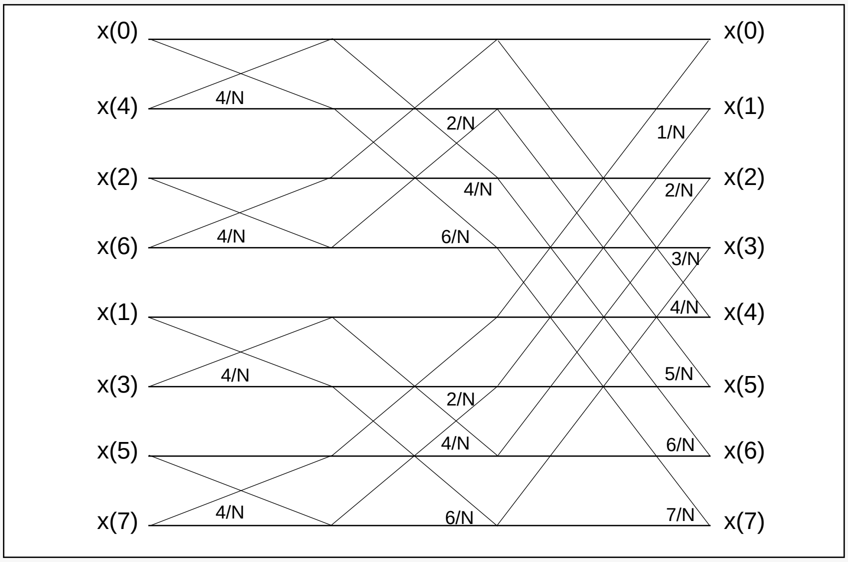


Figure 3-5 An 8-point, radix-2, Decimation-in-Time FFT

The basic flow diagram of Figure 3-5 can be further simplified by rearranging the terms in the basic building block (the butterfly) as in Figure 3-6. Also, it is seen from Figure 3-5 that input samples no longer occur in normal, sequential order. When the indices are represented in their binary equivalent, however, the input samples appear in “bit-reversed” order. Figure 3-8 shows how the diagram can be re-arranged for normally-ordered inputs and bit-reversed outputs.

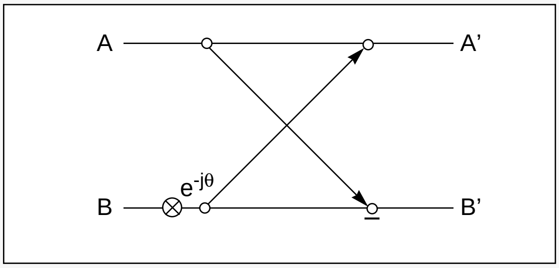


Figure 3-6 Rearrangement of the “butterfly” building block of the DIT FFT

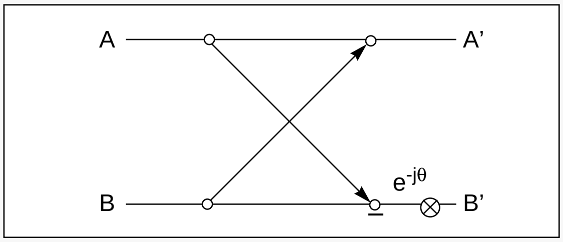


Figure 3-7 Rearrangement of the “butterfly” building block of the DIF FFT

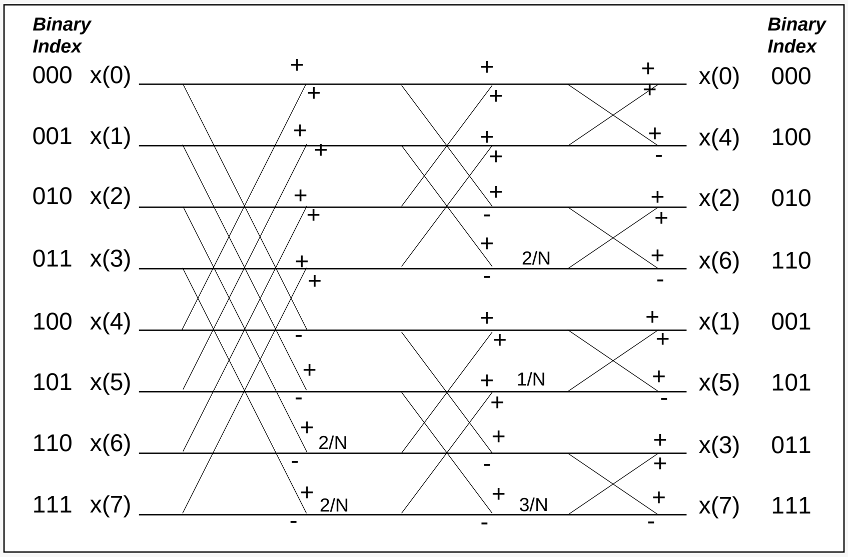


Figure 3-8 Rearrangement of the DIT computation of Figure 3-6

Figure 3-9 and Figure 3-10 show how the DFT with N frequency points can be obtained in terms of DFTs with a smaller number of frequency samples (decimation-in-frequency FFT). Note that the basic building block (butterfly) is different than for the DIT case (see Figure 3-10).

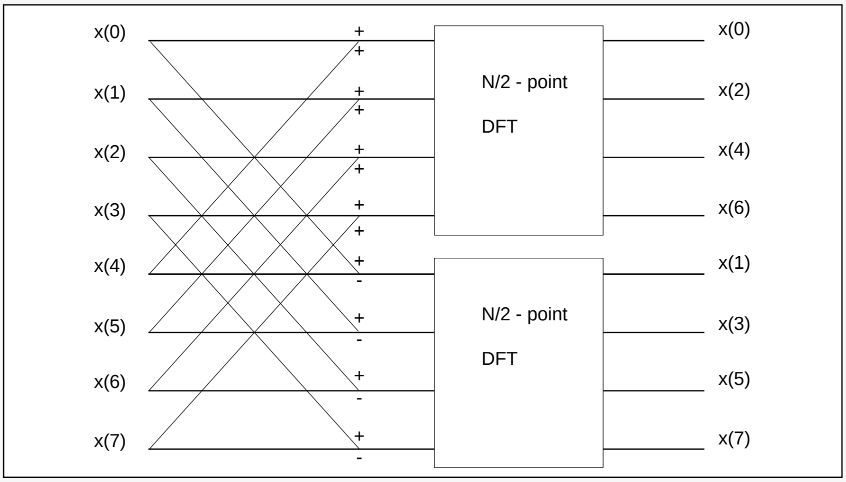


Figure 3-9 Decimation-in-Frequency concept

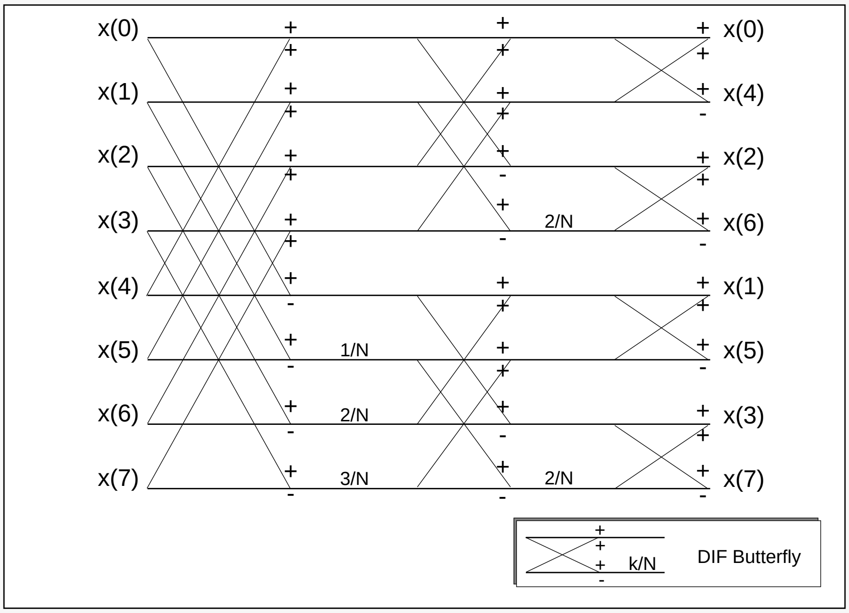


Figure 3-10 Complete 8-point radix-2 DIF FFT

3.4 The Decimation-in-Frequency Radix-2 Fast Fourier Transforms

If Eqn. 2-10 is decomposed from the frequency domain, we can show the following equations exist:

$$X_N(2k) = \sum_{r=0}^{(N/2)-1} [x(r) + x(r+N/2)]e^{-j\frac{2\pi rk}{N/2}}$$

Eqn. 3-2

$$X_N(2k+1) = \sum_{r=0}^{(N/2)-1} [x(r) - x(r+N/2)]e^{-j\frac{2\pi rk}{N/2}} e^{-j\frac{2\pi r}{N}}$$

Eqn. 3-3

The decimation in frequency butterfly is shown in Figure 3-9. ■

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."

*T*he 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 address 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.

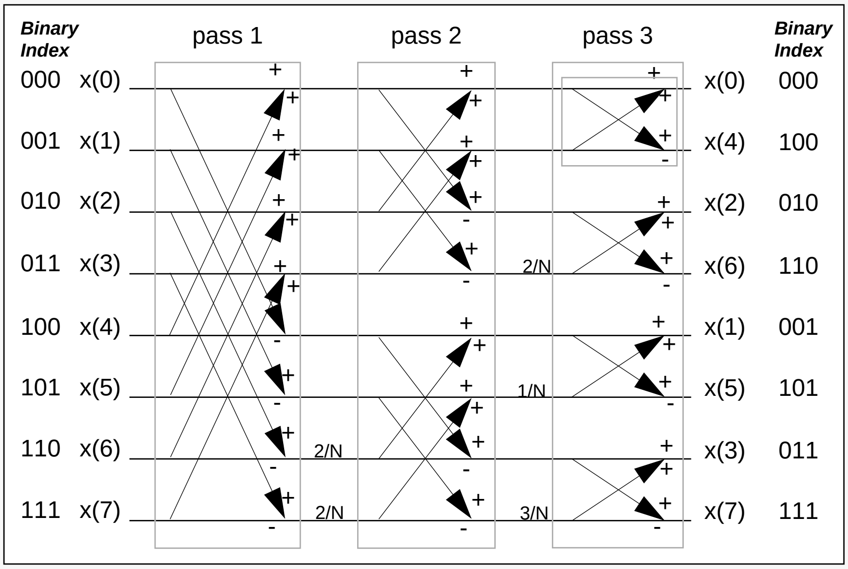


Figure 4-1 Grouping of Butterflies in the FFT Calculation

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.

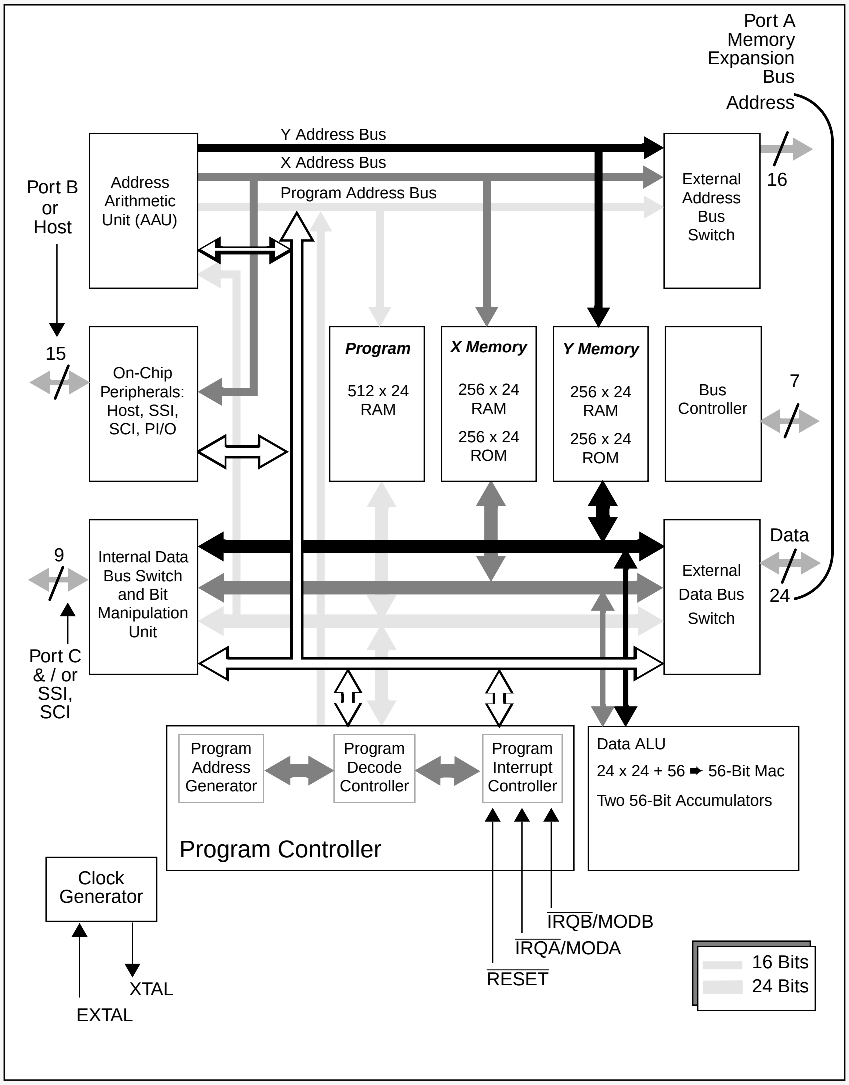
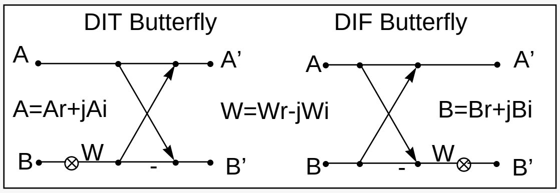


Figure 4-2 DSP56001 Architecture Block Diagram

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.

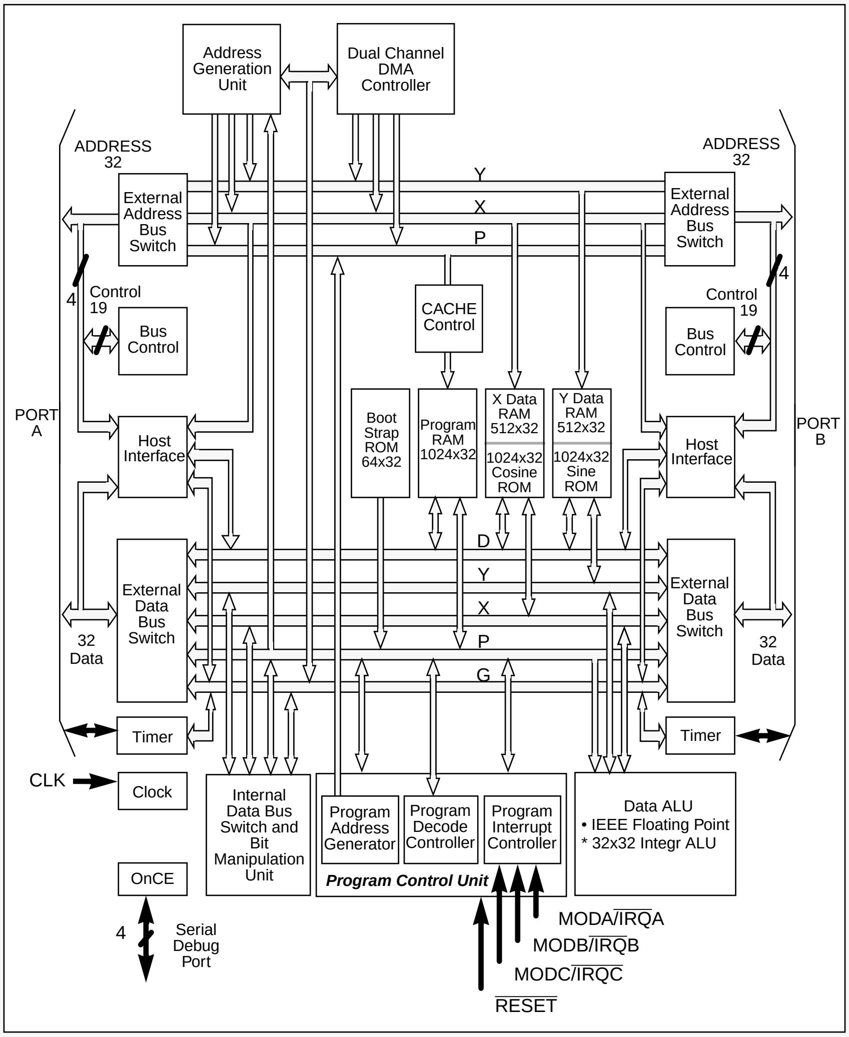


Figure 4-6 DSP96002 Architectural Block Diagram. Two symmetric bus expansion ports with two channel DMA controller that blow away the speed limit on external memory access and data I/O.

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.

