Freescale Semiconductor
Application Note

Document Number: AN3680
Rev. 0, 11/2010

Software Optimization of DFTs and IDFTs Using the StarCore SC3850 DSP Core

The Discrete Fourier transform (DFT) is a transform used for Fourier analysis of a finite sampled sequence of a signal. The DFT is widely employed in signal processing and related fields to analyze the frequencies contained in a signal. The Fast Fourier Transform (FFT) is a numerically efficient algorithm used to compute the DFT. For a complex N-point Fourier transform, the FFT reduces the number of complex multiplications from the order of N^2 to the order of NlogN. Mixed radix-2/3/4/5 FFTs can be used to implement the DFT algorithm with reduced computation if the number of DFT point $N = 2^k 3^l 4^m 5^n$ (k, l, m, and n are positive integers). For example, an FFT of size 1200 can be calculated in five stages using radices of 5, 4, and 3 ($1{,}200 = 5 \times 5 \times 4 \times 4 \times 3$). Therefore, an efficient implementation for the DFT is possible using the mixed radix FFT calculation.

This application note describes the implementation of the DFT/IDFT using mixed radix-2/3/4/5 decimation in time (DIT) FFTs/IFFTs on the Freescale Semiconductor SC3850 Digital Signal Processor (DSP) core. This document discusses how to use new features available in the SC3850 core, such as dual-multiplier, to improve the performance of the FFTs. The data structure, code optimization, test vector generation, and performance results are also investigated in this document. Typical reference code is included in this document to demonstrate the implementation details.

Contents

1 Introduction

The discrete Fourier transform (DFT) plays an important role in the analysis, design, and implementation of discrete-time signal processing algorithms and systems because efficient algorithms exist for the computation of the DFT. These efficient algorithms are called fast Fourier transform (FFT) algorithms. In terms of multiplications and additions, the FFT algorithms can be orders of magnitude more efficient than competing algorithms.

It is well known that the DFT takes N^2 complex multiplications and N^2 complex additions for complex N-point. Direct computation of the DFT is inefficient. The basic idea of the FFT algorithm is to break up an N-point DFT transform into successive smaller and smaller transform known as a butterfly (basic computational element). The small transform used can be a 2-point DFT known as Radix-2, 4-point DFT known as Radix-4, or other points. A two-point butterfly requires 1 complex multiplication and 2 complex additions, and a 4-point butterfly requires 3 complex multiplications and 8 complex additions. Therefore, a Radix-2 FFT reduces the complexity of a N-point DFT down to $(N/2)\log_2 N$ complex multiplications and $N\log_2 N$ complex additions because there are $\log_2 N$ stages and at each stage there are N/2 2-point butterflies. For a Radix-4 FFT, there are $\log_4 N$ stages and, at each stage, there are N/4 butterflies. Thus, the total number of complex multiplication is $(3N/4)\log_4 N = (3N/8)\log_2 N$, and the number of required complex additions is $8(N/4)\log_4 N = N\log_2 N$.

Because the FFTs reduce the number of complex multiplications from the order of N^2 to the order of NlogN, they are often used to compute the DFT. For instance, the size of the DFT/IDFT in 3GPP-LTE can be calculated as a product of the factors 2, 3, 4, and 5, which is equal to $2^k\ 3^l\ 4^m\ 5^n$ (k, l, m, and n are positive integers). The specific sizes are [12, 24, 36, 48, 60, 72, 96, 108, 120, 144, 180, 192, 216, 240, 288, 300, 324, 360, 384, 432, 480, 540, 576, 600, 648, 720, 768, 864, 900, 960, 972, 1080, 1152, and 1200]. Therefore, an efficient implementation of DFT/IDFT is possible using the mixed radix FFT/IFFT calculation.

The FFTs are of importance to a wide variety of applications, such as telecommunications. For instance, M-point DFT and IDFT are used to convert the signal between frequency domain and time domain in Third Generation Partnership Project-Long Term Evolution (3GPP-LTE).

To begin, consider an example that demonstrates how FFTs are used in real applications. In the 3GPP-LTE (Long Term Evolution), M-point DFT and Inverse DFT (IDFT) are used to convert the signal between frequency domain and time domain. 3GPP-LTE aims to provide for an uplink speed of up to 50 Mbps and a downlink speed of up to 100 Mbps. For this purpose, the 3GPP-LTE physical layer uses Orthogonal Frequency Division Multiple Access (OFDMA) on the downlink and Single Carrier-Frequency Division Multiple Access (SC-FDMA) on the uplink. Figure 1 shows the transmitter and receiver structure of OFDMA and SC-FDMA systems.

We can see from this example that DFT and IDFT are the key elements used to represent changing signals in time and frequency domains. In real applications, FFTs are normally used instead of direct DFT calculation to achieve higher performance.

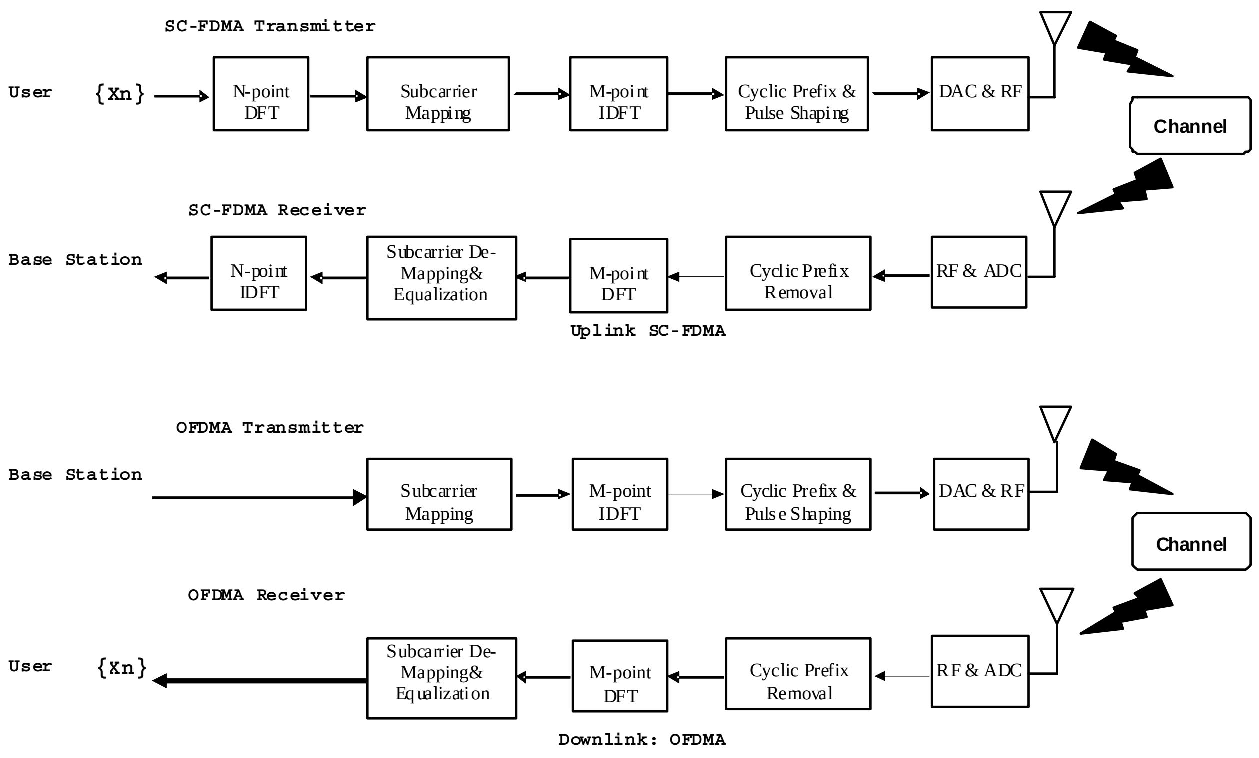


Figure 1. Transmitter and Receiver Structure of SC-FDMA and OFDMA Systems

2 Organization

This application note describes how to implement DFT/IDFT using the mixed radix FFT/IFFT algorithms on the StarCore DSP SC3850.The rest of the document is organized as follows:

- Section 3, “Mixed Radix FFT Algorithms”—Reviews the basic information on DFTs s and provides a brief introduction to the mixed radix FFT algorithms.
- Section 4, “Implementation of DFT on StarCore SC3850 DSP”—Discusses the algorithm implementation on SC3850, fixed point issues, and how to optimize the implementation for SC3850.
- Section 5, “Experimental Results”—Present results and discusses performance.
- Section 6, “Conclusions”—Presents the conclusions.
- Section 7, “References”—Provides a list of useful references.
- Appendix A, “DFT Reference Code”—Provides example code.

3 Mixed Radix FFT Algorithms

The following sections provide background information, definitions, and examples for using FFT and inverse FFT algorithms.

3.1 Definition of DFT and IDFT

The DFT X(k), k=0,1,2,...,N-1 of a sequence x(n), n=0,1,2,...,N-1 is defined as

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n)\exp\left(-j\frac{2\pi nk}{N}\right) \\ &= \sum_{n=0}^{N-1} x(n)W_N^{nk} \end{aligned}$$

Eqn. 1

$$\begin{aligned} W_N^{nk} &= \exp\left(-j\frac{2\pi nk}{N}\right) \\ &= \cos\left(\frac{2\pi nk}{N}\right) - j\sin\left(\frac{2\pi nk}{N}\right) \end{aligned}$$

Eqn. 2

In Equation 1 and Equation 2, N is the number of data, $j = \sqrt{-1}$, and W_N^{nk} is the twiddle factor. Equation 1 is called the N-point DFT of the sequence of x(n). For each value of k, the value of X(k) represents the Fourier transform at the frequency $\frac{2\pi k}{N}$. The IDFT is defined as follows:

$$\begin{aligned} x(n) &= \frac{1}{N}\sum_{k=0}^{N-1} X(k)\exp\left(j\frac{2\pi nk}{N}\right) \\ &= \frac{1}{N}\sum_{k=0}^{N-1} X(k)W_N^{-nk} \end{aligned}$$

Eqn. 3

$$\begin{aligned} W_N^{-nk} &= \exp\left(j\frac{2\pi nk}{N}\right) \\ &= \cos\left(\frac{2\pi nk}{N}\right) + j\sin\left(\frac{2\pi nk}{N}\right) \end{aligned}$$

Eqn. 4

Equation 3 is essentially the same as Equation 1. The differences are that the exponent of the twiddle factor in Equation 3 is the negative of the one in Equation 1 and the scaling factor is 1/N. The IDFT can be simply computed using the same algorithms for DFT but with conjugated twiddle factors. Alternatively, we can use the same twiddles factors for DFT with conjugated input and output to compute IDFT. Equation 1 is also called the *analysis equation* and Equation 3 the *synthesis equation*.