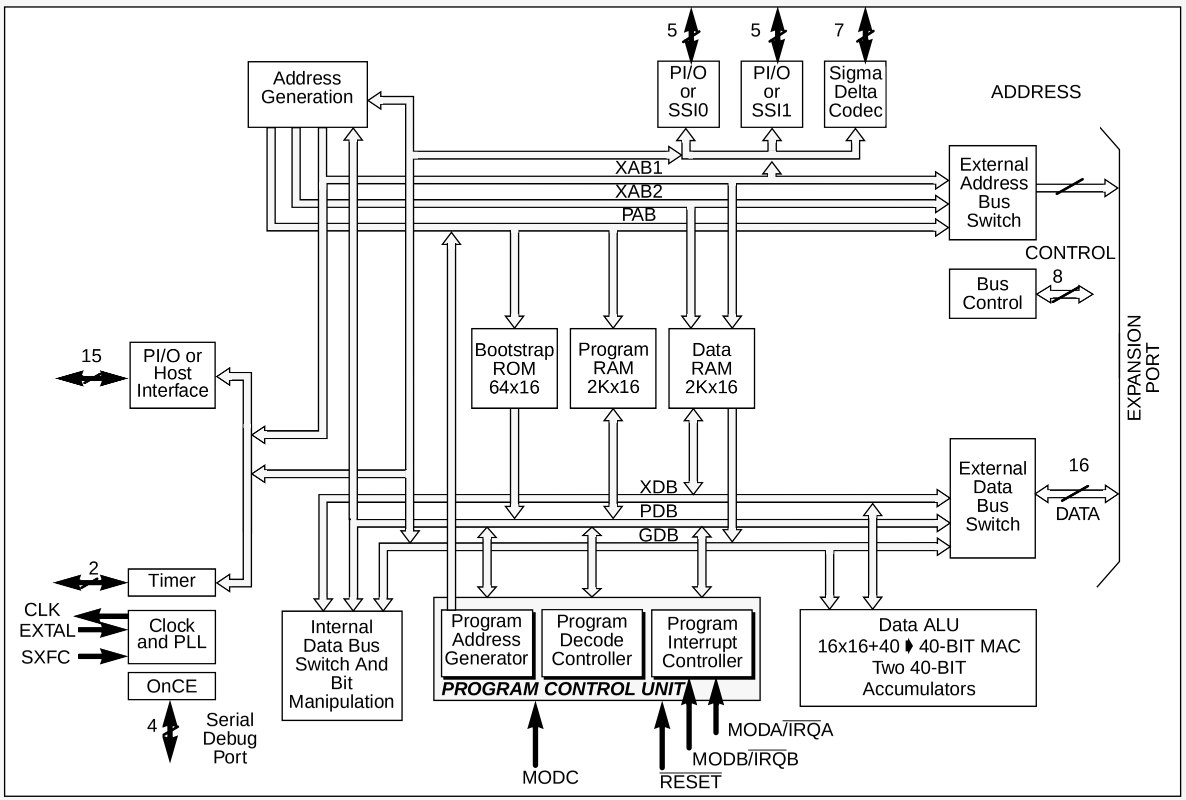


Figure 4-8 DSP56156 architectural block diagram. Only four address registers are available. On the single data memory space, only a dual-read per one instruction is allowed.

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 overflow 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***).

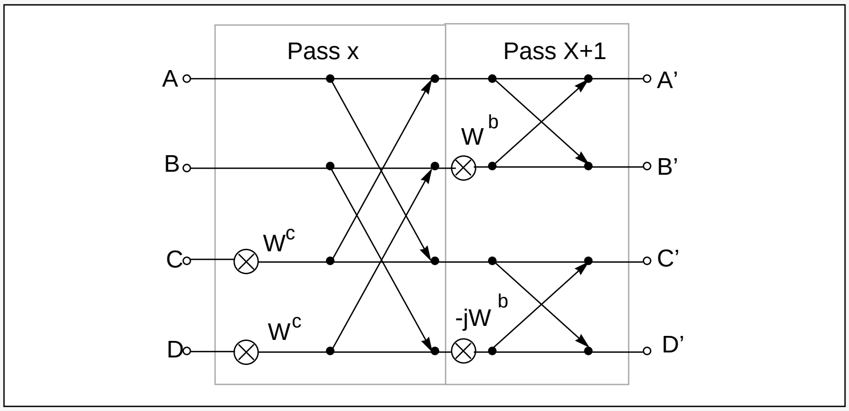


Figure 4-11 A flow diagram of two stages in a radix-2 DIT butterfly —four complex multiplications are involved in the computation.

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.

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

Let $W^bW^c = W^d$, 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.

$$\begin{aligned}A' &= A + CW^c + (BW^b + DW^d)\\ B' &= A + CW^c - (BW^b + DW^d)\\ C' &= A - CW^c - j(BW^b - DW^d)\\ D' &= A - CW^c + j(BW^b - DW^d)\end{aligned}$$ 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:

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

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

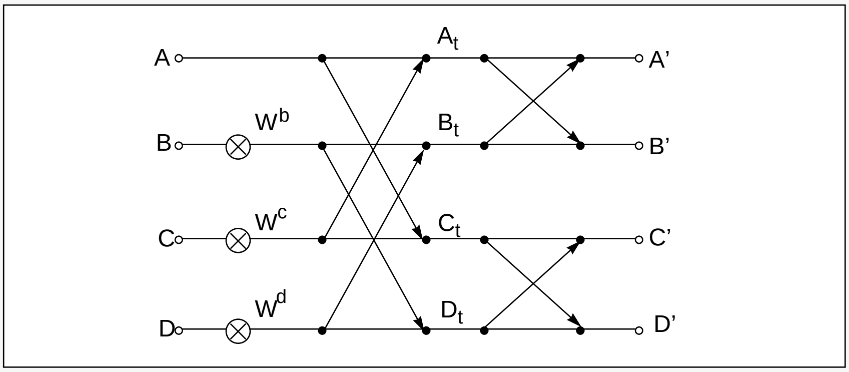


Figure 4-12 A flow diagram of a Radix-4 DIT butterfly. 12 multiplications and 22 additions or subtractions are required.

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

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

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

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

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

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

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

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

$$Ar' = Atr + Btr$$

$$Ai' = Ati + Bti$$

$$Br' = Atr - Btr$$

$$Bi' = Ati - Bti$$

$$Cr' = Ctr + Dti$$

$$Ci' = Cti - Dtr$$

$$Dr' = Ctr - Dti$$

$$Di' = Cti + Dtr$$

```
;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. ■

SECTION 5

Optimizing Performance of the FFT

5.1 Optimization

"Optimization saves . . . 2067 instruction cycles which equals about 10% cycle time of the optimized code."

Judging the performance of any program requires the consideration of both its time and space complexity. There is always a trade off between these two aspects. Time complexity indicates how fast an algorithm can be implemented on a specified microprocessor, while space complexity tells how much memory may be required. Optimization can either reduce memory requirement or minimize runtime of an algorithm. Since memory costs are continually decreasing, time optimization becomes more and more important.

One way to evaluate the time complexity of an algorithm is to compare its theoretical complexity, ideal implementation complexity, and practical complexity. Theoretical complexity refers to the number of additions and multiplications required by the given algorithm, independent of the microprocessor's architectures. This type of evaluating is only good for high-level comparison among algorithms and does not reflect the real performance of the algorithm on a given microprocessor. Not surprisingly, an algorithm that retains a lower theoretical complexity has a higher ideal implementation complexity. Ideal implementation complexity considers only the implementation of the core algorithm by the given microprocessor's instruction capabilities, such as available instruction type, addressing mode, parallel data move, etc. Ideal implementation complexity indicates the non-overhead performance of a given algorithm on a microprocessor, and always provides an optimistic estimation of an algorithm's performance. Practical complexity denotes the ideal implementation complexity plus the structure overhead of the microprocessor. (Structure overhead includes all required instructions not associated with the core algorithm.) Moving pointers, setting up DO loops, jumps to subroutines, and conditional jumps are typical structure overhead in microprocessors.

By distinguishing the different complexities, one can easily determine which microprocessor is competent for each aspect, and which instruction or address mode is critical to the specific algorithms. Also, chip designers may derive clues from the complexity analysis for determining which instruction or address mode should be added to the next revision. For example, the DSP96002 supports FMPY||ADD||SUB — an instruction with two parallel moves. The theoretical complexity of a radix-2 butterfly is four real multiplications and six additions or subtractions. Thus, the ideal implementation complexity of a radix-2 FFT on the DSP96002 is four instruction cycles. If each butterfly needs an average of 0.25 instructions to set up a pointer or DO loop, etc., the practical complexity of radix-2 is 4.25 instructions. The ratio of ideal implementation complexity to practical complexity reflects the efficiency of a microprocessor to perform a specific function. For example, the efficiency of the DSP96002 performing a radix-2 complex FFT could be:

$$\text{efficiency} = \frac{\text{ideal implementation complexity}}{\text{practical complexity}} = \frac{4}{4.25} = 0.94$$

Eqn. 5-1

In other words, the structure overhead for this particular example is about 6%. For FFTs implemented on programmable DSPs, the structure overhead should be between 3% and 15%. If a DSP has structure overhead higher than 15%, it can not be called a DSP. If one claims a structure overhead lower than 3%, it is probably an application specific integrated circuit (ASIC).

5.1.1 Minimum Memory Requirement — In-Place Calculation

Although each radix-2 butterfly has two complex input data and two complex output data, calculation of the butterfly can be done by using only one memory set called in-place calculation. Memory requirements may be minimized by:

- ***Reordering data into bit-reversed order.*** This can be done in-place since data is interchanged by pairs, as seen in Figure 4-9. Thus, only 2N real data locations are required.

- ***Reducing the size of the twiddle factor table*** from N real locations to N/2 real locations for normal order input DIT FFT (see reference 8). Notice that in normal order input DIT FFT the order that accesses the twiddle factor table is bit-reversal, i.e.

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

  N/2 complex numbers can be combined in pairs of two, which differ by a factor $W_N^{N/4} = -j$. In other words, the second twiddle factor in the pair can be obtained by multiplying -j with the first twiddle factor. In fact, this optimization can be implemented with a minor modification to the previous butterfly core. All odd indexed groups will use negated, real and imaginary exchanged twiddle factors from the previous even indexed groups. Therefore, the number of groups in a pass is reduced to half of the previous one and the access time of twiddle factors is also reduced to half of the previous one.

- ***Using a triple-nested DO-loop FFT*** to minimize the program memory space (as seen in Figure 4-5). Items 1 and 2 above save data memory space for the FFT calculation only.

5.1.2 Optimization for Faster Execution

Although the previously discussed program executes very efficiently, some applications may impose less stringent requirements on program memory size, but demand even faster execution. Faster execution can be obtained by further optimizing the previous algorithm. The following pages present several steps to achieve this optimization.

1. Since the first and second passes have trivial twiddle factors:

$$W_N^0 = 1, \text{ and } W_N^{N/4} = -j$$

it is common to combine the first and second passes as one radix-4 pass by calculating N/4 butterflies in the following equations.

$$\begin{aligned}
Ar' &= Ar + Cr + Br + Dr \\
Br' &= Ar + Cr - (Br + Dr) \\
Cr' &= Ar - Cr + (Bi - Di) \\
Dr' &= Ar - Cr - (Bi - Di) \\
Bi' &= Ai + Ci - (Bi + Di) \\
Ci' &= Ai - Ci - (Br - Dr) \\
Ai' &= Ai + Ci + Bi + Di \\
Di' &= Ai - Ci + (Br - Dr)
\end{aligned} \qquad \text{Eqn. 5-2}$$

Notice that there are eight additions and eight subtractions in . A DSP that has a multiplication and accumulation instruction with one or two parallel moves (type A DSP) may take at least sixteen instructions to do . A DSP that has a FMPY||ADD||SUB instruction with two parallel data moves (type B DSP) can do in eight instructions. After combining the first two trivial passes as a radix-4 pass, the number of instructions required in the radix-2 DIT complex FFT becomes:

$$(\mathrm{TRIV} \times N/4) + [(m-2) \times N/2 \times \mathrm{BFLY}]$$

where: TRIV is the number of instructions necessary to perform a trivial butterfly

Theoretically, for the DSP56001/2, the DSP96002, and the DSP56156, TRIV may be 16, 8, and 16 instruction cycles, respectively. Therefore, a 1024-point complex FFT on the DSP96002 can be done in (8 x 256) + (8 x 512 x 4) = 18,432 instruction cycles. This is a lower boundary of the radix-2 complex FFT. In fact, TRIV is 17, 8, and 22 on the DSP56001, the DSP96002, and the DSP56156, respectively. Cycle time of the FFT can be reduced further by exploring the simple relations among the remaining passes.

2. Trivial twiddle factors exist in the remaining passes as well. Special butterflies can take advantage of those simple relations. There are two types of trivial twiddle factors:

Type I $W_N^0 = 1, W_N^{N/4} = -j$

Type II $W_N^{N/8} = -W_N^{(3N)/8} = 0.707 - j0.707$

Type I trivial factors don’t involve multiplications as already shown in Eqn. 5-2. To utilize these simple relations in the remaining passes, different butterflies must be inserted in one pass. This change results in longer program code and some structure overhead, such as updating address registers, different DO loops, and modulo addressing.

Type II trivial factors are not really trivial for either type A or type B DSPs. Type II trivial factors reduce the theoretical complexity of a radix-2 butterfly to two real multiplications and six real additions or subtractions. With only one adder on type A DSPs, six instructions are required as before. The ideal implementation complexity could be 3 for type B DSPs, but unfortunately each radix-2 butterfly deals with four real inputs and four real outputs. Type B DSPs have only two parallel data moves, and each radix-2 butterfly still takes at least four instruction cycles for type II trivial factors. The type II trivial factor issue is addressed here because this is probably the last chance for further optimizing radix-2 FFTs.

Each group in the last pass consisted of a single butterfly. A triple nested DO loop is thus no longer required in this pass: it can be split and handled by a single DO loop.

3. Another alternative is to combine the last two passes into one radix-4 pass. Since each butterfly in the last pass requires a different twiddle factor, one instruction to fetch a twiddle factor must be appended in the butterfly core. The same fetch occurred in the second to last pass in every two butterflies. Combining four radix-2 butterflies into one radix-4 butterfly may save four multiplications but a special twiddle factor table has to be created for the radix-4 butterfly.

4. For longer FFTs (>256 points), internal memory in the DSP56001/DSP56002 is not sufficient to contain the complete data set. Consequently, the butterflies execute more slowly when the processor needs to fetch a data value in external X and in external Y memory in the same instruction cycle. This causes the instruction cycle to be “stretched”, resulting in slower execution time. Through intelligent memory usage, however, this effect can be minimized. In a further optimized routine (see ***Appendix A***), the first two passes are combined into a single pass. Next, separate 256-point FFTs are computed, whereby the data is moved into internal memory, and the results are not moved to external memory until the final pass. This process avoids the stretching of the instruction cycle on the middle passes, and makes optimal use of the available internal memory.

With these optimizations, a significantly faster routine is obtained. For instance, a 1024-point optimized complex FFT routine is available for DSP96002 which executes in 0.94 ms at 40MHz clock (see Fully Optimized Complex FFT in ***Appendix A***). A fully optimized complex FFT routine for DSP56001/2 is also listed in ***Appendix A*** (CFFT56.ASM). 0.704ms is needed to calculate a 512-point complex FFT at 40 MHz clock, which is 8.7% faster than an optimized complex FFT. For more benchmarks see ***SECTION 8***. Note, however, that “straight-line” code always results in longer programs.

5.2 Example of Optimization

5.2.1 *Fully Optimized Complex FFT for the DSP56001/2*

Program CFFT56.ASM in ***Appendix A*** is a good example of optimizing complex FFTs on the DSP56001/2 for fast execution time. Figure 5-1 shows passes, groups and butterflies for a 512-point complex FFT. There is a total of 9 passes. The number of groups in each pass doubles from pass to pass, while the number of butterflies in each group halves from pass to pass. Each pass has the same number of butterflies,.i.e. N/2=256 butterflies.

CFFT56.ASM takes advantage of the trivial twiddle factors in all the passes. Note that pass 0 and 1 can be done by simple radix-4 butterflies. A radix-4 butterfly has been coded by 17 instructions, which is the best case on the DSP56001/2. The parallel data move in this radix-4 butterfly has been deliberately arranged to avoid a dual data move involving external memory, although the first and next to last instruction may result in cycle stretch in some cases. Since half of the 512 data are in external memory, one instruction cycle is stretched, and 18 instruction cycles are used for a 512-point complex FFT. This equals 4.5 instruction cycles per radix-2 butterfly. The same radix-4 butterflies are also applied to passes 2, 3, 4, and 5. Note that in Figure 5-1, the groups highlighted by cross lines are trivial butterflies too, and are not covered by the simple radix-4 butterflies. These data points are calculated by 5-instruction radix-2 butterflies. As shown in Figure 5-1, each pass has 256 radix-2 butterflies and the first seven passes have 860 trivial butterflies. 772 of these radix-2 butterflies require 4.5 instruction cycles (simple radix-4 butterflies) while 88 of them require 5 instruction cycles. Therefore, the total cycle time for trivial butterflies is 772 x 4.5 + 88 x 5 = 3,914 which means a savings of 860 x 6 - 3914 = 1,246 cycles when compared to a non-optimization case. For program simplicity, the above calculation does not utilize the trivial butterflies in passes 7 and 8.

CFFT56.ASM uses N/2 real twiddle factors. This scheme reduces the data memory requirement and also reduces the structure overhead on group DO loops, because the group number in each pass changes to half of the previous scheme.

CFFT56.ASM fully utilizes internal memory to avoid cycle stretch when the DSP56001/2 accesses two data. A 512-point complex FFT is divided into two 256-point parts. The first 256-point part remains in internal memory until the last pass. The second 256-point data loads into internal memory after the first pass and stays there until the last pass.

The last two passes are implemented by two separate single loops to avoid the penalty of DO loop set-up. Each group has four radix-2 butterflies in the next-to-last pass, and two in the last pass. If group DO loop is still used, then each butterfly may take 6.75 and 7.5 cycles in the next-to-last pass and the last pass, respectively. The cycles saved from the separated DO loops are 256 x 6.75 + 256 x 7.5 - 512 x 6 = 576.

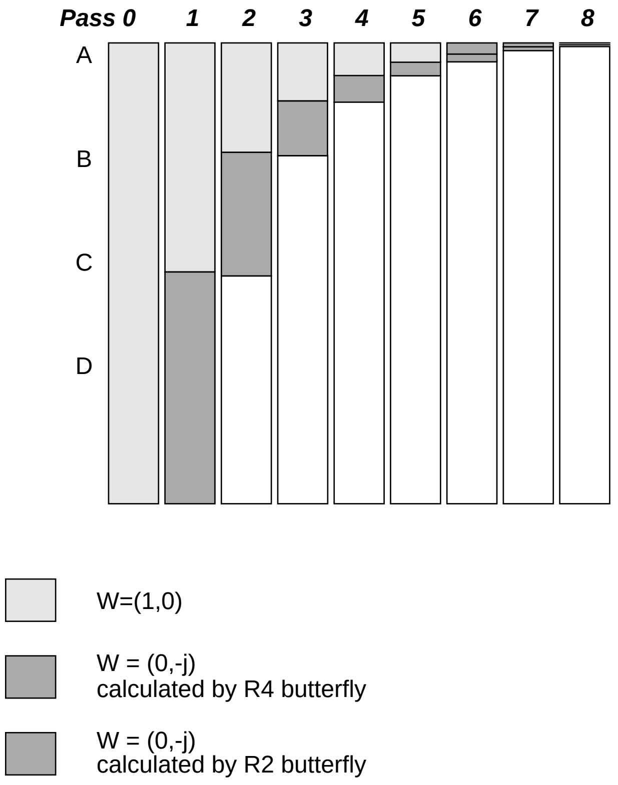


Figure 5-1 Trivial twiddle factors in a 12-point complex radix-2 DIT FFT. The butterflies in highlighted groups can be calculated without multiplications. A, B, C, and D are radix-4 butterfly pointers.

5.2.2 *Fully Optimized Complex FFT for the DSP96002*

APPENDIX A presents a fully optimized program for 1024-point complex input FFT for the DSP96002. Like the fully optimized program for the DSP56001/2, this program takes advantage of trivial twiddle factors in all of the passes as follows:

- Naturally, the first and second passes are combined into a radix-4 pass with each radix-4 butterfly requiring 8 instruction cycles. This is equal to 2 instruction cycles per radix-2 butterfly.

- All trivial butterflies in the middle passes are calculated by a separate routine.

- Each pass is written in a separate section to reduce the DO loop overhead. To reduce the program length, the special radix-4 and normal radix-2 butterflies are programmed in subroutines. Only two-nested DO loops are used for each pass.

- The last two passes are also combined into a radix-4 pass. After the combination, the number of instruction cycles per radix-2 butterfly is decreased from 5 to 4.25 instruction cycles. Because radix-4 butterflies are used in the last two passes, an extra set of 256 complex twiddle factors must be present in the external memory. These twiddle factors are generated off-line by MATHLAB software.

The fully optimized 1024-point complex FFT uses 18891 instruction cycles; while the optimized 1024-point complex FFT program (seen on the Motorola DSP bulletin board; Dr. BuB) uses 20958 instruction cycles. Optimization saves 20,958-18,891=2,067 instruction cycles which equals about 10% cycle time of the optimized code. Also note that the fully optimized code only works with fixed data length. ■

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

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

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.

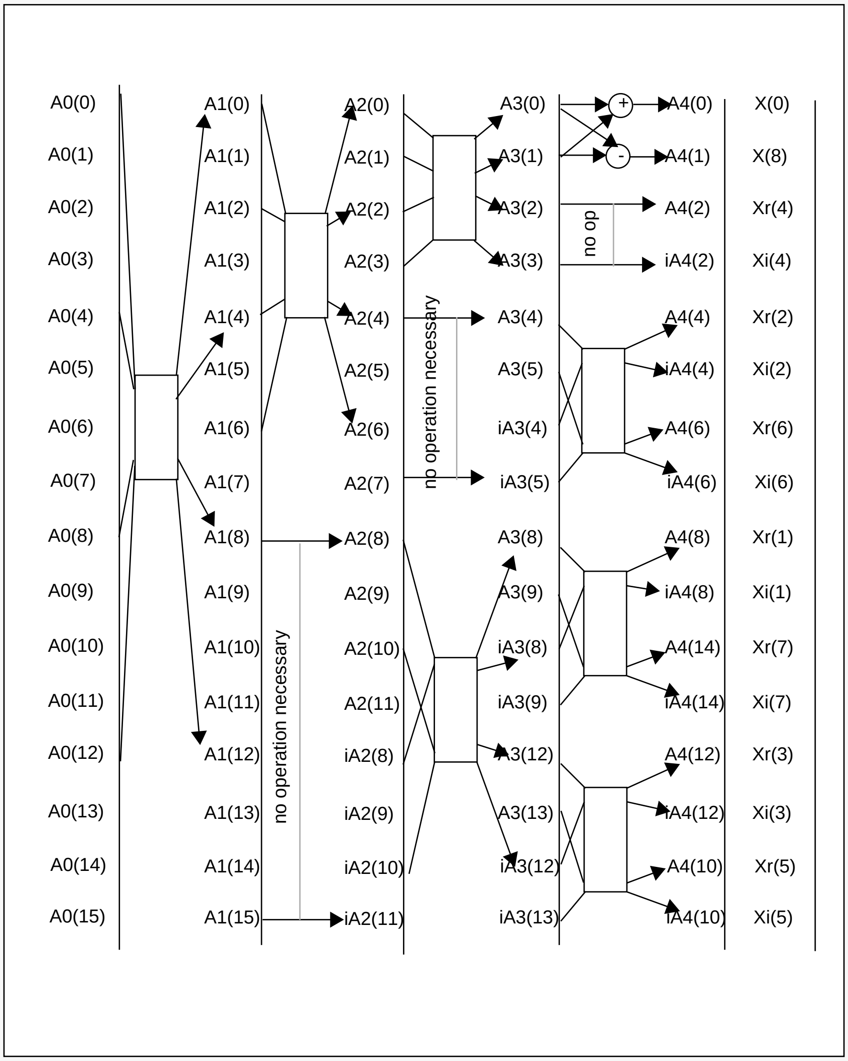


Figure 6-1 Non-redundancy calculation of the Cooly-Tukey radix-2 DIT FFT with real inputs

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.

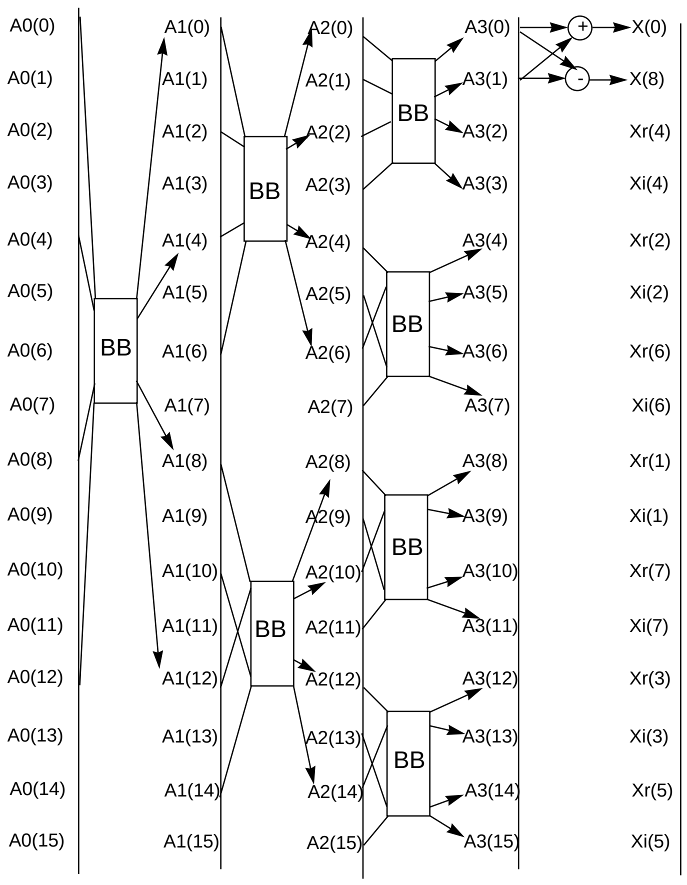


Figure 6-2 Bergland algorithm has only $\log_2(N)$-1 passes and one more addition and subtraction

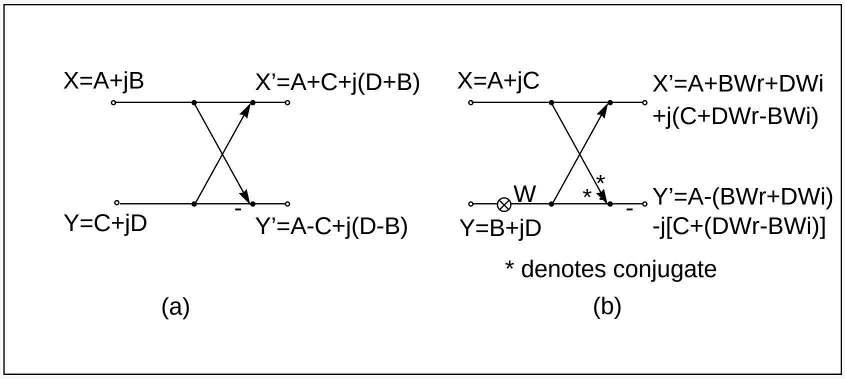


Figure 6-3 (a) Butterfly of Bergland Algorithm with $W = 1$
(b) Butterfly of Bergland Algorithm with $W \neq 1$

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,

$$\begin{aligned} Z(k) &= DFT[z(n)] = DFT[x(n) + jy(n)] \\ &= (DFT[x(n)] + jDFT[y(n)]) \\ &= (X_r(k) + jX_i(k) + j[Y_r(k) + jY_i(k)]) \\ &= ([X_r(k) - Y_i(k)] + j[X_i(k) + Y_r(k)]) \end{aligned}$$

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)

$$Z(N-k) = [X_r(k) + Y_i(k)] - j[X_i(k) - Y_r(k)]$$ 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:

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

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

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.

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

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:

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

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!

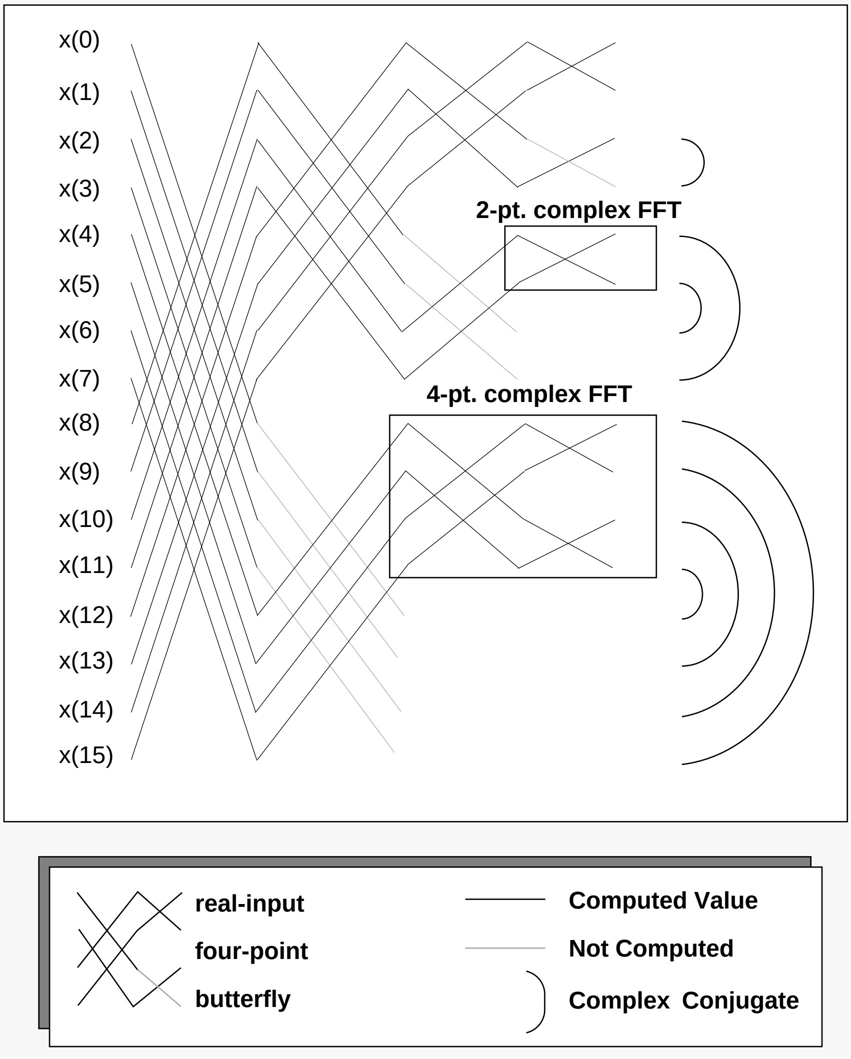


Figure 6-5 Computation of the Real-Input, DIT FFT

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

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:

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

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.

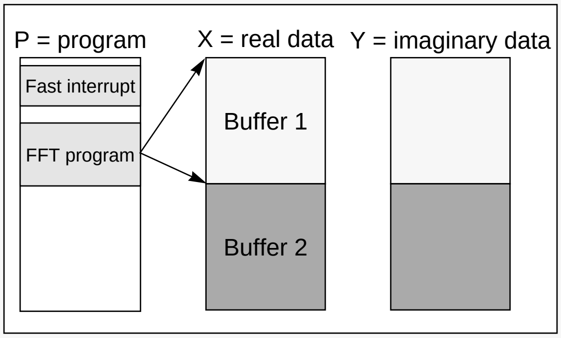


Figure 6-7 Double buffering input data so that data input can work with the FFT program concurrently

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.

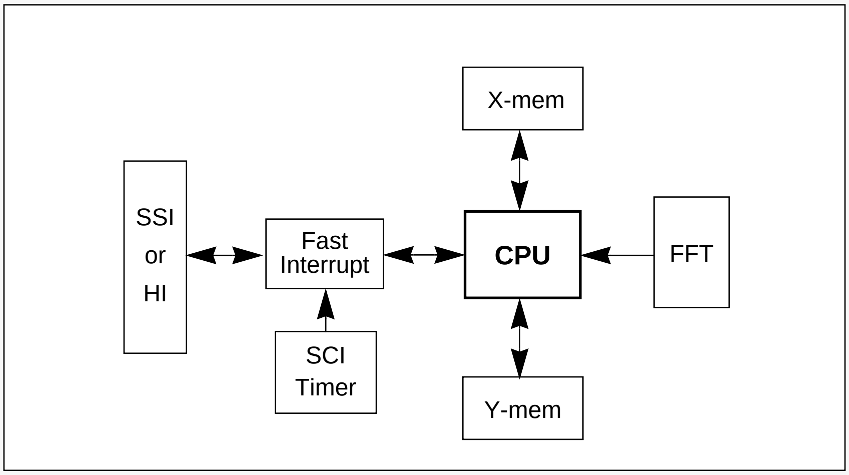


Figure 6-8 Block diagram of the double buffering technique. SSI/HI fast interrupt has higher priority than the MAIN or FFT program. The pointer of buffer is checked by SCI timer interrupt which has highest interrupt priority. The interval of the timer interrupt is set according to data length so that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

SECTION 7

Two Dimensional Fourier and Cosine Transforms

"To implement Eqn. 7-1, a two dimensional time sequence is decomposed according to its row or column."

Two dimensional Fourier transforms are widely used in image processing, image analysis, and video compression. Because the fast discrete cosine transform features high energy compaction and low implementing complexity, it is becoming more and more important in image and video compression.

7.1 Two Dimensional FFTs on the DSP96002

Two dimensional FFTs are simply an extension of one dimensional FFTs, and is shown by:

$$F(i, k) = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} x(m, n) e^{(-j(2\pi mi))/N} e^{(-j(2\pi nk))/N}$$

Eqn. 7-1

where: $i = 0,1,...N-1$

$k= 0,1,...N-1$

To implement Eqn. 7-1, a two dimensional time sequence is decomposed according to its row or column. Eqn. 7-1 can be rewritten in Eqn. 7-2.

$$F(i, k) = \sum_{m=0}^{N-1}\left(\sum_{n=0}^{N-1} x(m, n) e^{(-j(2\pi mi))/N}\right) e^{(-j(2\pi nk))/N}$$

Eqn. 7-2

The one dimensional FFT code discussed in **SECTION 4** can be used in this extension. The code included on the Motorola DSP bulletin board (2DFFT.asm) implements the two-dimensional DIT FFT by calling subroutine CFFT96.asm N times, if an N by N 2D FFT is to be performed. Also, the code demonstrates the implementation of a double buffer by the DMA controllers on the DSP96002 as discussed in **SECTION 5**.

7.2 Discrete Cosine Transform on the DSP96002

7.2.1 One Dimensional Discrete Cosine Transform (DCT)

The one dimensional cosine transform of a discrete time sequence x(n), n = 0,1,...,N-1 is defined as:

$$F(k) = \frac{2c(k)}{N}\sum_{n=0}^{N-1} x(n)\cos\left[\frac{(2n+1k\pi)}{2N}\right];\ k = 0, 1, \ldots, N-1$$

Eqn. 7-3

where:

$$\begin{aligned} c(k) &= \frac{1}{\sqrt{2}}, \quad k = 0 \\ &= 1, \quad k = 1, 2, \ldots, N-1 \\ &= 0, \quad \text{elsewhere} \end{aligned}$$

and the inverse transform is:

$$x(n) = \sum_{n=0}^{N-1} c(k)F(k)\cos\left[\frac{(2n+1k\pi)}{2N}\right];\ n = 0, 1, \ldots, N-1$$

Eqn. 7-4

A fast discrete cosine transform (FDCT) proposed by Chen and Smith [see reference 1] is adapted in this application note, and it’s flow diagram is plotted in Figure 7-1. Many optimized implementations on the FDCT have been published. The code given on the Motorola DSP bulletin board is not fully optimized; it simply demonstrates the simplicity of the DSP96002 assembly code.

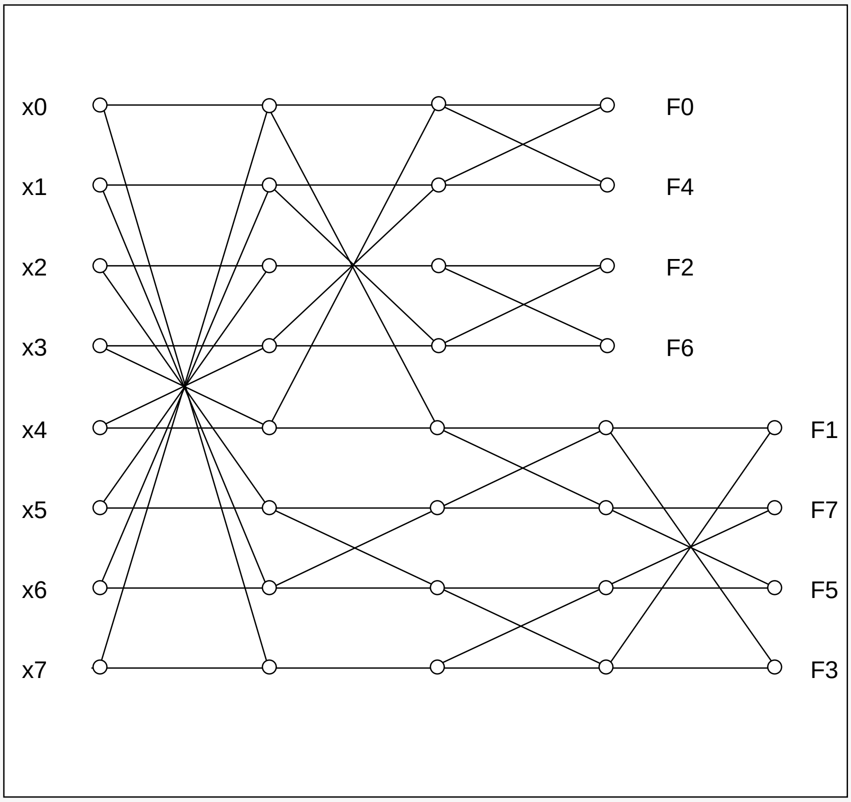


Figure 7-1 The flow diagram of an 8-point discrete cosine transform. Note that the output order of the transform is scrambled.

For $N=2^m$, $m > 2$, this algorithm requires:

$(3N/2)(\log_2 N-1)+2$ real additions and

$N\log_2 N - (3N/2)+4$ real multiplications.

7.2.2 Two Dimensional DCT

A one dimensional DCT can be easily extended to a two dimensional DCT as shown in .

$$F(j, k) = \frac{4c(j)c(k)}{N^2} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} x(m, n)\cos\left[\frac{(2n+1)k\pi}{2N}\right]$$
$$\times\cos\left[\frac{(2m+1)j\pi}{2N}\right]$$

Eqn. 7-5

Therefore, to calculate an N by N 2D DCT, repeat the N-point 1D DCT N times. An 8x8 2D DCT assembly code for the DSP96002 (DCT.asm) is presented on the Motorola DSP bulletin board . ■

SECTION 8

Competitive Analysis of FFT Performances

8.1 Most Popular Digital Signal Processors

"The total Icycles of the DSP56156 can be reduced to about 44000 Icycles and the twiddle factors can be cut to N/2 with further optimization."

Currently a variety of DSPs are available from a dozen of semiconductor vendors. This section addresses floating-point DSPs first, because the FFT is one of their most important benchmarks. The architecture of floating-point DSPs is optimized for FFTs.

Fixed-point DSPs are also discussed because they have a higher performance-to-cost ratio than the floating-point DSPs, and are used more frequently in DSP applications such as digital audio, speech processing, telecommunication, automobile control, and home electronics. Since cost is a very sensitive issue in fixed-point DSPs, some useful features such as address mode, instruction type, number of input operands in each operation, and I/O capability can be offset with a reduction in silicon area to keep the cost as low as possible.

In general, the FFT performance on fixed-point DSPs is less than floating-point DSPs if the comparison is conducted on DSPs from the same vendor. But it is not surprising that a fixed-point DSP from one manufacturer may offer a higher performance than a floating-point DSP from a different manufacturer. After comparing existing DSPs, one may decide which is an optimal architecture for FFTs regarding speed and cost, where cost refers to required memory speed, memory size, and silicon area for special hardware that aides FFT calculation. It is impractical to base the decision on selling prices because they can be strongly influenced by sales strategies of different DSP vendors.

The following sections compare DSPs from Motorola, Texas Instruments, AT&T, and Analog Devices. There are other DSPs from new players that may have their merits, but they are not included in the following discussion due to their short time on the market.

8.2 Performance of FFTs on Digital Signal Processors

Digital signal processors can be divided into two categories; floating-point DSPs and fixed-point DSPs. As is well known, the fixed-point DSPs suffer saturation problems in calculations. To solve this problem, the programmer must scale down input data either at the front or in the middle of the calculation, which results in a shrunken signal-to-noise ratio or dynamic range. The floating-point DSPs use an extra data section to hold exponent information, consequently, the dynamic range is so large that the chance of overflow is non-existent in most circumstances. Of course, one has to pay for this convenience by requiring wider data memory, a larger silicon area, and more power consumption.

8.2.1 FFTs on Floating-Point DSPs

Steps to implement various floating-point DSPs may differ depending on their conformance with the IEEE 754-1985 standard. In general, an IEEE floating-point DSP requires more computational steps to generate a normalized result than a proprietary implementation does. Although, the IEEE implementation may result in a bigger die design in achieving the same clock rate, it does, however, provide a standard interface to other microprocessors. In contrast, when proprietary formatted DSPs interface to other general purpose microprocessors, they require extra time to convert to the IEEE format. Motorola and Analog Devices are committed to the IEEE floating-point format. TI and AT&T use their own proprietary format.

Table 8-1 offers a fair comparison of complex FFTs on the different floating-point DSPs. Note that there are no constraints on the FFT algorithm. The FFT can be a Decimation in Time (DIT) or Decimation in Frequency (DIF), and can also be a radix two or radix four butterfly, as long as the algorithm can generate the best performance on a specified processor.

8.2.1.1 Complex FFT on Floating-Point DSPs

Table 8-1 1024-Point Complex FFT on Floating-Point DSPs

DSPs	96002[1]	AD21020[2]	TIC40[3]	TIC30[1]	AT&T32C[1]
Icycle (ns)	50	50	50	60	80
Algorithm	DIT	DIT	DIT	DIT	DIT
Radix	2	4	2	2	2
P Memory (word)	219	192	215	231	158
Data Memory (word)	4N	4N	4N	4N	2N
SIN/COS. table	3N/2	3N/2	N/2	N/2	N/2
Instruction length (bit)	32	48	32	32	32
SRAM for Zero Wait State (ns)	20	35	25	35	20

1. R.Meyer and K. Schwartz "FFT implementation on DSP-Chips-Theory and Practice" ICASSP, 1990.
2. Analog Devices, ADSP-21020 User's Manual.
3. Texas Instruments, TMS320C4x User's Guide.

NOTE: Icycle in Table 8-1 refers to **i**nstruction **cycle**. Minimum Icycle denotes the reciprocosity of the highest clock frequency available on the DSPs.

Table 8-1 shows that the Motorola DSP96002 performs the fastest 1024-point complex FFT. The Analog Devices' ADSP21020 performs almost as well as the DSP96002. The main factor that makes these two DSPs so fast in calculating the FFT is the special instruction "MPY||ADD||SUB". Supported by this instruction, the DSP96002 needs only four instruction cycles to perform one radix 2 butterfly, and the DSP21020 needs only fourteen instruction cycles to do one radix 4 butterfly. However, the DSP96002 has 2x512 data words on the chip and it features two on-chip DMA controllers. The on-chip memory and DMA controllers are extremely important features in implementing real-time data acquisition and control. The lack of peripherals and memory on the DSP21020 forces it into the position of competing with RISC chips. Although the DSP21020 requires lower cost SRAM for zero wait states interface, the program memory has to be 48-bits wide which negates the system cost benefits of using slow memory.

8.2.1.2 Real FFT on Floating-Point DSPs

Table 8-2 1024-Point Real Input FFT on Floating-Point DSPs

DSPs	96002[1]	TIC40[2]	TIC30[3]	AT&T32C[4]
Icycle (ns)	50	50	60	80
Total Icycles	11600	20396	31317	26300
Total Time (ms)	0.58	1.01984	1.879	2.106

1. See RFFT96T.asm on the Motorola DSP Bulletin Board (Dr. BuB).
2. Texas Instruments, TMSC4x User's Guide
3. Texas Instruments, Digital Signal Processing Applications with the TMS320 Family.
4. AT&T DSP32C User's Manual.

8.2.2 FFT on Fixed-Point DSPs

As mentioned previously, scaling must be performed on the fixed-point DSPs to prevent overflow in the intermediate stage of calculation. The following benchmarks, either complex or real FFT, assume that each input data has been divided by the number of the FFT.

8.2.2.1 Complex Input FFT

Table 8-3 1024-Point Complex FFT on Fixed-Point DSPs

DSPs	56001/2[1]	AD2100A[2]	TIC25[3]	TIC50[3]	56156[4]
Icycle (ns)	60/50	80	80	35	33
Algorithm	DIT	DIT	DIT	DIT	DIT
Radix	2	4	2	2	2
P Memory (word)	234	222			158
Data Memory (word)	4N	4N	2N	2N	4N
SIN/COS table	N/2	3N/2	5N/4	5N/4	N
Instruction length (bit)	24	24	16	16	16
Total Icycles	29949	34625	113487	82761	46373
Total Time (ms)	1.79694/ 1.49745	2.77	9.079	2.8967	1.53031

1. See CFFT56.asm on the Motorola DSP Bulletin Board (Dr. BuB).
2. R.Meyer and K. Schwartz “FFT Implementation on DSP Chips — Theory and Practice” ICASSP, 1990.
3. Texas Instruments TMS320 DSP Family Benchmarks.
4. See CFFT156.asm on the Motorola DSP Bulletin Board (Dr. BuB).

As shown in Table 8-3, the Motorola DSP56001/2 has a minimum icycle time and uses only N/2 locations for both real and imaginary twiddle factors. The total Icycles of the DSP56156 can be reduced to about 44000 Icycles and the twiddle factors can be cut to N/2 with further optimization.

8.2.2.2 Real Input FFT

Table 8-4 1024-Point Real Input FFT on Fixed-Point DSPs

DSPs	56002[1]	TIC25[2]	TIC50[2]
Icycle (ns)	50	80	35
Total Icycles	17443	56286	48055
Total Time (ms)	0.87215	4.50288	1.6819

1. See RFFT56T.asm on the Motorola DSP Bulletin Board (Dr. BuB).
2. Texas Instruments TMS320 DSP Family Benchmarks.

■

SECTION 9

Conclusion

“Motorola's family of digital signal processors, combined with Motorola's data conversion parts, provide a complete, cost-efficient solution to frequency domain problems . . .”

Frequency domain applications are becoming more important as inexpensive hardware solutions become more readily available. Motorola's Family of DSP56001/2 and DSP96002 digital signal processors provide particularly effective solutions to frequency domain problems. A highly parallel architecture, combined with an instruction set well suited for implementation of fast Fourier transforms, allow real-time computation of high-resolution FFTs up to very high sampling rates. Fast interrupts of the DSP56001/2 and the parallel DMA over a separate bus in the DSP96002 provide for data I/O with hardly any penalty in speed. Furthermore, the dual external buses on the DSP96002 allow fast calculation of FFTs of virtually unlimited size, with no performance penalty on external data access.

The large, 24-bit data representation of DSP56001/2, together with infinite-precision internal arithmetic and convergent rounding, lead to numerically superior results over 16-bit DSPs with truncation arithmetic. Special hardware provided in the DSP56001/2 allows no-overhead automatic scaling and block floating-point implementations of FFTs of virtually unlimited size, with result precision rivaling that of true floating point, for a fixed-point price.

For high-end applications, the DSP96002 provides full IEEE standard floating-point arithmetic for negligible roundoff errors. In addition to providing standard IEEE exception handling capabilities, the results obtained in the DSP96002 are portable across many applications that use the standard, such as high-level language simulations, data buses, etc. Motorola's family of digital signal processors, combined with Motorola's data conversion parts (see Reference 12), provide a complete, cost-efficient solution to frequency domain problems; from low-end small-size FFT applications, to high-end instrumentation and computer workstations for scientific computing. ■

Acknowledgments

The authors wish to thank Professor Raimund Meyer and K.Schwarz in University Erlangen in F.R Germany who provided the optimized 1024-point complex FFT program for the DSP96002; Vitus Ho in Dell Computer, Austin, Texas, who provided two dimensional FFT and DCT code for the DSP96002. Also a lot of thanks to Roman Robles in Motorola DSP Applications group for valuable comments and the C language code of Bergland’s algorithm.

APPENDIX A

Fully Optimized Complex FFT

A.1 Optimized Complex FFT for the DSP96002