From Equation 1, it is clear that computing X(k) for each k requires N complex multiplications and N-1 complex additions. So, for N values of k, that is, for the entire DFT, it requires N^2 complex multiplications and $N(N-1) \approx N^2$ complex additions. Thus, the DFT is very computationally intensive. Note that a multiplication of two complex numbers $((a+jb)c+jd) = (ac-bd)+j(bc+ad)$ requires four real multiplications and two real additions. A complex addition $(a+jb)+(c+jd) = (a+c)+j(b+d)$ requires two real additions.

The Fast Fourier transforms (FFT) refers to algorithms that compute the DFT in a numerically efficient manner. Different radices 2, 3, 4, and 5 can be combined to calculate DFT if the number of DFT point $N = 2^k\, 3^l\, 4^m\, 5^n$ (k, l, m, and n are positive integers). Therefore, in the following sections, we describe how to combine different radix FFTs and then review the decimation-in-time (DIT) algorithms for radix-2/3/4/5 FFTs. Please note that the DIT algorithm is more efficient to run on an SC3850 core because the core performs dual MACs.

3.2 Combination of Different Radix FFTs

The number of input data (the number of DFT points), N, is equal to the product of the FFT radixes, which is defined as follows:

$$\begin{aligned} N &= R(0) \times R(1) \times R(2) \times \ldots \times R(S-1) \\ &= \prod_{s=0}^{S-1} R(s) \end{aligned} \qquad \textit{Eqn. 5}$$

In Equation 5, the parameters are defined as follows:

- S: Specifies the number of FFT stages
- s: Specifies stage index starting at the FFT input
- R(s): Specifies radix of the FFT stage

Neither he FFT stage radixes nor the R(s) parameters are constrained to be prime numbers. Mixed radix FFTs can be arbitrarily assigned in the sequence order. For example, the DFT of 300 points can be factored as 5 × 5 × 3 × 4 and can be calculated as a combination of either of the following FFT radixes:

- (radix-5) × (radix-5) × (radix-3) × (radix-4)
- (radix-4) × (radix-3) × (radix-5) × (radix-5)

Each stage of the FFT represents N/R(s) executions of an R(s) point butterfly calculation. In this application note, radices 2, 3, 4, and 5 DIT FFTs are investigated because they are relatively easy to develop and provide a fairly large set of possible number of points for DFT calculation.

3.3 Radix-5 DIT FFT

The example uses the symmetry and periodicity properties in the derivation, which are defined as shown in Equation 6.

$$\text{symmetry property:} \quad W_N^{k+\frac{N}{2}} = -W_n^k$$
$$\text{periodicity property:} \quad W_N^{k+N} = W_N^k \qquad \textbf{\textit{Eqn. 6}}$$

The Radix-5 DIT FFT algorithm breaks the DFT calculation into several 5-point DFTs. The Radix-5 DIT FFT can be derived as follows:

$$) = \sum_{n=0}^{N-1} x(n)W_N^{nk}$$

$$= \sum_{n=0}^{\frac{N}{5}-1} x(5n)W_N^{5nk} + \sum_{n=0}^{\frac{N}{5}-1} x(5n+1)W_N^{(5n+1)k} + \sum_{n=0}^{\frac{N}{5}-1} x(5n+2)W_N^{(5n+2)k} + \sum_{n=0}^{\frac{N}{5}-1} x(5n+3)W_N^{(5n+3)k} + \sum_{n=0}^{\frac{N}{5}-1} x(5n+4)W_N^{(5n+4)}$$

$$= \sum_{n=0}^{\frac{N}{5}-1} x(5n)W_N^{5nk} + W_N^k \sum_{n=0}^{\frac{N}{5}-1} x(5n+1)W_N^{5nk} + W_N^{2k} \sum_{n=0}^{\frac{N}{5}-1} x(5n+2)W_N^{5nk} + W_N^{3k} \sum_{n=0}^{\frac{N}{5}-1} x(5n+3)W_N^{5nk} + W_N^{4k} \sum_{n=0}^{\frac{N}{5}-1} x(5n+4)W_N^{5r} \qquad \textbf{\textit{Eqn. 7}}$$

$$= \sum_{n=0}^{\frac{N}{5}-1} x(5n)W_{N/5}^{nk} + W_N^k \sum_{n=0}^{\frac{N}{5}-1} x(5n+1)W_{N/5}^{nk} + W_N^{2k} \sum_{n=0}^{\frac{N}{5}-1} x(5n+2)W_{N/5}^{nk} + W_N^{3k} \sum_{n=0}^{\frac{N}{5}-1} x(5n+3)W_{N/5}^{nk} + W_N^{4k} \sum_{n=0}^{\frac{N}{5}-1} x(5n+4)W$$

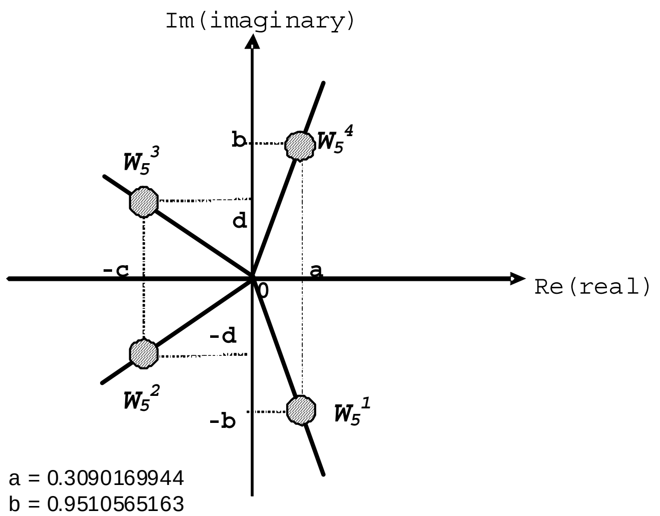
$$= P(k) + W_N^k Q(k) + W_N^{2k} R(k) + W_N^{3k} S(k) + W_N^{4k} T(k)$$

$$k = 0, 1, 2, 3, \ldots, N-1$$

Each of the sums, P(k), Q(k), R(k), S(k), and T(k), in Equation 7 is recognized as an N/5-point DFT. Although the index k ranges over N values, k=0,1,2,...,N-1, each of the sums can be computed only for k=0,1,2,...,N/5-1, since they are periodic with period N/5. The transform X(k) can be broken into five parts as follows:

$$
\begin{aligned}
X(k) &= P(k) + W_N^{k}Q(k) + W_N^{2k}R(k) + W_N^{3k}S(k) + W_N^{4k}T(k) \\
X\left(k+\frac{N}{5}\right) &= P(k) + W_N^{\left(k+\frac{N}{5}\right)}Q(k) + W_N^{2\left(k+\frac{N}{5}\right)}R(k) + W_N^{3\left(k+\frac{N}{5}\right)}S(k) + W_N^{4\left(k+\frac{N}{5}\right)}T(k) \\
&= P(k) + W^{1/5}W_N^{k}Q(k) + W^{2/5}W_N^{2k}R(k) + W^{3/5}W_N^{3k}S(k) + W^{4/5}W_N^{4k}T(k) \\
X\left(k+\frac{2N}{5}\right) &= P(k) + W_N^{\left(k+\frac{2N}{5}\right)}Q(k) + W_N^{2\left(k+\frac{2N}{5}\right)}R(k) + W_N^{3\left(k+\frac{2N}{5}\right)}S(k) + W_N^{4\left(k+\frac{2N}{5}\right)}T(k) \\
&= P(k) + W^{2/5}W_N^{k}Q(k) + W^{4/5}W_N^{2k}R(k) + W^{1/5}W_N^{3k}S(k) + W^{3/5}W_N^{4k}T(k) \\
X\left(k+\frac{3N}{5}\right) &= P(k) + W_N^{\left(k+\frac{3N}{5}\right)}Q(k) + W_N^{2\left(k+\frac{3N}{5}\right)}R(k) + W_N^{3\left(k+\frac{3N}{5}\right)}S(k) + W_N^{4\left(k+\frac{3N}{5}\right)}T(k) \\
&= P(k) + W^{3/5}W_N^{k}Q(k) + W^{1/5}W_N^{2k}R(k) + W^{4/5}W_N^{3k}S(k) + W^{2/5}W_N^{4k}T(k) \\
X\left(k+\frac{4N}{5}\right) &= P(k) + W_N^{\left(k+\frac{4N}{5}\right)}Q(k) + W_N^{2\left(k+\frac{4N}{5}\right)}R(k) + W_N^{3\left(k+\frac{4N}{5}\right)}S(k) + W_N^{4\left(k+\frac{4N}{5}\right)}T(k) \\
&= P(k) + W^{4/5}W_N^{k}Q(k) + W^{3/5}W_N^{2k}R(k) + W^{2/5}W_N^{3k}S(k) + W^{1/5}W_N^{4k}T(k) \\
k &= 0, 1, 2, \ldots, \frac{N}{5}-1
\end{aligned}
\qquad \textit{Eqn. 8}
$$

In the equation above, the periodicity property $W_N^{k+N} = W_N^{k}$ is used to simplify W_5^{k}. For example, $W_5^{12} = W_5^{2}$. The four complex numbers W_5^1, W_5^2, W_5^3, and W_5^4 can be expressed as shown in Figure 2.



a = 0.3090169944
b = 0.9510565163
c = 0.8090169944
d = 0.5877852523

Figure 2. Complex numbers of W_5^1, W_5^2, W_5^3, and W_5^4

$$\begin{aligned}
W_5^1 &= \exp\left(-j\frac{2\pi}{5}\right) = \cos\left(\frac{2\pi}{5}\right) - j\sin\left(\frac{2\pi}{5}\right) = a - jb \\
W_5^2 &= \exp\left(-j\frac{2\pi\times 2}{5}\right) = \cos\left(\frac{4\pi}{5}\right) - j\sin\left(\frac{4\pi}{5}\right) = -c - jd \\
W_5^3 &= \exp\left(-j\frac{2\pi\times 3}{5}\right) = \cos\left(\frac{6\pi}{5}\right) - j\sin\left(\frac{6\pi}{5}\right) = -c + jd \\
W_5^4 &= \exp\left(-j\frac{2\pi\times 4}{5}\right) = \cos\left(\frac{8\pi}{5}\right) - j\sin\left(\frac{8\pi}{5}\right) = a + jb
\end{aligned} \qquad \textit{Eqn. 9}$$