```
;*****************************************************************************
;                                                                           *
;   RMAXS.ASM  : START PROGRAM FOR THE FFT MACRO RMAX.ASM.                  *
;   THIS FILE SHOWS HOW TO CONFIGURE MOTOROLAS DSP96002                     *
;   TO USE THE FAST COMPLEX FFT.                                            *
;   TWIDDLE FACTORS IN R4TAB1.ASM                                           *
;                                                                           *
;   WRITTEN BY : KARL SCHWARZ, RAIMUND MEYER       10.11.89                 *
;                                                                           *
;       LEHRSTUHL FUER NACHRICHTENTECHNIK                                   *
;       UNIVERSITAET ERLANGEN-NUERNBERG                                     *
;                                                                           *
;*****************************************************************************
    include ‘r4tab1’
    include‘rmax’

    points     equ  1024          ; FFT-length, only 1024 possible
    passes     equ  10            ; ld(points)
    data       equ  $800          ; input data, normal order
    odata      equ   $C00         ; output data, normal order
    tab4       equ  $1000         ; start of radix-4 twiddle factors
                                  ; 766 complex values)

    r4  tab1   tab4

        org    p:$100

        move   #$008A0000,x:$FFFFFFFD      ; zero wait states in BCRB
        move   #$008A0000,x:$FFFFFFFE      ; zero wait states in BCRA
        move   #$0000FFFF,x:$FFFFFFFC      ; X port A, Y and P port B in PSR
                             ;Upper three moves won’t count for the benchmark,
                             ;only for initializing the simulator or the DSP.
                             ;They show how to configure the device.
```

Figure A-1 Optimized Complex FFT for the DSP96002 (sheet 1 of 20)

```
;   A T T E N T I O N   P L E A S E !!!!!!!