The constants in Equation 9 are as follows:

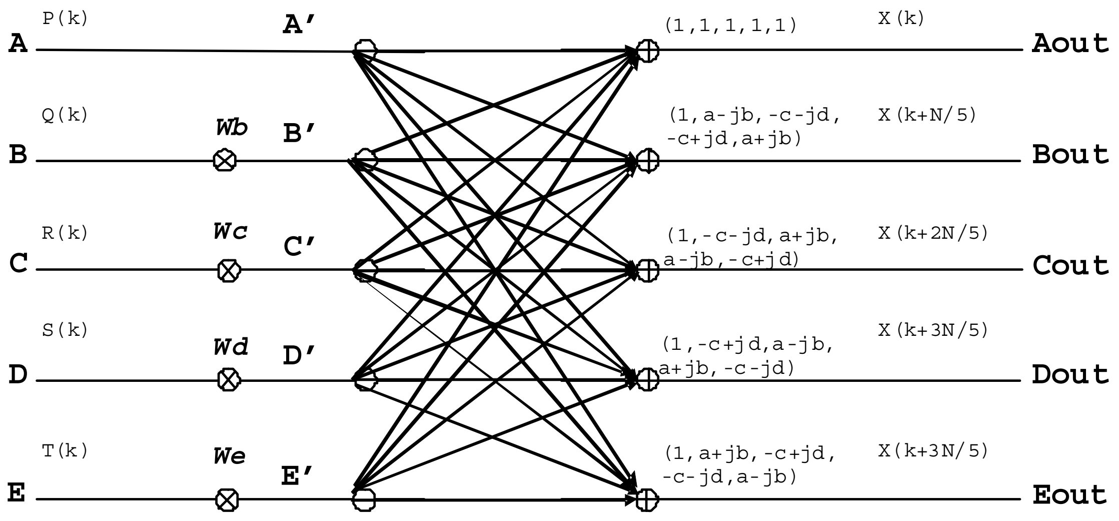
a = 0.3090169944,

b = 0.9510565163,

c = 0.8090169944,

d = 0.5877852523.

Figure 3 shows the corresponding Radix-5 DIT butterfly diagram.



$$Wb = \exp\left(-j\frac{2\pi}{N}k\right)$$
$$Wc = \exp\left(-j\frac{2\pi}{N}2k\right)$$
$$Wd = \exp\left(-j\frac{2\pi}{N}3k\right)$$
$$We = \exp\left(-j\frac{2\pi}{N}4k\right)$$

(k = 0,1, ...,N/5-1)

a = 0x278e
b = 0x79bc
c = 0x678e
d = 0x4b3d

Figure 3. Radix-5 DIT Butterfly

The twiddle factors in Figure 3 are defined as follows:

$$\begin{aligned} W_N^k &= Wb = Wbr + j \times Wb \\ W_N^{2k} &= Wc = Wcr + j \times Wc \\ W_N^{3k} &= Wd = Wdr + j \times Wd \\ W_N^{4k} &= We = Wer + j \times We \end{aligned}$$

Eqn. 10

The real and imaginary part of the outputs of Radix-5 DIT butterfly is specified as follows:

```
Aout_r = Ar'+Br'+Cr'+Dr'+Er'
Aout_i = Ai'+Bi'+Ci'+Di'+Ei'
Bout_r = Ar'+(a(Br'+Er')-c(Cr'+Dr'))+(b(Bi'-Ei')+d(Ci'-Di'))
Bout_i = Ai'+(a(Bi'+Ei')-c(Ci'+Di'))-(b(Br'-Er')+d(Cr'-Dr'))
Cout_r = Ar'-(c(Br'+Er')-a(Cr'+Dr'))+(d(Bi'-Ei')-b(Ci'-Di'))
Cout_i = Ai'-(c(Bi'+Ei')-a(Ci'+Di'))-(d(Br'-Er')-b(Cr'-Dr'))
Dout_r = Ar'-(c(Br'+Er')-a(Cr'+Dr'))-(d(Bi'-Ei')-b(Ci'-Di'))
Dout_i = Ai'-(c(Bi'+Ei')-a(Ci'+Di'))+(d(Br'-Er')-b(Cr'-Dr'))
Eout_r = Ar'+(a(Br'+Er')-c(Cr'+Dr'))-(b(Bi'-Ei')+d(Ci'-Di'))
Eout_i = Ai'+(a(Bi'+Ei')-c(Ci'+Di'))+(b(Br'-Er')+d(Cr'-Dr'))
where Ar', Ai', ...., Er', and Ei' are calculated as follows:
Ar' = Ar
Ai' = Ai
Br' = (Br*Wbr - Bi*Wbi)
Bi' = (Br*Wbi + Bi*Wbr)
Cr' = (Cr*Wcr - Ci*Wci)
Ci' = (Cr*Wci + Ci*Wcr)
Dr' = (Dr*Wdr - Di*Wdi)
Di' = (Dr*Wdi + Di*Wdr)
Er' = (Er*Wer - Ei*Wei)
Ei' = (Er*Wei + Ei*Wer)
```

NOTE

The principal difference between DIT and DIF (decimation in frequency) butterflies is that the order of calculation changes. In the DIF algorithm, the time domain data is “twiddled” before the sub-transforms are performed. In DIT, however, the sub-transforms are performed first, and the output is obtained by “twiddling” the resulting frequency domain data.

3.4 Radix-4 DIT FFT

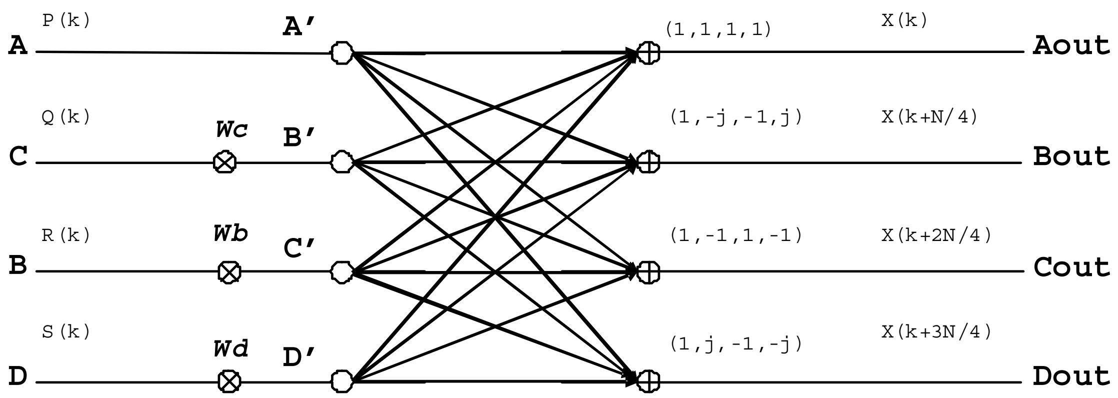
The Radix-4 DIT FFT algorithm breaks a N-point DFT calculation into a number of 4-point DFTs (4-point butterflies). Compared with direct computation of N-point DFT, 4-point butterfly calculation requires much less operations. The Radix-4 DIT FFT can be derived as follows:

$$\begin{aligned}
X(k) &= \sum_{n=0}^{N-1} x(n)W_N^{nk} \\
&= \sum_{n=0}^{\frac{N}{4}-1} x(4n)W_N^{4nk} + \sum_{n=0}^{\frac{N}{4}-1} x(4n+1)W_N^{(4n+1)k} + \sum_{n=0}^{\frac{N}{4}-1} x(4n+2)W_N^{(4n+2)k} + \sum_{n=0}^{\frac{N}{4}-1} x(4n+3)W_N^{(4n+3)k} \\
&= \sum_{n=0}^{\frac{N}{4}-1} x(4n)W_N^{4nk} + W_N^{k}\sum_{n=0}^{\frac{N}{4}-1} x(4n+1)W_N^{4nk} + W_N^{2k}\sum_{n=0}^{\frac{N}{4}-1} x(4n+2)W_N^{4nk} + W_N^{3k}\sum_{n=0}^{\frac{N}{4}-1} x(4n+3)W_N^{4nk} \\
&= \sum_{n=0}^{\frac{N}{4}-1} x(4n)W_{N/4}^{nk} + W_N^{k}\sum_{n=0}^{\frac{N}{4}-1} x(4n+1)W_{N/4}^{nk} + W_N^{2k}\sum_{n=0}^{\frac{N}{4}-1} x(4n+2)W_{N/4}^{nk} + W_N^{3k}\sum_{n=0}^{\frac{N}{4}-1} x(4n+3)W_{N/4}^{nk} \\
&= P(k) + W_N^{k}Q(k) + W_N^{2k}R(k) + W_N^{3k}S(k) \\
k &= 0, 1, 2, 3, \ldots, N-1
\end{aligned} \qquad \textit{Eqn. 11}$$

Each of the sums, P(k), Q(k), R(k), and S(k), in Equation 11 is recognized as an N/4-point DFT. Although the index k ranges over N values, k=0,1,2,...,N-1, each of the sums can be computed only for k=0,1,2,...,N/4-1, since they are periodic with period N/4. The transform X(k) can be broken into four parts as follows:

$$\begin{aligned}
X(k) &= P(k) + W_N^{k}Q(k) + W_N^{2k}R(k) + W_N^{3k}S(k) \\
X\left(k+\frac{N}{4}\right) &= P(k) + W_N^{\left(k+\frac{N}{4}\right)}Q(k) + W_N^{2\left(k+\frac{N}{4}\right)}R(k) + W_N^{3\left(k+\frac{N}{4}\right)}S(k) \\
&= P(k) - jW_N^{k}Q(k) - W_N^{2k}R(k) + jW_N^{3k}S(k) \\
X\left(k+\frac{2N}{4}\right) &= P(k) + W_N^{\left(k+\frac{2N}{4}\right)}Q(k) + W_N^{2\left(k+\frac{2N}{4}\right)}R(k) + W_N^{3\left(k+\frac{2N}{4}\right)}S(k) \\
&= P(k) - W_N^{k}Q(k) + W_N^{2k}R(k) - W_N^{3k}S(k) \\
X\left(k+\frac{3N}{4}\right) &= P(k) + W_N^{\left(k+\frac{3N}{4}\right)}Q(k) + W_N^{2\left(k+\frac{3N}{4}\right)}R(k) + W_N^{3\left(k+\frac{3N}{4}\right)}S(k) \\
&= P(k) + jW_N^{k}Q(k) - W_N^{2k}R(k) - jW_N^{3k}S(k) \\
k &= 0, 1, 2, \ldots, \frac{N}{4}-1
\end{aligned} \qquad \textit{Eqn. 12}$$