;   STEP THROUGH THE FIRST THREE LINES, THEN LOAD THE SIMULATOR NEW
;   WITH RMAXS AND INPUT VECTORS, THEN GET A NEW RUN

    rmaxpoints,passes,data,odata,tab4

    nop
    nop
    nop

        end
;*****************************************************************************
;                                                                            *
;                COMPLEX, RADIX-2,4 DIT FFT  :  RMAX.ASM                     *
;            --------------------------------------------                    *
;                                                                            *
;   MACRO FOR A FAST LOOPED-CODE MIXED-RADIX DIT FFT COMPUTATION             *
;                         IN DSP96002                                        *
;                                                                            *
;    WRITTEN BY: KARL SCHWARZ, RAIMUND MEYER       10.11.89                  *
;                                                                            *
;        LEHRSTUHL FUER NACHRICHTENTECHNIK                                   *
;        UNIVERSITAET ERLANGEN-NUERNBERG                                     *
;                                                                            *
;   REVISION : THIS PROGRAM IS SPEEDED UP FROM RMIX1.ASM                     *
;                                                                            *
;   PLEASE LOOK IN THE START FILE RMAXS.ASM HOW TO CONFIGURE THE DEVICE      *
;                                                                            *
;   FOR THIS PROGRAM THE FFTLENGTH IS 1024 POINTS                            *
;   SPECIAL FEATURES : RADIX-4 BUTTERFLY IN FIRST AND LAST TWO STAGES        *
;   SIMPLE RADIX-4 BUTTERFLY IN 1. TO 6. STAGE IF NO TWIDDLES ARE USED       *
;   TABLE IN USE : ONLY R4TAB1.ASM FOR RADIX-2 AND LAST RADIX-4 BUTTERFLY    *
;   LOOK IN R4TAB1.M HOW TO BUILT A TABLE                                    *
;                                                                            *
;*****************************************************************************
;                                                                            *
;   EXAMPLE FOR THE 1024 POINT COMPLEX FFT (WITH BITREVERSAL) :              *
;                                                                            *
;   MEMORY SIZE :  PROGRAM   :  219 WORDS                                    *
;                  DATA      : 4096 WORDS                                    *
;         TWIDDLE FACTORS    : 1532 WORDS                                    *
;                                                                            *
;    CYCLES PER BUTTERFLY :                                                  *
;       1. AND 2. STAGE:     2                                               *
;       3. AND 4. STAGE:     3.5                                             *
;       5. AND 6. STAGE:     3.875                                           *
;       7.  STAGE   :        4                                               *
;       8.  STAGE   :        4.25                                            *
;       9. AND 10.STAGE :    4.25                                            *
;   AVERAGE CYCLES/BUTTERFLY:    3.55                                        *
;   TOTAL BUTTERFLYCYCLES :      18176                                       *
;   INITIALIZATION OVERHEAD:     715  =  3.8 % OF TOTAL TIME                 *
;   TOTAL NUMBER OF INSTRUCTION CYCLES  : 18891                              *
;   TOTAL TIME FOR A 1024 POINT FFT:      1.399 msAT 27 MHz                  *
```

Figure A-1 Optimized Complex FFT for the DSP96002 (sheet 2 of 20)

```
;*****************************************************************************
;                                                                           *
;   USED RADIX-2 BUTTERFLY                                                  *
;                                               +                           *
;   AR + j AI -----------------------O-------------O--------- AR' + j AI'  *
;                                      \         / +                        *
;                                        \     /                            *
;                                          \ /                              *
;                                          / \                              *
;                                        /     \                            *
;                                      /         \ +                        *
;   BR + j BI ---- ( COS - j SIN ) --O-------------O--------- BR' + j BI'  *
;                                               -                           *
;                                                                           *
;   TR = BR * COS + BI * SIN                                                *
;   TI = BR * SIN - BI * COS                                                *
;   AR'= AR + TR                                                            *
;   AI'= AI - TI                                                            *
;   BR'= AR - TR                                                            *
;   BI'= AI + TI                                                            *
;                                                                           *
;*****************************************************************************
;                                                                           *
;   USED RADIX-4 BUTTERFLY                                                  *
;                                                                           *
;   AR + j AI ----------O-----------O----------O-----O--- AR' + j AI'      *
;                         \       /             \ /                         *
;                           \   /               / \                         *
;   BR + j BI ---(W1)---O-----X-----O----------O-----O--- BR' + j BI'      *
;                         \ /   \ /                 -                       *
;                         / \   / \                                         *
;   CR + j CI ---(W2)---O-----X-----O----------O-----O--- CR' + j CI'      *
;                           /   \  -        \ /                             *
;                         /       \         / \                             *
;   DR + j DI ---(W3)---O-----------O---(-j)---O-----O--- DR' + j DI'      *
;                                  -                -                       *
;                                                                           *
; MIXING OF RADIX-2 AND RADIX-4 BUTTERFLIES IS POSSIBLE WITHOUT TROUBLE !   *
;                                                                           *
;*****************************************************************************

; ---> about 10 % faster than ICASSP 89 Paper 40.D9.7 by Kloker and Lindsley

rmaxmacropoints,passes,data,odata,tab4

;points   : FFT-length (power of 2)
;passes   : log2(points)
;data     : start address of input vector
;odata    : start address of output vector due to bitreversal
;tab4     : start address of radix-4 twiddle factors from file r4tab.asm

    pam5    equ     passes-5
    pg2     equ     points/2
    pg4     equ     points/4
    pg8     equ     points/8
```

Figure A-1 Optimized Complex FFT for the DSP96002(sheet 3 of 20)

```
        pg 16       equ      points/16
        pg 32       equ      points/32
        pg 64       equ      points/64
        pg 128      equ      points/128
        pg 4m1      equ      points/4-1
        pg 16m1     equ      points/16-1
        pg 64m1     equ      points/64-1


;******************************************************************************
; ------------ FIRST 2 STAGES AS RADIX-4 BUTTERFLY ------------------------ *
;******************************************************************************

        move    #-1,m0
        move    m0,m1
        move    m0,m2
        move    m0,m3
        move    m0,m4
        move    m0,m5
        move    m0,m6
        move    m0,m7
        move    #data,r0
        move    #(data+pg4),r1
        move    #(data+2*pg4),r2
        move    #(data+3*pg4),r3
        move    #2,n0
        move    n0,n6
        move    n0,n5
        move    n0,n7
        move    #pg4m1,n1
    jsr _sr4

;****************************************************************************
; ------------ PARTS OF 3. AND 4. STAGE AS SPECIAL RADIX-4 BUTTERFLY ------ *
;****************************************************************************

        move    #data,r0
        move    #(data+pg16),r1
        move    #(data+2*pg16),r2
        move    #(data+3*pg16),r3
        move    #pg 16m1,n1
    jsr _sr4


;****************************************************************************
; ------------ PARTS OF 5. AND 6. STAGE AS SPECIAL RADIX-4 BUTTERFLY ------ *
;****************************************************************************

        move    #data,r0
        move    #(data+pg64),r1
        move    #(data+2*pg64),r2
        move    #(data+3*pg64),r3
        move    #pg 64m1,n1
    jsr _sr4
```

Figure A-1 Optimized Complex FFT for the DSP96002(sheet 4 of 20)

```
;*****************************************************************************
; ----------- REST OF 3. STAGE AS RADIX-2 BUTTERFLY --------------------- *
;*****************************************************************************

       move   #3,n6                  ; step for twiddle addressing in r4tab

       move   #(tab4+3),r6           ; address of sin cos table
       move   #(data+pg4),r0         ; input vector
       move   #(data+pg4+pg8),r1
       move   #3,n7                  ; still 3 r2 groups to calculate
       move   #(pg 8-3),r7           ; pg8 r2 butterflies in a group
       move   #(pg 8+1),n0           ; step to next group
    jsr _nr2

;*****************************************************************************
; ----------- REST OF 4. STAGE AS RADIX-2 BUTTERFLY --------------------- *
;*****************************************************************************

       move   #(tab4+6),r6           ; address of sin cos table
       move   #(data+pg4),r0         ; input vector
       move   #(data+pg4+pg16),r1
       move   #6,n7                  ; still 6 r2 groups to calculate
       move   #(pg16-3),r7           ; pg16 r2 butterflies in a group
       move   #(pg16+1),n0           ; step to next group
    jsr _nr2

;*****************************************************************************
; ----------- REST OF 5. STAGE AS RADIX-2 BUTTERFLY --------------------- *
;*****************************************************************************

       move   #(tab4+3),r6           ; address of sin cos table
       move   #(data+pg16),r0        ; input vector
       move   #(data+pg16+pg32),r1
       move   #15,n7                 ; still 15 r2 groups to calculate
       move   #(pg32-3),r7           ; pg32 r2 butterflies in a group
       move   #(pg32+1),n0           ; step to next group
    jsr _nr2

;*****************************************************************************
; ----------- REST OF 6. STAGE AS RADIX-2 BUTTERFLY --------------------- *
;*****************************************************************************

       move   #(tab4+6),r6           ; address of sin cos table
       move   #(data+pg16),r0        ; input vector
       move   #(data+pg16+pg64),r1
       move   #30,n7                 ; still 30 r2 groups to calculate
       move   #(pg64-3),r7           ; pg64 r2 butterflies in a group
       move   #(pg64+1),n0           ; step to next group
    jsr _nr2

;*****************************************************************************
; ----------- 7. STAGE AS RADIX-2 BUTTERFLY ----------------------------- *
;*****************************************************************************

       move   #(tab4),r6             ; address of sin cos table
       move   #(data),r0             ; input vector
```

Figure A-1 Optimized Complex FFT for the DSP96002(sheet 5 of 20)

```
    move       #(data+pg128),r1
    move       #64,n7                  ; still 64 r2 groups to calculate
    move       #(pg128-3),r7           ; pg128 r2 butterflies in a group
    move       #(pg128+1),n0           ; step to next group
jsr _nr2

;*****************************************************************************
; ------------ 8. STAGE AS RADIX-2 BUTTERFLY ----------------------------- *
;*****************************************************************************

    move       #5,n0
    move       #(data+4),r1
    move       #tab4,r6
    move       #data,r0
    move       r1,r5
    mover      0,r4
    move       n0,n1
    move       n0,n4
    move       n0,n5

    move                x:(r6),d9.s        y:,d8.s
    move                x:(r1)+,d0.s       y:,d1.s
    move                x:(r0),d4.s        y:(r6),d2.s
    move                                   y:(r1),d7.s
    faddsub.s d4,d0     x:(r1)+,d6.s       y:(r6)+n6,d3.s

do        #pg8,_end3                        ; loop of groups

    fmpy       d9,d6,d0      fsub.s    d1,d2     d0.s,x:(r4)   y:(r0)+,d5.s
    fmpy       d9,d7,d1      faddsub.s d5,d2     d4.s,x:(r5)   y:(r1),d7.s
    fmpy       d8,d6,d2      fadd.s    d3,d0     x:(r0),d4.s   d2.s,y:(r5)+
    fmpy       d8,d7,d3      faddsub.s d4,d0     x:(r1)+,d6.s  d5.s,y:(r4)+

    fmpy       d9,d6,d0      fsub.s    d1,d2     d0.s,x:(r4)   y:(r0)+,d5.s
    fmpy       d9,d7,d1      faddsub.s d5,d2     d4.s,x:(r5)   y:(r1),d7.s
    fmpy       d8,d6,d2      fadd.s    d3,d0     x:(r0),d4.s   d2.s,y:(r5)+

    fmpy       d8,d7,d3      faddsub.s d4,d0     x:(r1)+n1,d6.s    d5.s,y:(r4)+
    fmpy       d9,d6,d0      fsub.s    d1,d2     d0.s,x:(r4)   y:(r0)+,d5.s
    fmpy       d9,d7,d1      faddsub.s d5,d2     d4.s,x:(r5)   y:(r1),d7.s
    fmpy       d8,d6,d2      fadd.s    d3,d0     x:(r0),d4.s   d2.s,y:(r5)+
    move                                         x:(r6)+n6,d9.s      y:,d8.s
    fmpy       d8,d7,d3      faddsub.s d4,d0     x:(r1)+,d6.s  d5.s,y:(r4)+
    fmpy       d9,d6,d0      fsub.s    d1,d2     d0.s,x:(r4)   y:(r0)+n0,d5.s
    fmpy       d9,d7,d1      faddsub.s d5,d2     d4.s,x:(r5)   y:(r1),d7.s
    fmpy       d8,d6,d2      fadd.s    d3,d0     x:(r0),d4.s   d2.s,y:(r5)+n5

    fmpy       d8,d7,d3      faddsub.s d4,d0     x:(r1)+,d6.s  d5.s,y:(r4)+n4
_end3

;*****************************************************************************
; ------------ LAST TWO STAGES AS RADIX-4 BUTTERFLY ----------------------- *
;*****************************************************************************

    move       #$0,m2
    move       m2,m3
```

Figure A-1 Optimized Complex FFT for the DSP96002 (sheet 6 of 20)

```
    movem     2,m5
    movem     2,m7
    move      #data,r0
    move      #(data+1),r4
    move      #(data+2),r1
    move      #(tab4+1),r6
    move      #2,n4
    move      #4,n0
    move      n0,n1
    move      #odata,r5
    move      #(odata+pg2),r2
    move      #(odata+pg4),r7
    move      #(odata+pg4*3),r3
    move      #pg8,n5
    move      n5,n2
    move      n5,n7
    move      n5,n3

    move               x:(r4)+n4,d3.s     y:,d5.s   ;d3=Br,d5=Bi
    move               x:(r4)+n4,d1.s     y:,d2.s   ;d1=Dr,d2=Di
    faddsub.s d1,d3    x:(r0),d7.s                  ;d3=Br+Dr,d1=Br-Dr,d7=Ar
    faddsub.s d5,d2    x:(r1),d0.sd1.s,   y:(r7)    ;d5=Bi+di,d2=Bi-Di,d0=Cr,
                                                    ;temp store Br-Dr
    faddsub.s d7,d0    d3.s,d4.s     y:(r1)+n1,d1.s     ;d0=Ar+Cr,d7=Ar-
                                                        ;Cr,d4=Br+Dr,d1=Ci
    faddsub.s d7,d5    x:(r4),d6.s   y:(r0)+n0,d3.s     ;d7=Ar-Cr-(bi+Di)
    faddsub.s d0,d4    d7.s,x:(r3)   y:(r4)+n4,d7.s

    do  #pg4m1,_er4
    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,d2d4.s,    x:(r5)d3.s,d4.s
    fmpy      d6,d8,d1      fadd.s        d5,d3d0.s,    x:(r2)+n2d1.s,y:
    fmpy.s    d7,d9,d5                    x:(r6)+,d9.s  y:,d8.s
    fsub.s    d1,d5                       x:(r4)+n4,d6.sy:,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.sd0.s,y:(r7)+n7
    fmpy      d7,d9,d2      faddsub.s     d1,d3    x:(r6)+,d9.sy:,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.d4.s,y:(r3)+n3
    fmpy      d7,d9,d1      fadd.s        d3,d0    x:(r0),d7.sd1.s,y:(r7)
    fmpy      d6,d8,d3      faddsub.s     d7,d0
    faddsub.s d7,d5
    faddsub.s d0,d4d7.s,         x:(r3)        y:(r4),d7.s
    fsub.s    d3,d1              x:(r4)+n4,d6.sy:(r0)+n0,d3.s
_er4
    faddsub.s      d3,d1d5.s,    x:(r7)
    faddsub.s      d1,d2                       y:(r7),d6.s
    move           d0.s,         x:(r2)d1.s,   y:
    faddsub.s      d3,d6d4.s,    x:(r5)d2.s,   y:
    move           d6.s,                       y:(r7)
    move           d3.s,                       y:(r3)
```

Figure A-1 Optimized Complex FFT for the DSP96002 (sheet 7 of 20)

```
;_ponrjmp_ponr              ; REMOVE THIS COMMAND AND APPEND YOUR OWN JOB
       nop
       nop
       jmp *

;***************************************************************************
*
; ------------ END OF FFT -------------------------------------------------- *
;***************************************************************************
*

; SUBROUTINES FOLLOWING

;***************************************************************************
*
; ------------ SPECIAL RADIX-4 BUTTERFLY WITH SIMPLE TWIDDLES ------------- *
;***************************************************************************
*

_sr4
    move      r0,r4
    move      r1,r5
    move      r3,r7
    move      r2,r6

    move                               y:(r5)+,d1.s
    move               x:(r0)+,d0.s    y:(r7)+,d3.s
    faddsub.s d1,d3    x:(r2),d2.s
    faddsub.s d0,d2                    y:(r4),d5.s
    faddsub.s d0,d1    x:(r1),d4.s     y:(r6)+,d7.s
    faddsub.s d5,d7    d1.s,x:(r2)+
    faddsub.s d7,d3    x:(r3),d6.s     y:(r5)-,d1.s
    faddsub.s d6,d4    d0.s,x:(r3)+    d3.s,y:(r4)+
    faddsub.s d2,d4    x:(r0)-,d0.s    d7.s,y:(r5)+n5
    faddsub.s d5,d6    d2.s,x:(r1)+    y:(r7)-,d3.s

do n1,_st2
    faddsub.s d1,d3    x:(r2),d2.s     d5.s,y:(r7)+n7
    faddsub.s d0,d2    d4.s,x:(r0)+n0  y:(r4),d5.s
    faddsub.s d0,d1    x:(r1),d4.s     y:(r6)-,d7.s
    faddsub.s d5,d7    d1.s,x:(r2)+    d6.s,y:(r6)+n6
    faddsub.s d7,d3    x:(r3),d6.s     y:(r5)-,d1.s
    faddsub.s d6,d4    d0.s,x:(r3)+    d3.s,y:(r4)+
    faddsub.s d2,d4    x:(r0)-,d0.s    d7.s,y:(r5)+n5
    faddsub.s d5,d6    d2.s,x:(r1)+    y:(r7)-,d3.s
_st2
    move               d4.s,x:(r0)     d5.s,y:(r7)
    move                               d6.s,y:-(r6)
    rts


;***************************************************************************
```

Figure A-1 Optimized Complex FFT for the DSP96002 (sheet 8 of 20)

```
; ------------ NORMAL RADIX-2 BUTTERFLY ----------------------------------- *
;************************************************************************
**