Figure 4 shows the corresponding Radix-4 DIT butterfly diagram, and Figure 5 shows its simplified version.



$$Wb = \exp\left(-j\frac{2\pi}{N}k\right)$$
$$Wc = \exp\left(-j\frac{2\pi}{N}2k\right) \qquad (k = 0,1, \ldots, N/4-1)$$
$$Wd = \exp\left(-j\frac{2\pi}{N}3k\right)$$

Figure 4. Radix-4 DIT Butterfly

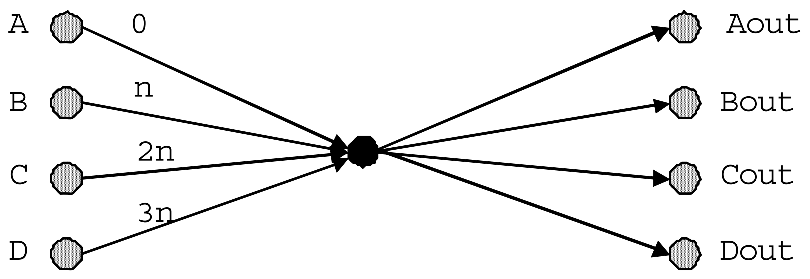


Figure 5. Simplified Radix-4 DIT Butterfly

The real and imaginary part of output data for Radix-4 DIT butterfly is specified in Equation 13.

$$\begin{aligned}
Aoutr' &= Ar + (Cr \times Wcr - Ci \times Wci) + (Br \times Wbr - Bi \times Wbi) + (Dr \times Wdr - Di \times Wdi) \\
Aouti' &= Ai + (Cr \times Wci + Ci \times Wcr) + (Br \times Wbi + Bi \times Wbr) + (Dr \times Wdi + Di \times Wdr) \\
Boutr' &= Ar - (Cr \times Wcr - Ci \times Wci) + (Br \times Wbi + Bi \times Wbr) - (Dr \times Wdi + Di \times Wdr) \\
Bouti' &= Ai - (Cr \times Wci + Ci \times Wcr) - (Br \times Wbr - Bi \times Wbi) + (Dr \times Wdr - Di \times Wdi) \\
Coutr' &= Ar + (Cr \times Wcr - Ci \times Wci) - (Br \times Wbr - Bi \times Wbi) - (Dr \times Wdr - Di \times Wdi) \\
Couti' &= Ai + (Cr \times Wci + Ci \times Wcr) - (Br \times Wbi + Bi \times Wbr) - (Dr \times Wdi + Di \times Wdr) \\
Doutr' &= Ar - (Cr \times Wcr - Ci \times Wci) - (Br \times Wbi + Bi \times Wbr) + (Dr \times Wdi + Di \times Wdr) \\
Douti' &= Ai - (Cr \times Wci + Ci \times Wcr) + (Br \times Wbr - Bi \times Wbi) - (Dr \times Wdr - Di \times Wdi)
\end{aligned} \qquad \textit{Eqn. 13}$$

3.5 Radix-3 DIT FFT

The Radix-3 DIF FFT can be derived as shown in Equation 14.

$$
\begin{aligned}
X(k) &= \sum_{n=0} x(n)W_N^{nk} \\
&= \sum_{n=0}^{\frac{N}{3}-1} x(3n)W_N^{3nk} + \sum_{n=0}^{\frac{N}{3}-1} x(3n+1)W_N^{(3n+1)k} + \sum_{n=0}^{\frac{N}{3}-1} x(3n+2)W_N^{(3n+2)k} \\
&= \sum_{n=0}^{\frac{N}{3}-1} x(3n)W_N^{3nk} + W_N^k \sum_{n=0}^{\frac{N}{3}-1} x(3n+1)W_N^{3nk} + W_N^{2k} \sum_{n=0}^{\frac{N}{3}-1} x(3n+2)W_N^{3nk} \\
&= \sum_{n=0}^{\frac{N}{3}-1} x(3n)W_{N/3}^{nk} + W_N^k \sum_{n=0}^{\frac{N}{3}-1} x(3n+1)W_{N/3}^{nk} + W_N^{2k} \sum_{n=0}^{\frac{N}{3}-1} x(3n+2)W_{N/3}^{nk} \\
&= P(k) + W_N^k Q(k) + W_N^{2k} R(k)
\end{aligned}
\qquad \textit{Eqn. 14}
$$

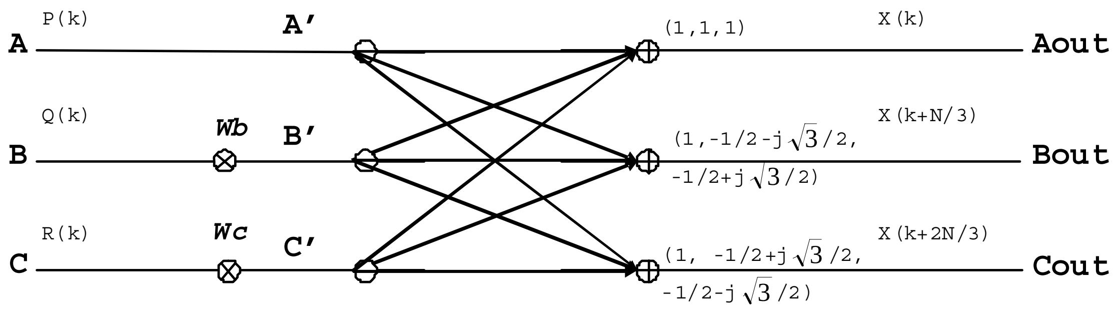
Each of the sums, P(k), Q(k), and R(k), in Equation 14 is recognized as an N/3-point DFT. The transform X(k) can be broken into three parts as shown in Equation 15.

$$
\begin{aligned}
X(k) &= P(k) + W_N^k Q(k) + W_N^{2k} R(k) \\
X\left(k + \frac{N}{3}\right) &= P(k) + W_N^{\left(k+\frac{N}{3}\right)} Q(k) + W_N^{2\left(k+\frac{N}{3}\right)} R(k) \\
&= P(k) + W^{1/3} W_N^k Q(k) + W^{2/3} W_N^{2k} R(k \\
\left(k + \frac{2N}{3}\right) &= P(k) + W_N^{\left(k+\frac{2N}{3}\right)} Q(k) + W_N^{2\left(k+\frac{2N}{3}\right)} R(\\
&= P(k) + W^{2/3} W_N^k Q(k) + W^{1/3} W_N^{2k} R(k \\
k &= 0, 1, 2, \ldots, \frac{N}{3} - 1
\end{aligned}
\qquad \textit{Eqn. 15}
$$

In Equation 15, the periodicity property $W_N^{k+N} = W_N^k$ is used to simplify $W_3^4 = W_3^1$. The complex numbers W_3^1 and W_3^2 can be expressed as shown in Equation 16.

$$
\begin{aligned}
W_3^1 &= \exp\left(-j\frac{2\pi}{3}\right) = -\frac{1}{2} - \frac{\sqrt{3}}{2}j \\
W_3^2 &= \exp\left(-j\frac{2\pi \times 2}{3}\right) = -\frac{1}{2} + \frac{\sqrt{3}}{2}j
\end{aligned}
\qquad \textit{Eqn. 16}
$$

Figure 6 shows the corresponding Radix-3 DIT butterfly diagram.



$$Wb = \exp\left(-j\frac{2\pi}{N}k\right) \qquad (k = 0,1, \ldots, N/3-1)$$
$$Wc = \exp\left(-j\frac{2\pi}{N}2k\right)$$

Figure 6. Radix-3 DIT Butterfly

The twiddle factors in Figure 6 are defined as shown in Equation 17.

$$W_N^k = Wb = Wbr + j \times Wb$$
$$W_N^{2k} = Wc = Wcr + j \times Wc \qquad \textit{Eqn. 17}$$

The real and imaginary part of the outputs of Radix-3 DIT butterfly is specified as follows:

```
Aout_r = Ar'+Br'+Cr'
Aout_i = Ai'+Bi'+Ci'
Bout_r = Ar'-(Br'+Cr')/2+(Bi'-Ci')*√3/2
Bout_i = Ai'-(Bi'+Ci')/2-(Br'-Cr')*√3/2
Cout_r = Ar'-(Br'+Cr')/2-(Bi'-Ci')*√3/2
Cout_i = Ai'-(Bi'+Ci')/2+(Br'-Cr')*√3/2
where Ar', Ai', Br', Bi', Cr', and Ci' are calculated as follows:
Ar' = Ar
Ai' = Ai
Br' = (Br*Wbr - Bi*Wbi)
Bi' = (Br*Wbi + Bi*Wbr)
Cr' = (Cr*Wcr - Ci*Wci)
Ci' = (Cr*Wci + Ci*Wcr)
```

3.6 Radix-2 DIT FFT

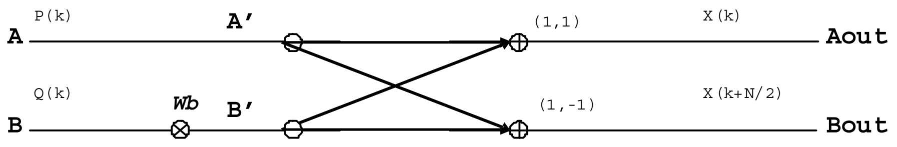
The Radix-2 DIF FFT can be derived as shown in Equation 18.

$$
\begin{aligned}
X(k) &= \sum_{n=0} x(n)W_N^{nk} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x(2n)W_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)W_N^{(2n+1)k} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x(2n)W_N^{2nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)W_N^{2nk} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x(2n)W_{N/2}^{nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)W_{N/2}^{nk} \\
&= P(k) + W_N^k Q(k)
\end{aligned}
\qquad \textbf{\textit{Eqn. 18}}
$$

Each of the sums, P(k) and Q(k), in Equation 18 is recognized as an N/2-point DFT. The transform X(k) can be broken into two parts as shown in Equation 19.

$$
\begin{aligned}
X(k) &= P(k) + W_N^k Q(k) \\
\left(k+\frac{N}{2}\right) &= P(k) + W_N^{\left(k+\frac{N}{2}\right)} Q(k \\
&= P(k) - W_N^k Q(k) \\
k &= 0, 1, 2, \ldots, \frac{N}{2} - 1
\end{aligned}
\qquad \textbf{\textit{Eqn. 19}}
$$

Figure 7 shows the corresponding Radix-2 DIT butterfly diagram.



$$Wb = \exp\left(-j\frac{2\pi}{N}k\right) \qquad (k = 0,1, \ldots,N/2-1)$$

Figure 7. Radix-2 DIT Butterfly

The twiddle factors in Figure 7 are defined in Equation 20.

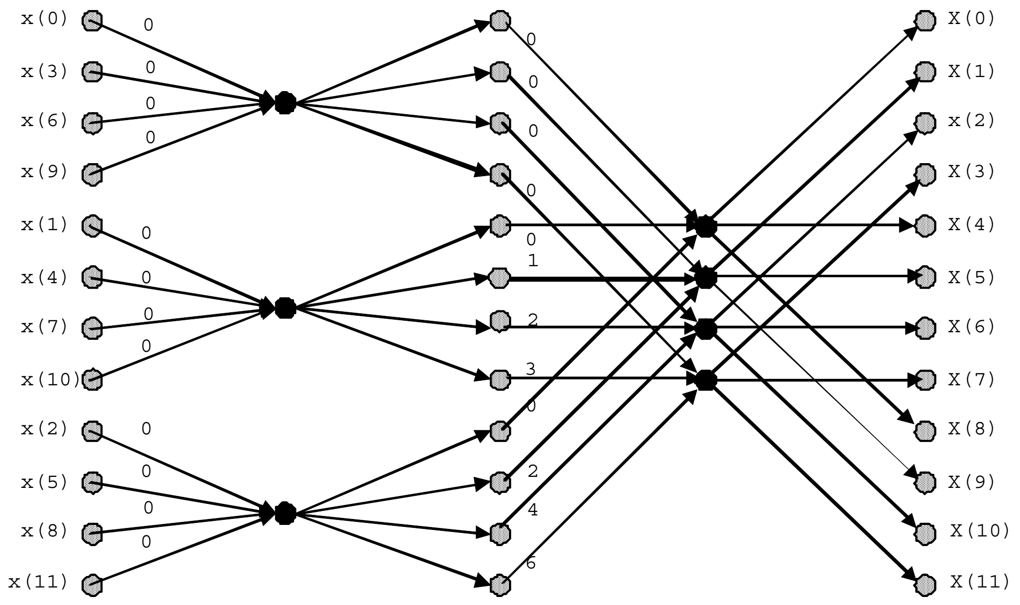
$$W_N^k = Wb = Wbr + j \times Wb \qquad \textit{Eqn. 20}$$

The real and imaginary part of the outputs of Radix-2 DIT butterfly is specified as follows:

```
Aout_r = Ar+(Br*Wbr - Bi*Wbi)
Aout_i = Ai+(Br*Wbi + Bi*Wbr)
Bout_r = Ar-(Br*Wbr - Bi*Wbi)
Bout_i = Ai-(Br*Wbi + Bi*Wbr)
```

3.7 An Example of 12-Point Mixed Radix DIT FFT

An example of 12-point mixed Radix DIT FFT diagram is shown in Figure 8 to illustrate an entire DIT algorithm. There are 2 stages to process the whole DFT computation. Radix-4 DIT butterflies are used in the first stage, and there 3 butterflies; Radix-3 DIT butterflies are used in the second stage, and there 4 butterflies.



Stage	0	1
Radix	4	3
Butterflies / Subgroup	1	4
Subgroups	3	1

Figure 8. An Example of a 12-point Mixed Radix DIT FFT

In Figure 8, the inputs are digit-reversed ordered while the outputs are presented in a normal order. The digital reversed order of the inputs are stored in an array and pre-calculated outside of the DFT function. The pointer of the array is passed into the DFT function to get the digital reversed indices.

The digital reversed indices *n* can be calculated as follows:

$$n = r(0)A(0) + r(1)A(1) + \ldots + r(S-1)A(S-1) \qquad \textit{Eqn. 21}$$

where S is the total number of stages, $r(s) = \{0, 1, \ldots, R(s) - 1\}$ specifies possible digits of Radix R(s), and $A(s) = \prod_{m=0}^{S-1} R(m)$ specifies the product of the radixes of the following stages.

For example, consider the digit-reversed order of the 300-point DFT. The 300-point DFT can be factored as $4 \times 3 \times 5 \times 5$ and we assume that the calculated radix of FFT stage is performed in this order. So the products of radixes of the following FFT stages and the digits at each stage are

At first stage, A(0) = 3 x 5 x 5 = 75, r(0) = {0,1,2,3},

At second stage, A(1) = 5 x 5 = 25, r(0) = {0,1,2},

At third stage, A(2) = 5, r(0) = {0,1,2,3,4},

At fourth stage, A(3) = 1, r(0) = {0,1,2,3,4},

The final output indexes in the digit-reversed order are calculated as

```
k   =  r(0)A(0)  + r(1)A(1)  + r(2)A(2)  + r(3)A(3)
0   =  0         + 0         + 0         + 0
75  =  75        + 0         + 0         + 0
150 =  150       + 0         + 0         + 0
225 =  225       + 0         + 0         + 0
25  =  0         + 25        + 0         + 0
100 =  75        + 25        + 0         + 0
175 =  150       + 25        + 0         + 0
250 =  225       + 25        + 0         + 0
50  =  0         + 50        + 0         + 0
125 =  75        + 50        + 0         + 0
200 =  150       + 50        + 0         + 0
275 =  225       + 50        + 0         + 0
5   =  0         + 0         + 5         + 0
80  =  75        + 0         + 5         + 0
155 =  150       + 0         + 5         + 0
230 =  225       + 0         + 5         + 0
...
```

The general C program to calculate the output indexes in the digit-reversed order is shown below. This program can calculate the output indexes in the digit-reversed order up to 6 FFT stages.

```
//=================================================
// Input indices are in the digit-reversed order
//=================================================
    short *psiRadix;           // Pointer to the array of radix every stage
    short *psiDigitReversedAddress; // Pointer to the Digit Reversed Address

    short siI;
    short siI0,siI1,siI2,siI3,siI4,siI5;
    short siIr0,siIr1,siIr2,siIr3,siIr4,siIr5;
    short siR[6];              // Local variable for radix every stage
    short siRR[6][5];          // Digits of the radix number
    short my_psiSubgroup[6];   // Product of radixes of the following DFT stages

    // Radix number every stage
    for(siI=0;siI<6;siI++)
    {
        siR[siI] = psiRadix[siI];
        if(psiRadix[siI] == 0)
        {
            siR[siI] = 1;
        }
    }

    // Loop index to calculate digit reversed address
    siIr0 = siR[0];
    siIr1 = siR[1];
    siIr2 = siR[2];
    siIr3 = siR[3];
    siIr4 = siR[4];
    siIr5 = siR[5];

    // Digits of the radix number
    for(siI=0;siI<5;siI++)
    {
        siRR[0][siI] = siI;
        siRR[1][siI] = siI;
        siRR[2][siI] = siI;
        siRR[3][siI] = siI;
        siRR[4][siI] = siI;
        siRR[5][siI] = siI;
    }

    // Product of radixes of the following DFT stages
    my_psiSubgroup[0] =  siIr1*siIr2*siIr3*siIr4*siIr5;
    for(siI=1;siI<6;siI++)
    {
        if(psiRadix[siI] == 0)
        {
            my_psiSubgroup[siI] =  0;
        }
        else
        {
            my_psiSubgroup[siI] =  my_psiSubgroup[siI-1]/siR[siI];
        }
    }
```

```
// Digit reversed address
 siI = 0;
 for(siI5=0;siI5<siIr5;siI5++)
 {
     for(siI4=0;siI4<siIr4;siI4++)
     {
         for(siI3=0;siI3<siIr3;siI3++)
         {
             for(siI2=0;siI2<siIr2;siI2++)
             {
                 for(siI1=0;siI1<siIr1;siI1++)
                 {
                     for(siI0=0;siI0<siIr0;siI0++)
                     {
                          psiDigitReversedAddress[siI]
                          = siRR[0][siI0]*my_psiSubgroup[0] +
                          siRR[1][siI1]*my_psiSubgroup[1]
                          + siRR[2][siI2]*my_psiSubgroup[2] +
                          siRR[3][siI3]*my_psiSubgroup[3]
                          + siRR[4][siI4]*my_psiSubgroup[4] +
                          siRR[5][siI5]*my_psiSubgroup[5];
                          siI++;
                     }
                 }
             }
         }
     }
 }
```

3.8 Mixed Radix Inverse FFT Algorithms

The common method of calculating the IFFT is to conjugate the twiddle factors and use the FFT routine to get the IFFT. However, this method needs an additional memory to store the twiddle factors. Therefore, it is not an efficient way for reduction of memory requirement.

If the complex conjugate of IDFT (Equation 3) is considered, the equation is defined as shown in Equation 22.

$$\begin{aligned} x^*(n) &= \frac{1}{N}\sum_{k=0}^{N-1}\left\lfloor X(k)\exp\left(j\frac{2\pi nk}{N}\right)\right\rfloor \\ &= \frac{1}{N}\sum_{k=0}^{N-1} X^*(k)\exp\left((-j)\frac{2\pi nk}{N}\right) \\ &= \frac{1}{N}\sum_{k=0}^{N-1} X^*(k)\left\{\cos\left(\frac{2\pi nk}{N}\right) - j\sin\left(\frac{2\pi nk}{N}\right)\right\} \\ &= \frac{1}{N}\sum X^*(k)W_N^{nk} \end{aligned} \qquad \textit{Eqn. 22}$$

The asterisk indicates the complex conjugate. In this form, there is no need to have additional memory for the twiddle factors since the twiddle factor is the same as the DFT. The IDFT is calculated by applying the DFT on the complex conjugate of X(k). The complex conjugate of the resulting time signal, x(n), is the output of the IDFT (with appropriate scaling). The advantage of this calculation is that the same twiddle factors can be used for both the DFT and the IDFT, and there is no need to save additional conjugated twiddle factors.