_nr2
    move      r0,r4
    move      r1,r5
    move      n0,n1
    move      n0,n4
    move      n0,n5
    move                          x:(r6)+n6,d9.s     y:,d8.s
    move                                             y:(r1),d7.s

    fmpy.s    d8,d7,d3            x:(r1)+,d6.s
    fmpy.s    d9,d6,d0
    fmpy.s    d9,d7,d1                               y:(r1),d7.s
    fmpy      d8,d6,d2            fadd.s    d3,d0   x:(r0),d4.s

    fmpy      d8,d7,d3            faddsub.s d4,d0   x:(r1)+,d6.s

    do    n7,_endgrp                                 ; loop of groups
    do    r7,_bfly                                   ; butterflyloop

    fmpy  d9,d6,d0  fsub.s        d1,d2    d0.s,x:(r4)       y:(r0)+,d5.s
    fmpy  d9,d7,d1  faddsub.s     d5,d2    d4.s,x:(r5)       y:(r1),d7.s
    fmpy  d8,d6,d2  fadd.s        d3,d0    x:(r0),d4.s       d2.s,y:(r5)+
    fmpy  d8,d7,d3  faddsub.s     d4,d0    x:(r1)+,d6.s      d5.s,y:(r4)+
_bfly
    fmpy  d9,d6,d0  fsub.s        d1,d2    d0.s,x:(r4)       y:(r0)+,d5.s
    fmpy  d9,d7,d1  faddsub.s     d5,d2    d4.s,x:(r5)       y:(r1),d7.s
    fmpy  d8,d6,d2  fadd.s        d3,d0    x:(r0),d4.s       d2.s,y:(r5)+

    fmpy  d8,d7,d3  faddsub.s     d4,d0    x:(r1)+n1,d6.s    d5.s,y:(r4)+
    fmpy  d9,d6,d0  fsub.s        d1,d2    d0.s,x:(r4)       y:(r0)+,d5.s
    fmpy  d9,d7,d1  faddsub.s     d5,d2    d4.s,x:(r5)       y:(r1),d7.s
    fmpy  d8,d6,d2  fadd.s        d3,d0    x:(r0),d4.s       d2.s,y:(r5)+

    move                                   x:(r6)+n6,d9.s    y:,d8.s

    fmpy  d8,d7,d3  faddsub.s     d4,d0    x:(r1)+,d6.s      d5.s,y:(r4)+
    fmpy  d9,d6,d0  fsub.s        d1,d2    d0.s,x:(r4)       y:(r0)+n0,d5.s
    fmpy  d9,d7,d1  faddsub.s     d5,d2    d4.s,x:(r5)       y:(r1),d7.s
    fmpy  d8,d6,d2  fadd.s        d3,d0    x:(r0),d4.s       d2.s,y:(r5)+n5
    fmpy  d8,d7,d3  faddsub.s     d4,d0    x:(r1)+,d6.s      d5.s,y:(r4)+n4
_endgrp
    rts

    endm
%   MATLAB-File to generate the radix-4 twiddle factor table for the
%   fast FFT-program RMIX1.ASM .
%   By increasing the variable fftlength you can make tables for higher
%   FFT-lengths than 1024.
%
%   Karl Schwarz, Raimund Meyer       17.10.1989
%   Lehrstuhl fuer Nachrichtentechnik
%   Universitaet   Erlangen-Nuernberg
```

Figure A-1 Optimized Complex FFT for the DSP96002 (sheet 9 of 20)

```
    fftlength=1024           ;
    fg4=fftlength/4          ;
    fg4m1=fg4-1              ;
    x=0:fg4m1                ;
    a=bitrev(x)              ;
    a=a(2:fg4)               ;
    i=1                      ;
    c(i)=1                   ;
    s(i)=0                   ;
    i=i+1                    ;
    kon=2*pi/fftlength       ;
    for k=1:fg4m1
    c(i)=cos(kon*a(k))       ;
    s(i)=sin(kon*a(k))       ;
    i=i+1                    ;
    c(i)=cos(kon*a(k)*3)     ;
    s(i)=sin(kon*a(k)*3)     ;
    i=i+1                    ;
    c(i)=cos(kon*a(k)*2)     ;
    s(i)=sin(kon*a(k)*2)     ;
    i=i+1                    ;
end
    c=c'                     ;% real part of twiddle factor (cos)
    s=s'                     ;% imaginary part of twiddle factor (sin)
end

Optimized Complex FFT for the DSP56001/2
        page    132,60
        opt     nomd,mex,loc,nocex,mu

        include 'sincosc'
        include 'bitrevtwd56'
        include 'gen56'
        include 'cfft56'

; Latest revision - 14-Oct.-92

    reset     equ      0
    start     equ      $40
    POINTS    equ      512
    IDATA     equ      $0
    COEF      equ      $800
    ODATA     equ      $1000

    sincosc   POINTS,COEF
    gen56     POINTS,IDATA

    opt       mex
    org       p:reset
    jmp       start

    org       p:start
    movep     #0,X:$FFFE                 ;0 wait states
    bitrevtwd56 POINTS,COEF
    CFFT56    IDATA,COEF,POINTS,ODATA
```

Figure A-1 Optimized Complex FFT for the DSP96002(sheet 10 of 20)

```
        nop
        nop
        jmp *
        end
;
; Sine-Cosine Table Generator for rfft56.asm and cfft56.asm
;
; Last Update 10/28/92
;
sincosc  macro   points,coef
sincosc  ident   1,2
;
;       sincosc  -      macro to generate sine and cosine coefficient
;                       lookup tables for Decimation in Time complex FFT
;                       twiddle factors. Only points/4 coefficients
;                       are generted. For real FFT another points/4
;                       coefficients with higher freq. are created.
;
;       points  -       number of points (2 - 32768, power of 2)
;       coef    -       base address of sine/cosine table
;                       positive cosine value in X memory
;                       positive sine value in Y memory
;
;    8/12/92

    pi    equ      3.141592654
;freq     equ      2.0*pi/@cvf(points*2)

;         org      x:coef-points/2
;count    set      0
;         dup      points/2
;         dc       @cos(@cvf(count)*freq)
;count    set      count+1
;         endm
;
;         org      y:coef-points/2
;count    set      0
;         dup      points/2
;         dc       -@sin(@cvf(count)*freq)
;count    set      count+1
;         endm

freq1    equ      2.0*pi/@cvf(points)

;  int i,j=1,k,tmp=0;
;  k=1<<(length-1);
;  for(i=0;i<length;i++){
;     if (integer&j) tmp=tmp|k;
;     j=j<<1;
;     k=k>>1;
        org    x:coef
count   set    0
        dup    points/4
        dc     @cos(@cvf(count)*freq1)
count   set    count+1
        endm
```

Figure A-1 Optimized Complex FFT for the DSP96002 (sheet 11 of 20)

```
       org    y:coef
count  set    0
       dup    points/4
       dc     @sin(@cvf(count)*freq1)
count  set    count+1
       endm

       endm                          ;end of sincosr  macro


bitrevtwd56   macro    POINTS,COEF
bitrevtwd56   ident    1,2
;
;      bitrevtwd -     macro to sort sine and cosine coefficient
;                      lookup tables in bit reverse order for 56156
;
;      POINTS  -       number of points (2 - 32768, power of 2)
;      COEF    -       base address of sine/cosine table
;                      negative cosine (Wr) and negative sine (Wi) in X memory
;
; Wei Chen
; July-28, 1992
;


       move   #COEF,r1                ;twiddle factor start address
       move   #0,m0                   ;bit reverse address
       move   #POINTS/8,n0            ;sincosr use N/4 points data,
                                      ;offset for bit rev. is N/8
       move   #POINTS/4-1,n2
       move   r1,r0                   ;r1 ptr to normal order data
       move   (r1)+                   ;no swap on 1st data
       move   (r0)+n0                 ;r0 ptr to bitrev
       do     n2,_end_bit             ;does N/4-1 points swap
       move   r1,x0
       move   r0,b
       cmp    x0,b
       jgt    _swap
       move   (r1)+                   ;no swap but update points
       move   (r0)+n0
       jmp    _nothing
_swap
       move   r1,r5
       move   r0,r4
       move   x:(r1),x0     y:(r5),y0
       move   x:(r0),a      y:(r4),b
       move   x0,x:(r0)+n0  y0,y:(r4)
       move   a,x:(r1)+     b,y:(r5)
_nothing
       nop
_end_bit
       endm                          ;end of bitrevtwd macro
```

Figure A-1 Optimized Complex FFT for the DSP96002 (sheet 12 of 20)

```
; Input signal for FFT rfft56.asm and cfft56.asm
;
; Last Update 10/28/92
;
gen56    macro   POINTS,IDATA
;
;   gen56 -   macro to generate input signal for FFT test on 56001
;             2000 Hz sinewave with scaling factor POINTS in X and Y memory
;
;       POINTS -    number of points (2 - 32768, power of 2)
;       IDATA  -    base address of signal
;
srate   set    44100     ;Hz
ffreq   set    2000      ;Hz
ppi     equ    3.141592654
freq2   equ    2.0*ppi*ffreq/@cvf(srate)

        org    x:IDATA
count   set    0
        dup    POINTS
        dc     @sin(@cvf(count)*freq2)/POINTS
count   set    count+1
        endm

        org    y:IDATA
count   set    0
        dup    POINTS
        dc     @sin(@cvf(count)*freq2)/POINTS
count   set    count+1
        endm

        endm     ;end of gen56  macro




;
; 512-Point, 28174 clock cycles Non-In-Place FFT.
;
; Sept. 11 92   Version 1.0
;
CFFT56  macro   IDATA,COEF,POINTS,ODATA
CFFT56  ident   1,0
;
; 512 Point Complex Fast Fourier Transform Routine
; using the Radix 2, Decimation in Time, Cooley-Tukey FFT algorithm.
;
; This routine performs a 512 point complex FFT by taking advantages of
;   1).   internal memory access by starting first half data at location 0,
;         avoid cycle stretching;
;   2).   using N/4 complex twiddle factors based on the fact that two
;         consectivetwiddle factors in DIT FFT has a difference -j
;   3).   trivial twiddle factors (1,0) and (0,-1)  are utilized.
;
```

Figure A-1 Optimized Complex FFT for the DSP96002(sheet 13 of 20)

```
;
;    Complex input and output data
;        Real data in X memory
;        Imaginary data in Y memory
;    Normally ordered input data
;    Bit reversed output data for 1024 real input FFT
;       Coefficient lookup table
;        +Cosine values in X memory
;        -Sine values in Y memory
;
;
; Address pointers are organized as follows:
;
;   r0 = ar,ai input pointer     n0 = group offset       m0 = modulo (points)
;   r1 = br,bi input pointer     n1 = group offset       m1 = modulo (points)
;   r2 = ext. data base address  n2 = groups per pass    m2 = 256 pt fft counter
;   r3 = coef. offset each pass  n3 = coefficient base addr. m3 = linear
;   r4 = ar',ai' output pointer  n4 = group offset       m4 = modulo (points)
;   r5 = br',bi' output pointer  n5 = group offset       m5 = modulo (points)
;   r6 = wr,wi input pointer     n6 = coef. offset       m6 = bit reversed
;   r7 = not used (*)            n7 = not used (*)       m7 = not used (*)
;
;       * - r7, n7 and m7 are typically reserved for a user stack pointer.
;
; 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
;       r2      n2      m2
;       r3      n3      m3
;       r4      n4      m4
;       r5      n5      m5
;       r6      n6      m6
;
; Alters Program Control Registers
;       pc      sr
;
; Uses 8 locations on System Stack
;
;



;-----------------------------------------------------------------------------;
; Initialize pointers to r0->Ar,r1->Cr,r4->Bi,r5->Di, and r3->temp location  ;
; r0,r1,r4, and r5 are modular addressing with modulo N/2                     ;
;-----------------------------------------------------------------------------;

        move    #IDATA,r0       ;r0 -> Ar
        move    r0,n3
        move    #ODATA,r3       ;r3 always has ODATA
        move    #COEF+1,n6      ;n6 always has COEF,(0,1) is not used
        move    #POINTS/4,n0    ;offset and butterflies per group
```

Figure A-1 Optimized Complex FFT for the DSP96002 (sheet 14 of 20)

```
        move   #POINTS/2-1,m0         ;modulo addressing
        move   r0,r6                  ;r6=0 flag reg. for trivial groups

        do     #3,_end_trivial        ;do three R4 passes
        move   n0,n1                  ;pointer offset
        move   n0,n4                  ;pointer offset
        move   n0,n5                  ;
        lea    (r0)+n0,r4             ;r4 -> Bi
        move   m0,m5
        lea    (r4)+n4,r1             ;r1 -> Cr
        move   m0,m1                  ;
        move   m0,m4
        lea    (r1)+n1,r5             ;r5 -> Di
;----------------------------------------------------------------------------;
;   First two passes are combined into a R4 pass without multiplication        ;
;   because Wr=1,Wi=0 in first R2 pass and Wr=0, Wi=-1 in 2nd R2 pass          ;
;                                                                              ;
; Ar'=Ar+Cr+(Br+Dr)  Br'=Ar+Cr-(Br+Dr)  Cr'=(Ar-Cr)+(Bi-Di)   Dr'=(Ar-Cr)-(Bi-Di);
; Ai'=Ai+Ci+(Bi+Di)  Bi'=Ai+Ci-(Bi+Di)  Ci'=(Ai-Ci)+(Dr-Br)   Di'=(Ai-Ci)-(Dr-Br);
;                                                                              ;
; This two passes fully ultilize internal memory by storing input data at location 0;
; For 1024-point complex FFTs, only 256-point in internal, rest of them in      ;
; external, 17+2 instructions are needed for one butterfly because first and next to;
; the last instruction in the loop takes two Icycles. Other parallel move seems to;
; take two  cycles, but one of the two moves is internal, only one cycle is needed.  ;
; 4.75 Icycles per R2 butterfly in the fisrt two passes.                        ;
;                                                                              ;
; For 512-point complex FFT, 17+1 instructoins are used because first instruction in
;the loop takes only one Icycle. 4.5 Icycles per R2 butterfly.                 ;
;                                                                              ;
; For 256 or less point complex FFT, 17 Icycles are needed. 4.25 Icycles/bfly.
;
;----------------------------------------------------------------------------;
    move      x:(r0)+n0,a               ; a= Ar r0 -> Br
    move      x:(r1)+n1,b               ; b= Cr,r1 -> Dr

    do        n0,_twopass
    add  a,b  x:(r0)+n0,x1  y:(r5)+n5,y1  ;b=Ar+Cr,x1=Br,y1=Di,r0->Ar,r5->Ci
    subl b,a  b,x:(r0)      y:(r4),b      ;a=Ar-Cr,save Ar+Cr temp in Ar,b=Bi
    add  y1,b a,x0          y:(r4)+n4,a   ;b=Bi+Di,x0=Ar-Cr,a=Bi again,
                                          ;save Ar-Cr in Dr,r4->Ai
    sub  y1,a b,x:(r3)x0,b           ;a=Bi-Di,store Bi+Di temp in x:ODATA,b=Ar-Cr
    sub  a,b  x:(r1),x0              ;b=Ar-Cr-(Bi-Di)=Dr',x0=Dr,r0 -> Ar
    addl b,a  b,x:(r1)+n1   x0,b     ;a=Ar-Cr+(Bi-Di)=Cr',save Dr',b=Dr,r1->Cr
    sub  x1,b a,x:(r1)+     x0,a     ;b=Dr-Br,save Cr',a=Dr,r1->nCr
    add  x1,a x:(r0)+n0,b   b,y1     ;a=Dr+Br,b=Ar+Cr, y1=Dr-Br,r0->Br
    sub  a,b                y:(r5),y0;a=Ar+Cr-(Dr+Br)=Br',y0=Ci
    addl b,a  b,x:(r0)+n0   y:(r4),b ;a=Ar+Cr+(Dr+Br)=Ar',save Br',r0->Ar,b=Ai
    sub  y0,b a,x:(r0)+     y:(r4),a ;b=Ai-Ci,a=Ai again, save Ar',r0->nAr
    add  y0,a x:(r3),b      b,y0     ;a=Ai+Ci,y0=Ai-Ci,b=Bi+Di, r5->Ci
    add  a,b                         ;b=Ai+Ci+(Bi+Di)=Ai'
    subl b,a  y0,b     b,y:(r4)+n4   ;a=Ai+Ci-(Bi+Di)=Bi',b=Ai-Ci,save Ai'
    add  y1,b y0,a     a,y:(r4)+ ;b=Ai-Ci+(Dr-Br)=Ci',a=Ai-Ci,save Bi',r4->nBi
    sub  y1,a x:(r1)+n1,b   b,y:(r5)+n5;a=Ai-Ci-(Dr-Br)=Di',b=nCr,  save Ci',
    move  x:(r0)+n0,a       a,y:(r5)+ ;a=nAr,  save Di',r5->nDi
_twopass
;----------------------------------------------------------------------------;
```

Figure A-1 Optimized Complex FFT for the DSP96002 (sheet 15 of 20)