The following code describes the Radix-5 DIT butterfly calculations of the IFFT: Note that the conjugating process is embedded in the equations.

```
Aout_r = Ar'+Br'+Cr'+Dr'+Er'
Aout_i = Ai'-Bi'-Ci'-Di'-Ei'
Bout_r = Ar'+(a(Br'+Er')-c(Cr'+Dr'))+(b(Bi'-Ei')+d(Ci'-Di'))
Bout_i = Ai'-(a(Bi'+Ei')-c(Ci'+Di'))+(b(Br'-Er')+d(Cr'-Dr'))
Cout_r = Ar'-(c(Br'+Er')-a(Cr'+Dr'))+(d(Bi'-Ei')-b(Ci'-Di'))
Cout_i = Ai'+(c(Bi'+Ei')-a(Ci'+Di'))+(d(Br'-Er')-b(Cr'-Dr'))
Dout_r = Ar'-(c(Br'+Er')-a(Cr'+Dr'))-(d(Bi'-Ei')-b(Ci'-Di'))
Dout_i = Ai'+(c(Bi'+Ei')-a(Ci'+Di'))-(d(Br'-Er')-b(Cr'-Dr'))
Eout_r = Ar'+(a(Br'+Er')-c(Cr'+Dr'))-(b(Bi'-Ei')+d(Ci'-Di'))
Eout_i = Ai'-(a(Bi'+Ei')-c(Ci'+Di'))-(b(Br'-Er')+d(Cr'-Dr'))
where Ar', Ai', ...., Er', and Ei' are calculated as follows:
Ar' = Ar
Ai' = Ai
Br' = (Br*Wbr + Bi*Wbi)
Bi' = (Br*Wbi - Bi*Wbr)
Cr' = (Cr*Wcr + Ci*Wci)
Ci' = (Cr*Wci - Ci*Wcr)
Dr' = (Dr*Wdr + Di*Wdi)
Di' = (Dr*Wdi - Di*Wdr)
Er' = (Er*Wer + Ei*Wei)
Ei' = (Er*Wei - Ei*Wer)
a = 0.3090169944
b = 0.9510565163
c = 0.8090169944
d = 0.5877852523
```

The following code listing describes the Radix-4 DIT butterfly calculations of the IFFT:

```
Aout_r = (Ar'+Cr')+(Br'+Dr')
Aout_i = (Ai'-Ci')-(Bi'+Di'))
Bout_r = (Ar'+Cr')-(Br'+Dr')
Bout_i = (Ai'-Ci')+(Bi'+Di')
Cout_r = (Ar'-Cr')+(Bi'-Di')
Cout_i = (Ai'+Ci')+(Br'-Dr'))
Dout_r = (Ar'-Cr')-(Bi'-Di')
Dout_i = (Ai'+Ci')-(Br'-Dr')
where Ar', Ai', ...., Dr', and Di' are calculated as follows:
Ar' =  Ar
Ai' = -Ai
Br' = (Br*Wbr + Bi*Wbi)
Bi' = (Br*Wbi - Bi*Wbr)
Cr' = (Cr*Wcr + Ci*Wci)
Ci' = (Cr*Wci - Ci*Wcr)
Dr' = (Dr*Wdr + Di*Wdi)
Di' = (Dr*Wdi - Di*Wdr)
```

The following code listing describes the Radix-3 DIT butterfly calculations of the IFFT:

```
Aout_r = Ar'+Br'+Cr'
Aout_i = Ai'-Bi'-Ci'
Bout_r = Ar'-(Br'+Cr')/2+(Bi'-Ci')*√3/2
Bout_i = Ai'+(Bi'+Ci')/2+(Br'-Cr')*√3/2
Cout_r = Ar'-(Br'+Cr')/2-(Bi'-Ci')*√3/2
Cout_i = Ai'+(Bi'+Ci')/2-(Br'-Cr')*√3/2
where Ar', Ai', Br', Bi', Cr', and Ci' are calculated as follows:
Ar' = Ar
Ai' = Ai
Br' = (Br*Wbr + Bi*Wbi)
Bi' = (Br*Wbi - Bi*Wbr)
Cr' = (Cr*Wcr + Ci*Wci)
Ci' = (Cr*Wci - Ci*Wcr)
```

The following code listing describes the Radix-2 DIT butterfly calculations of the IFFT:

```
Aout_r = Ar+(Br*Wbr + Bi*Wbi)
Aout_i = Ai-(Br*Wbi - Bi*Wbr)
Bout_r = Ar-(Br*Wbr + Bi*Wbi)
Bout_i = Ai+(Br*Wbi - Bi*Wbr)
```

4 Implementation of DFT on StarCore SC3850 DSP

In this section, an implementation of the mixed Radix DIT FFT algorithm will be presented. Please note that the implementation of IDFT is similar, and thus the reference code of IDFT is not listed in this application note.

4.1 Overflow in Fixed-Point Computation and Scaling

The SC3850 is a fixed-point digital signal processor. In fixed-point processors, numbers are represented either as integers or as fractions between [-1.0, +1.0). In the implementation of the DFT algorithm, fractional number are used to interpret all input data, output data, and twiddle factors to avoid multiplication overflow because a fractional number multiplied with another fractional number always results in value less than one (under an assumption that the two operants can’t take -1 at the same time).

Another overflow comes from additions in the DFT calculation. For example, in a Radix-5 DIT butterfly, although the inputs of the butterfly have real and imaginary parts with magnitudes less than one, the maximum magnitude of outputs can have a up to 3-bit growth as shown in Equation 23.

$$\begin{aligned} A &= \sqrt{(Ar + Br + Cr + Dr + Er)^2 + (Ai + Bi + Ci + Di + Ei)^2} \\ &= \sqrt{(1+1+1+1+1)^2 + (1+1+1+1+1)^2} \\ &= 5\sqrt{2} \\ &= 7.071 \end{aligned} \quad \textit{Eqn. 23}$$

Scaling is an important operation in fixed-point DSP applications to avoid the overflow with the loss of precision and dynamic range. When the inputs are scaled to a smaller range, some of the lower-order bits of the original value are lost through truncation or rounding resulting in reduced precision. Thus, scaling must be done with great care, balancing the need to eliminate the possibility of overflow with the need to maintain adequate dynamic range and precision.

Two scaling methods to prevent overflowing are implemented in the DFT calculation. The first method is to scale down the input data by a fixed number of bits at each stage without considering the range of the input data. This is called fixed scaling method. The scaling factor is determined based on the maximum bit growth of the butterfly calculation. For example, the inputs to the Radix-5 butterfly are scaled by 3 bits (3-bit right shift) since it may cause up to 3-bit growth. This method is easy to implement. However, it may results in unnecessary scaling down of the input data, especially when the input data values are in a small range.

Another method is to detect the bit growth at the output of each DFT stage. Once the number of bit growth is known, the number of scaling bits can be determined to reverse the bit growth. This method is called automatic scaling method. The precision of the DFT stage can be enhanced by predicting bit growth of an upcoming stage and normalizing the data accordingly.

The automatic scaling method algorithm is as follows:

1. The input data are scaled down at the first stage to gain extra guard bits. The scaling factors are 2 for the first Radix-2 stage and 4 for the first Radix-4 stage.
2. At each butterfly calculation, search the maximum magnitude of the input data and detect the bit growth. The number of leading zeros or ones (except the sign bit) of the data with the maximum magnitude is recorded.
3. Based on the number of leading bits and the maximum bit growth S of the current DFT stage, the number of scaling bits is determined. The input data to the DFT stage are shifted accordingly. Table 1 shows the detailed scaling parameters used at different stages in the DFT calculation.

Table 1. Automatic Scaling at Different Stages

	Radix-2 First Stage	**N/A**	**N/A**

Automatic Scaling	if(leadbits > 2) Shift = 2 - leadBits;
	else Shift = 2 - leadBits
		N/A	N/A

	N/A	**Radix-3 Middle Stage**	**Radix-3 Last Stage**

Automatic Scaling	N/A	if(leadBits > 2) Shift = 0;
		else if(leadBits == 2) Shift = 1;
		else Shift = 2;
			if(leadBits > 2) Shift = 0;
			else Shift = 2 - leadBits;

	Radix-4 First Stage	**Radix-4 Middle Stage**	**N/A**

Automatic Scaling	if(leadbits > 3) Shift = 3- leadBits;
	else Shift = 3- leadBits
		if(leadBits > 3) Shift = 0;
		else if(leadBits == 3) Shift = 2;
		else Shift = 2;
			N/A

	N/A	**Radix-5 Middle Stage**	**Radix-5 Last Stage**

Automatic Scaling	N/A	if(leadBits > 3) Shift = 0;
		else Shift = 3 - leadBits;
			if(leadBits > 2) Shift = 2 - leadBits;
			else Shift = 2 - leadBits;

4. The final output data obtained by the DFT kernel may need to be scaled because the automatic scaling process may use different scaling factors for different input data. The DFT kernel does not implement this step since difference applications may require different scaling factors. Users can scale the output data outside of the DFT kernel based on application requirement.

The following code is used to find the maximum magnitude and determines the number of leading bits for the automatic scaling. Four real parts and four imaginary parts are compared simultaneously in a loop within two cycles.

```
;-- Find max value (real & imag) & stage initialization

    [
    move.l (sp-8),r0     ; input base (A)
    move.l (sp-60),r2    ; DFT point
    ]
    [
    move.w (r2),d15      ; DFT point
    ]
    [
    tfr d15,d14
    sub #8,d15           ; d15-8
    ]
    [
    asrr #2,d15          ; (d15-8)/4
    ]
    [
    adda #8,r0,r1        ; Input address
    move.w #2,n0
    ]
    [
    move.2l (r0)+n0,d0:d1
    move.2l (r1)+n0,d2:d3
    ]
    [
    abs2 d0,d0   abs2 d1,d1
    abs2 d2,d2   abs2 d3,d3
    move.2l (r0)+n0,d4:d5
    move.2l (r1)+n0,d6:d7
    ]
    [
    abs2 d4,d4   abs2 d5,d5
    abs2 d6,d6   abs2 d7,d7
    doensh0 d15          ; loop count for max search
    ]

    ; Loop for max (real,imag) search
    FALIGN
    loopstart0
        [
        max2 d0,d4   max2 d1,d5
        max2 d2,d6   max2 d3,d7
        move.2l (r0)+n0,d0:d1
        move.2l (r1)+n0,d2:d3
        ]
        [
        abs2 d0,d0   abs2 d1,d1
        abs2 d2,d2   abs2 d3,d3
        ]
    loopend0

    [
    max2 d0,d4   max2 d1,d5
    max2 d2,d6   max2 d3,d7
    move.l (sp-40),r3     ; Stage counter
    ]
    [
    tfr d6,d0   tfr d7,d1
    move.l (sp-20),r2      ; num_butterfly
    move.l (sp-24),r4      ; num_subgroup
    ]
    [
    max2 d0,d4   max2 d1,d5
    ]
    [
    tfr d5,d0
    ]
    [
    clr d15
    max2 d0,d4
    move.l (sp-32),r13     ; digit_reversed_address
    ]
    [
    add #2,d15             ; scale down = 2
    sxt.w d4,d0
    adda r3,r2          ; num_butterfly[s]
    adda r3,r4          ; num_subgroup[s]
    ]
    [
    extract #16,#16,d4,d4
    move.l (sp-68),r1   ; psiScale
    ]
    [
    max d0,d4           ; Max real/imag
    move.l (sp-8),r0    ; A input pointer
    move.l (sp-12),r9   ; A output pointer
    ]
    [
    clb d4,d0           ; count reading bit
    move.w (r2),r14     ; num_butterfly[s]
    move.w (r4),r5      ; num_subgroup[s]
    ]
    [
    neg d0              ; negates
    dosetup0 radix2_dit_1_stage_loop1
    move.l (sp-28),r4   ; num_radix_offset
    ]
    [
    clr d4
    sub #16,d0          ; limit 16-bit
    move.w #S22,d2
    ]
    [
    add #1,d4
    cmpgt.w #(S21+1),d0     ; Check if siNorm (leading bits) > S21+1
    move.w #S21,d1
    ]
    [
    ift clr d15             ; scale down = 0      if siNorm > S21+1
    ifa
    cmpeq.w #(S21+1),d0     ; Check if siNorm == S21+1
    ]
    tfrt d4,d15             ; scale down = 1     if siNorm == S21+1
    ; d15 is the automatic scaling bits
```

4.2 Implementation of Mixed Radix DIT FFTs

Figure 9 shows the actual process of calculation for 600-point DFT as an example.

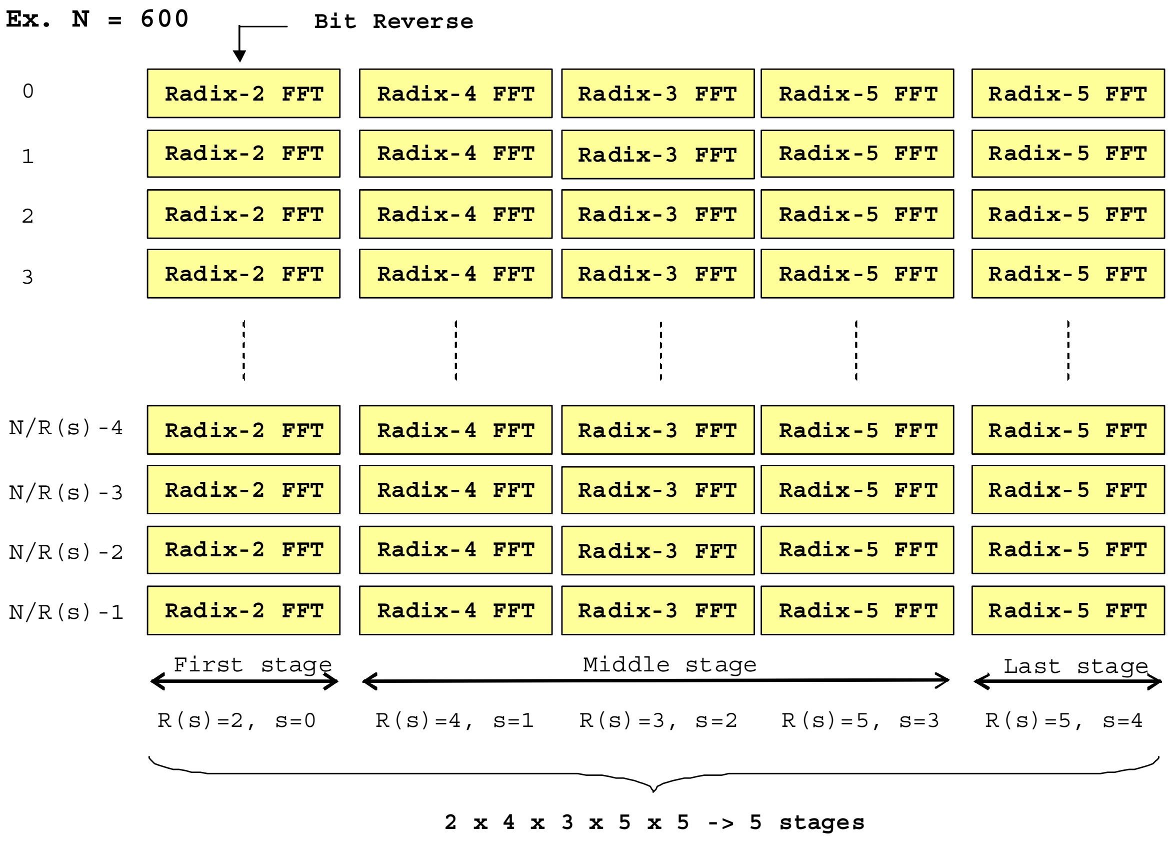


Figure 9. Diagram of 1024-point FFT Calculation Using Radix-4 DIT Algorithm

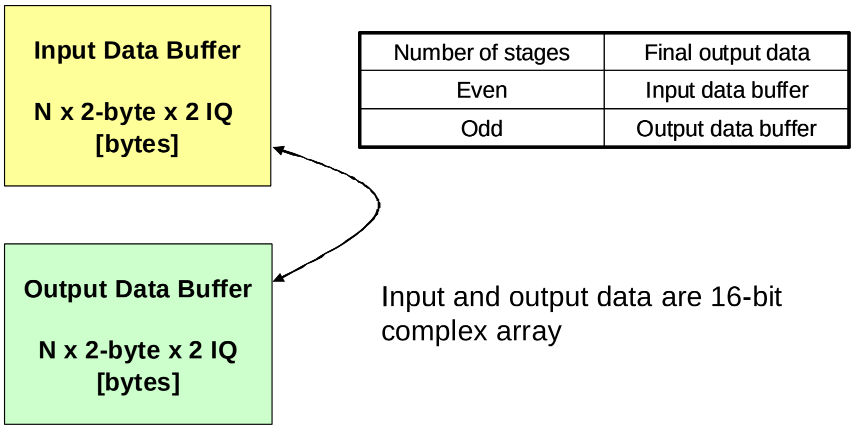
In this implementation, the DFT is factorized as $2 \times 4 \times 3 \times 5 \times 5$. There are S=5 stages, and the stage index s = 0, 1, 2, 3, 4. The whole DFT process is summarized as follows:

1. First stage: The input data are loaded in the bit-reversed addressing mode. The bit-reversed indices are calculated outside the DFT kernel and they are passed into the kernel as one of the parameters. The input data is checked to detect the maximum magnitude, and the scaling factor is determined. The Radix-2 (R(s=0)=2) DIT butterfly calculation is repeated for N/2 times on the input data. In the first stage, the twiddle factors are all equal to 1. As a result, there is no multiplication in the first stage. The data are scaled according to the scaling factor to prevent the overflow.
2. Middle stage: There 3 middle stages in this example. In each stage, there are N/R(s) Radix-R(s) DIT butterfly calculations, where s = 1, 2, 3. The twiddle factors vary with stages.
   - In stage s=1, the twiddle factors are equal to (W_N^{75k}, W_N^{2*75k}, W_N^{3*75k}) (k = 0, N/300-1).
   - In stage s=2, the twiddle factors are (W_N^{25k}, W_N^{2*25k}) (k = 0, 1, 2, ... , N/75-1).
   - In stage s=3, the twiddle factors are (W_N^{5k}, W_N^{2*5k}, W_N^{3*5k},W_N^{4*5k}) (k = 0, 1, 2, ... , N/25-1), and so on.

The input data is checked to detect the maximum magnitude, and the scaling factor is determined. The data are scaled according to the scaling factor to prevent the overflow.

3. Last stage: The Radix-5 (R(s=4)=5) DIT butterfly calculation is repeated for N/5 times. The twiddle factors are (W_N^k, W_N^{2k}, W_N^{3k},W_N^{4k}) (k = 0, 1, 2, ... , N/5-1) in this stage. The scaling factor is determined based on the maximum magnitude of the input data and the data are scaled accordingly.

Input and output data are 16-bit complex array, each of length N. Two separate buffers are used to store the input and output data, as shown in Figure 10. Each stage uses the input and output data buffer one after the other. Therefore, if stage n takes the input data buffer as the input and the output data buffer as the output, the next stage, n + 1, uses the output data buffer as the input and the input data buffer as the output. The final output data are stored in input data buffer if the number of stages S is even or in output data buffer if S is odd.


Input Data Buffer
N x 2-byte x 2 IQ [bytes]
Output Data Buffer
N x 2-byte x 2 IQ [bytes]
Input and output data are 16-bit complex array

Number of stages	Final output data
Even	Input data buffer
Odd	Output data buffer

Figure 10. Input and Output Data Buffers

At each stage, the output data are stored into the memory and they are used as input to the next stage. For example, if Radix-4 butterflies are used in stage n, then the outputs of a butterfly are A', B', C', and D'. Note that A', B', C', and D' are not consecutively stored in memory, as shown in Figure 11.

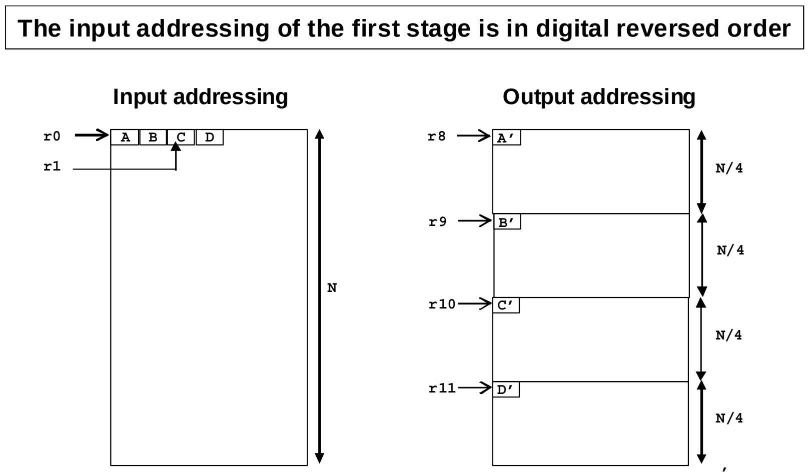


Figure 11. Input and output addressing in the DFT implementation

The offset value between the output data samples is calculated as shown in Equation 24.

$$4 \times \frac{N}{R(s)} \qquad \textbf{Eqn. 24}$$

where s specifies the stage index and $R(s)$ is the Radix of the FFT stage s. The offset value is multiplied by 4 to align the 2-byte for real part and 2-byte for imaginary part.