```
; Do rest of trivial group by 5 Icyc butterfly
;-------------------------------------------------------------------------;
    move      n5,a                       ;n5 contains ptr to Ar already
    asr       a      n5,r1               ;r1->Ar
    move      a,n1                       ;get offset
    move      r1,r5                      ;r5->Ai
    lea       (r1)+n1,r4                 ;r4->Bi
    move      #2,n4                      ;for pointer
    move      r4,r0                      ;r0->Br
    move      x:(r1),a  y:(r4)+,b        ;a=Ar,b=Bi,r4->nBi
    do        n1,_no_more                ;w=(0,-1), R2 butterfly
    add       a,b  x:(r0),x0 y:(r4)-,y0  ;b=Ar+Bi=Ar’,x0=Br,y0=nBi
    subl      b,a  b,x:(r1)+ y:(r5),b    ;a=Ar-Bi=Br’,save Ar’,b=Ai
    add       x0,b a,x:(r0)+ y:(r5),a    ;b=Ai+Br=Bi’,save Br’,a=Ai again
    subl      b,a  y0,b      b,y:(r4)+n4 ;a=Ai-Br=Ai’,save Bi’,b=nBi
    move      x:(r1),a  a,y:(r5)+        ;a=nAr,save Ai’
_no_more
    move      n0,a
    asr       a    n3,r0                 ;r0->IDATA
    asr       a    a,r2
    move      a,n0                       ;(points in a group)/4 after a radix
4 pass
    move      x:(r2)-,b      ;dec r2
    move      r2,m0
_end_trivial
    move      r0,r4                           ;output pointer
    move      n1,r1                           ;r1->Br
    move      r1,r5                           ;r5->Bi
    move      x:(r0),a                        ;a=Ar
    move      x:(r1),b                        ;b=Br
    do        n1,_extra                       ;w=(1,0)
    add       a,b                 y:(r5),y0   ;b=Ar+Br=Ar’, y0=Bi
    subl      b,a  b,x:(r0)+      y:(r4),b    ;a=Ar-Br=Br’,save Ar’,b=Ai
    add       y0,b a,x:(r1)+      y:(r4),a    ;b=Ai+Bi=Ai’,save Br’,a=Ai
    sub       y0,a x:(r1),b       b,y:(r4)+   ;a=Ai-Bi=Bi’,save Ai’,b=nBr
    move      x:(r0),a            a,y:(r5)+   ;a=nAr,save Bi’
_extra

;-------------------------------------------------------------------------;
;   Remaining passes are broken down to POINTS/256 sets,                  ;
;   each set has 256-point R2 FFT                                         ;
;   and runs on internal data and externalcoefficients.                   ;
;In each pass, first two groups takes advantages of trivial twiddle factors and;
;no multiplication is carried out. Remaining groups use complex twiddle factors.;
;
;                                                                         ;
; Radix 2, Decimation In Time Cooley-Tukey FFT algorithm                  ;
;              __________                                                 ;
;             |          |       Ar’= Ar + Wr*Br - Wi*Bi                  ;
; Ar,Ai ----> |  Radix-2 |---->  Ai’= Ai + Wi*Br + Wr*Bi                  ;
; Br,Bi ----> | Butterfly|---->  Br’= Ar - Wr*Br + Wi*Bi = 2*Ar - Ar’     ;
;             |__________|       Bi’= Ai - Wi*Br - Wr*Bi = 2*Ai - Ai’     ;
;                  ^                                                      ;
;                  |                                                      ;
;               W= Wr-jWi                                                 ;
;                                                                         ;
```

Figure A-1 Optimized Complex FFT for the DSP96002 (sheet 16 of 20)

```
;   r0->A,r1->B,r4->A',r5->B',r6->TF,n0=offset for B pointer,
;   n2=numberof bflies in a group ;
;   n3=number of groups in a pass, m3=number of pass. r2=n3 or n3+1       ;
;-----------------------------------------------------------------------------;
    move      m2,m0             ;linear address
    move      m2,m1
    move      m2,m4
    move      m2,m5
    move      #POINTS/4,r0      ;start location of a pass
    move      #4,m3             ;4 passes in first 256-point
    move      #POINTS/16,n0     ;offset to point to Br and Bi
    move      n0,n1
    move      n0,n4
    move      n0,n5
    move      n6,r6             ;r6->COEF
    move      n0,n2             ;number of bflies in the first pass=R2 bfies/4
    lea       (r0)+n0,r1        ;r1->Br
    move      r0,r4             ;r4->Ai
    lea       (r1)-,r5          ;r5->Bi-1 for pointer reason
    jsr       _body

;-----------------------------------------------------------------------------;
; The second 256-point FFT has no any trivial twiddle factors,              ;
; three nested loops do it                                                  ;
;-----------------------------------------------------------------------------;
    move      #256,r0           ;start location of first pass in 2nd 256
    move      #5,m3             ;5 passes in second 256-point
    move      #POINTS/8,n0      ;offset to point to Br and Bi
    move      #COEF+1,r6        ;twiddle factor pointer
    move      n0,n1
    move      n0,n4
    move      #IDATA,r4         ;r4->A' =IDATA
    move      n0,n5
    lea       (r4)+n4,r5        ;r5->B'
    lea       (r0)+n0,r1        ;r1->B
    move      x:(r5)-,a         ;r5->B'-1 for pointer reason
    jsr       _body
    jmp       _end_FFT

;-----------------------------------------------------------------------------;
; All subroutines
;-----------------------------------------------------------------------------;
_body
    move      #1,n3        ;number of groups in a pass
    move      n3,r2        ;copy of n3
    jset      #0,m3,_set_grp;first 256-point has number of group 1,3,7,15,..
    move      #2,r2
_set_grp
    do        m3,_inner_loop
    jsr       _inner_pass
    move      n0,a
    asr       a         #IDATA,r0 ;r0=IDATA
    move      a,n0              ;n0=offset of B
    move      r2,a
    asl       a         n0,n1
    move  a,r2                  ;r2=r2*2
```

Figure A-1 Optimized Complex FFT for the DSP96002 (sheet 17 of 20)

```
    asr   b        r2,n3     ;n3=number of groups in second 256
    jset #0,m3,_inner_set    ;set up start address,for 2nd 256-point r0 is already ok
    lea   (r2)-,n3           ;n3=number of groups in first 256
    move  n4,a
    asl   a        n6,r6     ;for 1st 256, TF always starts at first location
    move  a,r0               ;r0=start location of first 256-point
_inner_set
    move  n0,n4
    move  n0,n5
    move  n0,r4
    lea   (r0)+n0,r1         ;r1->B
    move  r0,r4              ;r4->A'
    lea   (r1)-,r5           ;r5->B'
_inner_loop

;-----------------------------------------------------------------------------;
;  End inner loop
;-----------------------------------------------------------------------------;
    move  #IDATA,r0
    move  #32,n2                      ;n2=number of groups in the next to last
                                      ;pass for 1st 256
    jset  #0,m3,_no_set               ;set up start address of TF,for 2nd
                                      ;256-point r6 is already ok
    move  #COEF,n6                    ;now n6 -> COEF
    move  n6,r6                       ;r6=COEF
    lea   (r0)+n0,r1                  ;r1->Br
    move  r0,r4                       ;r4->Ai
_no_set
    move  #-1,r5                      ;r5->Bi
    move  #3,n0
    move  n0,n1
    move  n0,n4
    move  n0,n5
    jsr  _next_last                   ;do the pass next to last
;-----------------------------------------------------------------------------;
;  End _next_last pass
;------------------------------------------------------------------------------
;
    move      #IDATA,r0               ;r0->IDATA
    move      r3,r4                   ;r4->A',output ptr -> external memory
    jclr      #0,m3,_add_offset       ;set up output address for 2nd 256-point
    move      #256,n3
    move      r6,n6                   ;start address of TF for 2nd 256
    lea       (r3)+n3,r4
_add_offset
    lea       (r0)+,r1                ;r1->B
    lea       (r4)-,r5                ;r5->B'
    move      #64,n2                  ;number of blies in the last pass
    move      #2,n0
    move      n0,n1
    move      n0,n4
    move      n0,n5
    move      n6,r6                   ;r6=COEF
    jsr       _last
    rts
```

Figure A-1 Optimized Complex FFT for the DSP96002 (sheet 18 of 20)

```
_inner_pass
    do   n3,_end_grp                               ;do groups in a pass
    move      x:(r5),a                             ;for pointer reason,a=something
    move      x:(r6),x0     y:(r0),b               ;x0=Wr,b=Ai
    move      x:(r1),x1     y:(r6)+,y0             ;x1=Br,y0=Wi
    do   n0,_end_bfy1                              ;Radix 2 DIT butterfly kernel
                                                   ;with y0=Wi,x0=Wr
    mac  -x1,y0,b y:(r1)+,y1                       ;b=Ai-BrWi,y1=Bi, r1->nBi
    macr x0,y1,b  a,x:(r5)+     y:(r0),a           ;b=Ai-BrWi+BiWr=Ai',
                                                   ;save prev.Br',a=Ai
    subl b,a      x:(r0),b      b,y:(r4)           ;a=2Ai-Ai'=Bi',b=Ar,save Ai'
    mac  x1,x0,b  x:(r0)+,a     a,y:(r5)           ;b=Ar+BrWr,a=Ar,save Bi',r0->nAi
    macr y1,y0,b  x:(r1),x1                        ;b=Ar+BrWr+BiWi=Ar',x1=nBr
    subl b,a      b,x:(r4)+     y:(r0),b           ;a=2Ar-Ar'=Br',
                                                   ;save Ar',b=nAi,r4->nAr
_end_bfy1
    move          a,x:(r5)+n5   y:(r1)+n1,b        ;save preve. Br' inc r5 and r1
    move          x:(r4)+n4,a   y:(r0)+n0,b        ;inc r0,r4
    move          x:(r1),x1                        ;x1=nGBr
    move          x:(r5),a      y:(r0),b           ;for pointer reason,
                                                   ;a=something,b=nGAr
    do   n0,_end_bfy2                                       ;W=-j*W
    mac  -x1,x0,b y:(r1)+,y1                       ;b=Ai-BrWr,y1=Bi,r1->nBi
    macr -y0,y1,b a,x:(r5)+ y:(r0),a               ;b=Ai-BrWr-BiWi=Ai',
                                                   ;save prev. Br',a=Ai
    subl b,a      x:(r0),b      b,y:(r4)           ;a=2Ai-Ai'=Bi',b=Ar,save Ai'
    mac  -x1,y0,b x:(r0)+,a     a,y:(r5)           ;b=Ar-BrWi,a=Ar,save Bi',r0->nAi
    macr y1,x0,b  x:(r1),x1                        ;b=Ar-BrWi+BiWr=Ar',x1=nBr
    subl b,a      b,x:(r4)+     y:(r0),b           ;a=2Ar-Ar'=Br',
                                                   ;save Ar',b=nAi,r4->nAr
_end_bfy2
    move          a,x:(r5)+n5   y:(r1)+n1,b        ;save preve. Br' inc r5 and r1
    move          x:(r4)+n4,a   y:(r0)+n0,b        ;inc r0,r4
_end_grp
    rts

_next_last
    move          x:(r5),a      y:(r0),b           ;a=something,b=Ai
    move          x:(r1),x1     y:(r6),y0          ;x1=Br,y0=Wi
    do   n2,_n_last                                ;do the pass next to last,
                                                   ;internal to internal
    mac  -x1,y0,b x:(r6)+,x0    y:(r1)+,y          ;b=Ai-BrWi,x0=Wr,y1=Bi, r1->nBi
    macr x0,y1,b  a,x:(r5)+n5   y:(r0),a           ;b=Ai-BrWi+BiWr=Ai',
                                                   ;save prev. Br',a=Ai
    subl b,a      x:(r0),b      b,y:(r4)           ;a=2Ai-Ai'=Bi',b=Ar,save Ai'
    mac  x1,x0,b  x:(r0)+,a     a,y:(r5)           ;b=Ar+BrWr,a=Ar,save Bi',r0->nAi
    macr y1,y0,b  x:(r1),x1                        ;b=Ar+BrWr+BiWi=Ar',x1=nBr
    subl b,a      b,x:(r4)+     y:(r0),b           ;a=2Ar-Ar'=Br',
                                                   ;save Ar',b=nAi,r4->nAr

    mac  -x1,y0,b               y:(r1)+n1,y1       ;b=Ai-BrWi,y1=Bi, r1->nGBi
    macr x0,y1,b  a,x:(r5)+     y:(r0),a           ;b=Ai-BrWi+BiWr=Ai',
                                                   ;save prev. Br',a=Ai
    subl b,a      x:(r0),b      b,y:(r4)           ;a=2Ai-Ai'=Bi',b=Ar,save Ai'
    mac  x1,x0,b  x:(r0)+n0,a   a,y:(r5)           ;b=Ar+BrWr,a=Ar,
                                                   ;save Bi',r0->nGAi
```

Figure A-1 Optimized Complex FFT for the DSP96002 (sheet 19 of 20)

```
    macr  y1,y0,b   x:(r1),x1                 ;b=Ar+BrWr+BiWi=Ar',x1=nGBr
    subl  b,a       b,x:(r4)+n4   y:(r0),b  ;a=2Ar-Ar'=Br',save Ar',
                                            ;b=nGAi,r4->nGAr

    mac   -x1,x0,b           y:(r1)+,y1     ;b=Ai-BrWr,y1=Bi,r1->nGBi
    macr  -y0,y1,b  a,x:(r5)+n5y:(r0),a     ;b=Ai-BrWr-BiWi=Ai',
                                            ;save prev. Br',a=Ai,r5->nGBi
    subl  b,a       x:(r0),b      b,y:(r4)  ;a=2Ai-Ai'=Bi',b=Ar,save Ai'
    mac   -x1,y0,b  x:(r0)+,      a,y:(r5)  ;b=Ar-BrWi,a=Ar,save Bi',r0->nGAi
    macr  y1,x0,b   x:(r1),x1               ;b=Ar-BrWi+BiWr=Ar',x1=nBr
    subl  b,a       b,x:(r4)+     y:(r0),b      ;a=2Ar-Ar'=Br',
                                                ;save Ar',b=nAi,r4->nGAr

    mac   -x1,x0,b                y:(r1)+n1,y1  ;b=Ai-BrWr,y1=Bi,r1->nGBi
    macr  -y0,y1,b  a,x:(r5)+     y:(r0),a      ;b=Ai-BrWi-BiWr=Ai',
                                                ;save prev. Br',a=Ai,r5->Bi
    subl  b,a       x:(r0),b      b,y:(r4)      ;a=2Ai-Ai'=Bi',b=Ar,save Ai'
    mac   -x1,y0,b  x:(r0)+n0,a   a,y:(r5)      ;b=Ar-BrWi,a=Ar,
                                                ;save Bi',r0->nGAi
    macr  y1,x0,b   x:(r1),x1     y:(r6),y0     ;b=Ar-BrWi+BiWr=Ar',
                                                ;x1=nBr,y0=nWi
    subl  b,a       b,x:(r4)+n4   y:(r0),b      ;a=2Ar-Ar'=Br',save Ar',
                                                ;b=nAi,r4->nGAr

_n_last
    move            a,x:(r5)
    rts

_last
    move  x:(r5),a  y:(r0),b                ;a=something,b=Ai
    move  x:(r1),x1 y:(r6),y0               ;x1=Br,y0=Wi
    do    n2,_end_last                      ;do last pass, internal to external
    mac   -x1,y0,b  x:(r6)+,x0    y:(r1)+n1,y1  ;b=Ai-BrWi,x0=Wr,y1=Bi, r1->nGBi
    macr  x0,y1,b   a,x:(r5)+n5   y:(r0),a  ;b=Ai-BrWi+BiWr=Ai',
                                            ;save prev. Br',a=Ai
    subl  b,a       x:(r0),b      b,y:(r4)  ;a=2Ai-Ai'=Bi',b=Ar,save Ai'
    mac   x1,x0,b   x:(r0)+n0,a   a,y:(r5)  ;b=Ar+BrWr,a=Ar,save Bi',r0->nGAi
    macr  y1,y0,b   x:(r1),x1               ;b=Ar+BrWr+BiWi=Ar',x1=nGBr
    subl  b,a       b,x:(r4)+n4   y:(r0),b  ;a=2Ar-Ar'=Br',
                                            ;save Ar',b=nGAi,r4->nGAr

    mac   -x1,x0,b                y:(r1)+n1,y1  ;b=Ai-BrWr,y1=Bi,r1->nGBi
    macr  -y0,y1,b  a,x:(r5)+n5   y:(r0),a  ;b=Ai-BrWr-BiWi=Ai',
                                            ;save prev. Br',a=Ai,r5->Bi
    subl  b,a       x:(r0),b      b,y:(r4)  ;a=2Ai-Ai'=Bi',b=Ar,save Ai'
    mac   -x1,y0,b  x:(r0)+n0,a   a,y:(r5)  ;b=Ar-BrWi,a=Ar,save Bi',r0->nGAi
    macr  y1,x0,b   x:(r1),x1     y:(r6),y0 ;b=Ar-BrWi+BiWr=Ar',
                                            ;x1=nBr,y0=nWi
    subl  b,a       b,x:(r4)+n4   y:(r0),b  ;a=2Ar-Ar'=Br',
                                            ;save Ar',b=nAi,r4->nGAr
_end_last
    move            a,x:(r5)
    rts
_end_FFT
    endm
```

Figure A-1 Optimized Complex FFT for the DSP96002 (sheet 20 of 20)

APPENDIX B

Real-Valued Input FFT

B.1 Faster real FFT for the DSP96002

```
    page 132,60,1,1
    opt mex
;*******************************************
;Motorola Austin DSP Operation  20 August 1992
;*******************************************
;Test program for DSP96002 rfft96.asm
;**************************************************************************
;       1024 real-valued inputs
;   Maximum sample rate:  0.58 ms at 40.0 MHz
;   Memory Size:   Prog:141 + 32 words ;
;                  Data:2*1024 words(idata+odata) + 256 words (twiddle factor)
;   Number of clock cycles: 23200 (11600 instruction cycles)
;   Clock Frequency: 40.0MHz
;   Instruction cycle time: 50.ns
;**************************************************************************
;
;  Real-Valued Input Radix 2 Cooley-Tukey Decimation in Time FFT
;
;
;      normally ordered input data
;      normally ordered output data
;

;****************************************************************************
; Equates Section
;****************************************************************************

    RESET     equ      $00000000          ; reset isr
    MAIN      equ      $00000100          ; main routine

    points    equ      512                ;points=real data number /2
    passes    equ      9                  ;log2(points)=passes
    idata     equ      $0
    odata     equ      $1000
    coef      equ      $800
```

Figure B-1 Faster real FFT for the DSP96002 (sheet 1 of 4)

```
    BCRA      equ      $FFFFFFFE          ; port a bus control reg
    BCRB      equ      $FFFFFFFD          ; port b bus control reg
    PSR       equ      $FFFFFFFC          ; port select reg

    include   'sincosf.asm'               ;using external cos and sin table,
                                          ;if use internal ROM, delete this line
    include   'gen96.asm'
    include   'cfft96.asm'
    include   'split96.asm'


;***************************************************************************
;***************************************************************************
    sincosf   points,coef   ;twiddle factor for split is a full cycle sin and cos
    gen96     points,idata

    org   p:MAIN
    movep #$0,x:BCRA              ; no wait states for portb P,X,Y,I/O
    movep #$0,x:BCRB              ; ...don't care about page fault
    movep #$00FF00FF,x:PSR        ; external X:memory on Port-B
                                  ;          Y:memory on Port-A
    bclr  #$3,omr                 ; disable the internal data ROMs


    CFFT96  points,passes,idata,coef,odata
    SPLIT96 points,coef,odata
    nop
    nop
    jmp   *

    END


;
; Sine-Cosine Table Generator for rfft96.asm
;
; Last Update 5 August 92
;
sincosf  macro   points,coef
sincosf  ident   1,2
;
;       sincosf-   macro to generate sine and cosine coefficient
;                  lookup tables for Decimation in Time FFT
;                  twiddle factors.
;
;       points -   number of points (2 - 32768, power of 2)
;       coef   -   base address of sine/cosine table
;                  positive cosine value in X memory
;                  positive sine value in Y memory
;
;    8/12/92

pi       equ       3.141592654
freq     equ       2.0*pi/@cvf(points*2)

         org       x:coef-points/2
```

Figure B-1 Faster real FFT for the DSP96002 (sheet 2 of 4)

```
count    set      0
         dup      points/2
         dc       @cos(@cvf(count)*freq)
count    set      count+1
         endm

         org      y:coef-points/2
count    set      0
         dup      points/2
         dc       -@sin(@cvf(count)*freq)
count    set      count+1
         endm

freq1    equ      2.0*pi/@cvf(points)

       org    x:coef
count  set    0
       dup    points/2
       dc     -@cos(@cvf(count)*freq1)
count  set    count+1
       endm

       org    y:coef
count  set    0
       dup    points/2
       dc     -@sin(@cvf(count)*freq1)
count  set    count+1
       endm

       endm                                ;end of sincosf  macro

;***************************************************************************
;
; Split N/2 Complex FFT(Hn) for N real FFT(Fn)
;
SPLIT96 macro points,coef,odata
SPLIT96 ident 1,2
;
; Fi=0.5(Hi+Hn/2-i*)-0.5j(Hi-Hn/2-i*)W i=0,1,,,N-1
;
; points is real data /2
;
    move  #points-1,n0                     ;number of complex FFT input data
    move  #points/2-1,n4                   ;loop counter
    move  #odata,r0                        ;r0 ptr to A=Hi
    move  r0,r4                            ;r4 ptr to A’
    move  #-1,m6                           ;linear address
    move  m6,m5    move     m6,m4
    move  #coef-points/2+1,r6              ;twiddle factor start location
    lea   (r0)+n0,r1                       ;r1 ptr to B= Hn/2-i
    move  r1,r5                            ;r5 ptr to B’
    move  x:(r0)+,d0.s y:,d1.s             ;DC=Ar0+Ai0
    faddsub.s d0,d1    x:(r0)+,d2.s y:,d3.s   ;d0=Niquest=Ar0-Ai0,
                                           ;d1=DC,d2=Ar,d3=Ai
    move  d1.s,x:(r4)+ d0.s,y:             ;save DC and Niq
```

Figure B-1 Faster real FFT for the DSP96002 (sheet 3 of 4)

```
    move  x:(r1)-,d7.s y:,d1.s                   ;d7=Br,d1=Bi
    faddsub.s d7,d2     x:(r6)+,d8.s y:,d9.s     ;d7=Br-Ar=-H2i,
                                                 ;d2=Ar+Br=H1r,d8=Wr,d9=Wi<0
    fmpy  d9,d7,d0  faddsub.s d3,d1    d2.s,d5.s ;d0=Wi*H2i,d3=Ai-Bi=H1i,
                                                 ;d1=Ai+Bi=H2r,d5=H1r
    fmpy.sd8,d1,d1 d1.s,d6.s                     ;d1=Wr*H2r,d6=H2r
    move  #0.5,d4.s                              ;d4=0.5
;-----------------------------------------------------------------------
;  H1r=Ar+Br, H1i=Ai-Bi, H2r=Ai+Bi, H2i=Ar-Br
;  Ar'=(H1r+Wr*H2r-Wi*H2i)/2
;  Br'=(H1r-(Wr*H2r-Wi*H2i))/2
;  Ai'=(Wi*H2r-Wr*H2i+H1i)/2
;  Bi'=((Wi*H2r-Wr*H2i)-H1i)/2
;
;-----------------------------------------------------------------------

    do  n4,_end_split                                ;do points/2-1
    fmpy  d8,d7,d2  fsub.s   d0,d1    x:(r1),d7.s    ;d2=-Wr*H2i, d1=Wr*H2r-
                                                     ;Wi*H2i, d7=nBr
    fmpy  d9,d6,d0  faddsub.s d5,d1    x:(r6)+,d8.s y:,d9.s
                                 ;d0=Wi*H2r, d1=2*Ar',d5=2*Br',d8=nWr,d9=nWi
    fmpy  d4,d5,d2  fadd.s   d2,d0    x:(r0),d1.s  d1.s,d6.s
                                 ;d2=Br',d0=Wi*H2r-Wr*H2i, d6=2*Ar',d1=nAr
    fmpy  d4,d6,d2  faddsub.s d0,d3    d2.s,x:(r5)   ;d2=Ar', d3=2*Ai',
                                                     ;d0=2*Bi',save Br'
    fmpy.sd4,d3,d3 d2.s,x:(r4)   y:(r1)-,d2.s       ;d3=Ai',save Ar',d2=nBi
    fmpy  d4,d0,d0  faddsub.s     d7,d1  d3.s,d6.s    y:(r0)+,d3.s
;d0=Bi',d1=nAr+nBr,d7=nBr-nAr=nH2i,d3=nAi
    fmpy  d9,d7,d0  faddsub.s d3,d2 d1.s,d5.sd0.s,y:(r5)
;d0=nWi*nH2i,d2=nAi+nBi=nH2r,d3=nAi-nBi,save Bi'
    fmpy.s  d8,d2,d1    d2.s,d6.s d6.s,y:(r4)+  ;d1=nWr*nH2r,d6=nH2r,save Ai'
_end_split
    move      y:(r4),d0.s                    ;conjugate of last Ai element
    fneg.s    d0
    move      d0.s,y:(r4)

    endm                                     ;split
```

Figure B-1 Faster real FFT for the DSP96002 (sheet 4 of 4)

B.2 Real FFT for DSP56001/2

```
;
; This program originally available on the Motorola DSP bulletin board.
; It is provided under a DISCLAMER OF WARRANTY available from
; Motorola DSP Operation, 6501 Wm. Cannon Drive W., Austin, Tx., 78735.
;
; 1024-Point Real Input Non-In-Place FFT. (test program)
; 34886 clock cycles. Sampling period can be 0.87215ms @ 40 Mhz clock rate
; Use 292 program words,4*512 words for data and 2*(128+256) words for twiddle
; factor
;
; Store EVEN index input data to X memory and ODD index input data to Y.
; Assume scaling down at input, i.e. all input data are divided by 1024 before FFT.
; The outputs of this real input FFT are twice larger than true values. If the
; original FFT values are desired, scaling up factor should be 512.
;
; ‘sincosr’ generates twiddle factor for FFT.
; ‘bitrevtwd56’ sorts the twiddle factor in bit-reverse order.
; The generation and reordering of twiddle factors can be done off-line.
; ‘gen56’ generates input test signals, delete it if you provide input.
; ‘CFFT56’ does 512 points FFT.
; ‘SPLIT56’ extractes 512-point complex values for real input FFT.
; Only DC to Niquest frequency are calculated by this program.
; Input data always starts at location IDATA=0, a 512-complex buffer starts at any
; external memory location, ODATA, is required to hold 256-point output data
; groups.
;
; The output of the FFT replace the inputs started at IDATA.
; X:IDATA contains DC*2 and Y:IDATA contains Niquest*2.
;
;
RFFT56T ident  1,0
        page   132,60
        opt    nomd,nomex,loc,nocex,mu

        include     ‘sincosr’
        include     ‘bitrevtwd56’
        include     ‘gen56’
        include     ‘cfft56’
        include     ‘split56’
;
;
; Latest revision - Nov. 11 92

reset   equ         0
start   equ         $40
POINTS  equ         512
IDATA   equ         $00
ODATA   equ         $1000
COEF    equ         $800

        sincosr     POINTS,COEF
        gen56       POINTS,IDATA
```

Figure B-2 Real FFT for DSP56001/2 (sheet 1 of 5)

```
       opt        mex
       org        p:reset
       jmp        start

       org    p:start
       movep  #0,x:$fffe                      ;0 wait states
       bitrevtwd56     POINTS,COEF
       CFFT56          IDATA,COEF,POINTS,ODATA
       SPLIT56         IDATA,COEF,POINTS,ODATA

        end


;
; Sine-Cosine Table Generator for rfft56.asm
;
; Last Update 11/11/92
;
sincosr  macro   points,coef
sincosr  ident   1,2
;
;       sincosr-   macro to generate sine and cosine coefficient
;                  lookup tables for Decimation in Time real FFT
;                  twiddle factors. Only points/4 coefficients
;                  are generted. For real FFT another points/4
;                  coefficients with higher freq. are created.
;
;       points -   number of points (2 - 32768, power of 2)
;       coef   -   base address of sine/cosine table
;                  positive cosine value in X memory
;                  positive sine value in Y memory
;
;    8/12/92

pi      equ    3.141592654
freq    equ    2.0*pi/@cvf(points*2)

        org    x:coef-points/2
count   set    0
        dup    points/2
        dc     @cos(@cvf(count)*freq)
count   set    count+1
        endm

        org    y:coef-points/2
count   set    0
        dup    points/2
        dc     -@sin(@cvf(count)*freq)
count   set    count+1
        endm

freq1   equ    2.0*pi/@cvf(points)

        org    x:coef
count   set    0
```

Figure B-2 Real FFT for DSP56001/2 (sheet 2 of 5)

```
       dup    points/4
       dc     @cos(@cvf(count)*freq1)
count  set    count+1
       endm

       org    y:coef
count  set    0
       dup    points/4
       dc     @sin(@cvf(count)*freq1)
count  set    count+1
       endm

       endm                                        ;end of sincosr  macro


bitrevtwd56  macro   POINTS,COEF
bitrevtwd56  ident   1,2
;
;      bitrevtwd  -    macro to sort sine and cosine coefficient
;                      lookup tables in bit reverse order for 56156
;
;      POINTS     -    number of points (2 - 32768, power of 2)
;      COEF       -    base address of sine/cosine table
;                      negative cosine (Wr) and negative sine (Wi) in X memory
;
; Wei Chen
; July-28, 1992
;
       move   #COEF,r1              ;twiddle factor start address
       move   #0,m0                 ;bit reverse address
       move   #POINTS/8,n0          ;sincosr use N/4 points data,
                                    ;offset for bit rev. is N/8
       move   #POINTS/4-1,n2
       move   r1,r0                 ;r1 ptr to normal order data
       move   (r1)+                 ;no swap on 1st data
       move   (r0)+n0               ;r0 ptr to bitrev
       do     n2,_end_bit           ;does N/4-1 points swap
       move   r1,x0
       move   r0,b
       cmp    x0,b
       jgt    _swap
       move   (r1)+                 ;no swap but update points
       move   (r0)+n0
       jmp    _nothing
_swap
       move   r1,r5
       move   r0,r4
       move   x:(r1),x0 y:(r5),y0
       move   x:(r0),a  y:(r4),b
       move   x0,x:(r0)+n0 y0,y:(r4)
       move   a,x:(r1)+  b,y:(r5)
_nothing
       nop
_end_bit
       endm                         ;end of bitrevtwd macro
```

Figure B-2 Real FFT for DSP56001/2 (sheet 3 of 5)

```
;***************************************************************************
;
; Split N/2 Complex FFT(Hn) for N real FFT(Fn)
;
SPLIT56 macro IDATA,COEF,POINTS,ODATA
SPLIT56 ident    1,0
;
;
; Fi=0.5(Hi+Hn/2-i*)-0.5j(Hi-Hn/2-i*)W i=0,1,,,N-1
;
; Bit reverse input, Normal order output
; This macro amplifies coefficients of FFT by 2.
; If absolute values of spectrum are desired, then scaling up factor is 2^(N-1),
; assuming inputs are scaled by 2^N before complex FFT.
; POINTS is the number of real data /2
; COEF is twiddle factor location other than TF used in complex FFT (see sincosr)
;
;

    move      #POINTS-1,n0               ;number of complex FFT input data -1
    move      #POINTS/2-1,n2             ;loop counter
    move      #ODATA,r0                  ;r0 ptr to Ar=Hi
    move      #COEF-POINTS/2+1,r2        ;twiddle factor start location
    move      r2,r6                      ;r6 -> Wi
    lea       (r0)+n0,r5                 ;r5 ptr to Br & Bi
    move      #IDATA,r3                  ;r3 pointer for A’
    move      r3,r4
    move      n0,r1                      ;r1 ptr for B’, r1=B
    move      #POINTS/2,n0
    move  n0,n5
    move  m5,m3                                  ;m3 and m1 linear address
    move  m5,m1
    move  #0,m0                                  ;bit reverse address
    move  m0,m5                                  ;bit reverse address
    move           x:(r0),b                      ;b=Ar0
    move           x:(r5),x1     y:(r0),a        ;a=Ai0,x1=Br
    add   a,b      x:(r1)+,x0                ;b=Ar0+Ai0=DC, for ptr reason inc r1
    subl  b,a      r3,r4                         ;r4 ptr to temp location
    asl   b                      y:(r1),a        ;a=something
    asl   a        b,x:(r3)+     y:(r5),b        ;a=Niquist=Ar0-Ai0,save DC,b=Bi
    move                         a,y:(r0)+n0     ;save Niq in y:ODATA temp,
    move                         y:(r0),y0       ;y0=Ai

    do    n2,_end_split
    add   y0,b     y0,a     a,y:(r1)-            ;b=Ai+Bi=H2r,a=Ai, save prev. Bi’
    subl    b,a    x:(r0),b  b,y1                ;a=Ai-Bi=H1i, b=Ar, y1=H2r
    sub   x1,b     x:(r0)+n0,a a,y:(r4)          ;b=Ar-Br=H2i,a=Ar again,
                                                 ;save H1i temp,r0->nA
    subl  b,a      x:(r2)+,x1    y:(r6)+,y0      ;a=Ar+Br=H1r,x1=Wr,y0=Wi
    mac   x1,y1,a  b,x0          a,y:(r5)  ;a=H1r+Wr*H2r,x0=H2i,save H1r temp
    macr  y0,x0,a                y:(r5)-n5,b     ;a=H1r+Wr*H2r-Wi*H2i=Ar’, b=H1r
    subl  a,b      a,x:(r3)  y:(r4),a            ;b=H1r-(Wr*H2r-Wi*H2i)=Br’,
                                                 ;a=H1i,save Ar’
    mac   -x1,x0,a b,x:(r1)      y:(r5),b        ;a=H1i-Wr*H2i,save Br’,b=nBi
```

Figure B-2 Real FFT for DSP56001/2 (sheet 4 of 5)

```
    macr  y1,y0,a                y:(r4),y0     ;a=Wi*H2r-Wr*H2i+H1i=Ai’,y0=H1i
again
    sub   y0,a      x:(r5),x1     a,y:(r3)+     ;a=Wi*H2r-Wr*H2i, x1=nBr,save
Ai’
    sub   y0,a                   y:(r0),y0     ;a=Wi*H2r-Wr*H2i-H1i=Bi’,y0=nAi
_end_split
    move  y0,a      a,y:(r1)                   ;save last Bi’,conjugate last Ai
    neg   a         #ODATA,r5
    move            #IDATA,r0
    move                         a,y:(r4)
    move                         y:(r5),a
    move                         a,y:(r0)      ;move Niq. back

    endm                                       ;split56
```

Figure B-2 Real FFT for DSP56001/2 (sheet 5 of 5)