4.3 Optimization Used in DFT Implementation

The optimization techniques improve the performance by taking special advantage of StarCore's parallel computation capability. The SC3850 cores efficiently deploy a Variable Length Execution Set (VLES) instruction set. The SC38500 architecture has:

- Four parallel ALUs, each is capable of performing dual MAC and most other arithmetic operations
- Two Address Arithmetic Units (AAUs) for address arithmetic operations

A VLES can contain up to four data ALU instructions and two AAU instructions. To maximize the computational performance, four ALUs and two AAUs should be utilized simultaneously as much as possible. Because the StarCore architecture has high instruction-level parallelism, it is possible to schedule independent blocks in parallel to increase performance.

Code optimization also considers the memory structure to improve the performance. The SC3850 architecture provides a total sixteen 40-bit data registers, D0-D15 and sixteen 32-bit address registers, R0-R15. The dual Harvard architecture in the SC3850 is capable to access up to two 64-bit data per cycle. The following subsections present the optimization techniques used to increase the speed of the DIT DFT algorithm.

4.3.1 Single Instruction Multiple Data (SIMD) and Parallel Computing

The multiplication and addition operations in the DIT butterflies can be calculated in parallel using the multiple ALUs in SC3850. The SC3850 has some instructions which enable faster implementation of the DIT butterfly algorithms. The instructions used for the computation are shown in Table 2.

Table 2. SC3850 SIMD Instructions for DFT

Instruction	Description
MAX2 Da,Db	Writes the larger of the signed 16-bit values in each of the corresponding portions in a pair of source data registers to the second of the two registers (SIMD). Compares the high and low portions of the two registers independently.
ABS2 Da,Dn	Stores the absolute values of the high portion and low portion of the source data register in the high portion and low portion, respectively, of the destination data register (SIMD).
ASRR2 Da,Dn	Shifts the bits in each portion of the destination data register to the right, the number of positions specified in an immediate unsigned 4-bit value (SIMD).
SOD2FFCC	Sum or difference of two 16-bit values - function & cross (SIMD instruction). Performs two separate 16-bit additions or subtractions between the high and low portions of two source data registers and stores the results in the two portions of the destination data register. FF: A for addition and S for subtraction CC: XX for crossed and II for not crossed
MAC2FFGGR	SIMD signed fractional multiply and wide accumulate - real

Table 2. SC3850 SIMD Instructions for DFT

Instruction	Description
MAC2FFGGI	SIMD signed fractional multiply and wide accumulate - imaginary
MOVE2.2F	Transfers 2 16-bit fractional data between the memory and one data registers in packed 20-bit format, in a single 32-bit access
MOVE2.4F	Transfers 4 16-bit fractional data between the memory and two data registers in packed 20-bit format, in a single 64-bit access
`MOVERH.4F`	Move Four Wide High Fractional Words to Memory With Scaling, Rounding, and Saturation (AGU)
`MOVERL.4F`	Move Four Wide Low Fractional Words to Memory With Scaling, Rounding, and Saturation (AGU)

The functionality of the MAC2FFGGR and MACFFGGI are shown in Figure 12 and Figure 13, respectively.

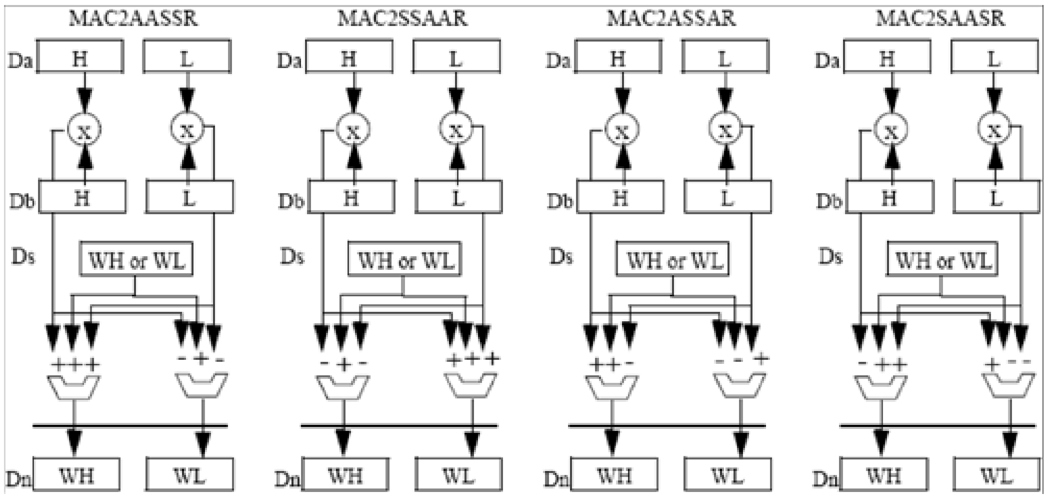


Figure 12. MAC2AASSR Da,Db,Ds.H/L,Dn

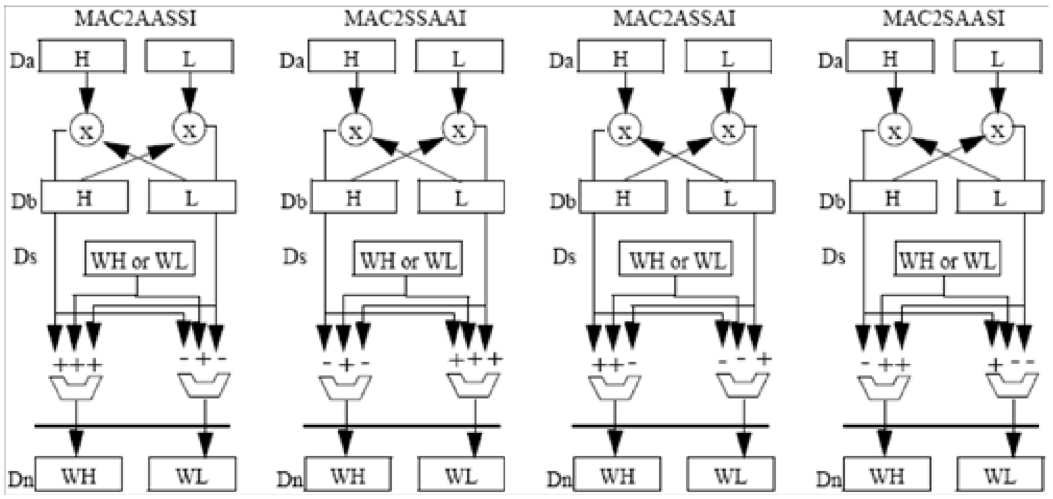


Figure 13. MAC2AASSI Da,Db,Ds.H/L,Dn

In the implementation, the parallel computing technique is used to maximize the usage of StarCore multiple ALUs for the calculation of independent output values that have an overlap input source data values. For example, in a Radix-4 DIT butterfly, outputs A', C', B', and D' can be calculated in parallel.

4.3.2 Packed Data Move

In the StarCore architecture, most element combinations that are supported have data width up to 64-bit. This means that four short words (4 × 16-bits) or two long words (2 × 32-bits) can be fetched at the same time. For example, the **MOVE2.2F, MOVE2.4F**, **MOVERH.4F** and **MOVERL.4F** instructions are used in the DFT computation to efficiently load and store the data samples. For this purpose, the memory addresses should be aligned to the access width of the instructions used. For example, 8-byte (4 × short words) accessing should be aligned to the 8-byte addresses.

4.3.3 Software Pipelining

Software pipelining is an optimization technique where a sequence of instructions is transformed into a pipeline of several copies of the sequence. The sequences then work in parallel to leverage more of the available parallelism of the architecture. For example, two set of Radix-4 DIT butterflies can be calculated inside one loop with the software pipelining.

4.3.4 Function Alignment

Functions should be aligned so that they fall on cache boundaries. On a Change of Flow (COF), the core has only filled one fetch buffer. If the destination VLES is within two fetch sets, the core will have to fill another fetch buffer and that will cause a stall. To avoid this situation, ensure that the destination VLES of all COF operations are contained within one fetch set. The **FALIGN** directive will pad the previous VLES with NOP to ensure the following VLES is contained within one fetch set.

4.4 DFT Implementation Diagrams

This is an implementation of the mixed Radix FFTs, where Radix-5, 4, 3, and 2 FFTs are combined together to perform the DFT computation. Decimation in time (DIT) algorithm is used. Bit-reversed addressing is not performed inside the kernel. The "psiDigitReversedAddress" array contains offsets for digit reversed address, which is pre calculated.

The butterfly diagrams are shown here to help understand the DFT algorithm. The DFT reference code is listed at Appendix A, “DFT Reference Code”. The implementation diagrams of IDFT are the same except some operation sign changes. The IDFT reference code is not included to save space.

```
************************************************************************************
     RADIX-2       FIRST LOOPS
     =======       =======================================

     INPUT:      r0 r1 r2 r3 in digital reversed order
     OUTPUT:     r8 r9 offset between pointers is 4*N/2
     TWIDDLES:   all equal to 1, no multiplication


                                   A+C
    (r0,r2)       A --------0---0------------------------- A' (r8)+
                             \ /+
                              \
                             / \+  A-C
    (r1,r3)       C --------0---0------------------------- B' (r9)+


     A' =A + C
     B' =A - C

     Fixed scaling: Scale down bits(d15) = 2
     Auto scaling:  if #of leading bits > 2, Scale up bits(d15) = #of leading bits - 2
                    else, Scale down bits(d15) = 2 - #of leading bits

************************************************************************************
```

Figure 14. First Stage Radix-2 Butterfly Diagram

```
************************************************************************************

     RADIX-3       MIDDLE LOOPS
     =======       =======================================

     INPUT:      r0
     OUTPUT:     r8 r9 r10, offset between pointers is 4*N/3
     TWIDDLES:   r5:Wb r6:Wc
     radix3parameter: r15


                                 (1,1,1)
        (r0)+      A -------O-------O------------------------------------ A' (r8)+
                             \\    //
                              \ x /
                              /\ /\  (1,-1/2-jsqrt(3)/2,-1/2+jsqrt(3)/2)
        (r0)+      B --Wb---O---x---O------------------------------------ B' (r9)+
                              \/ \/
                              / x \
                             //    \\ (1,-1/2+jsqrt(3)/2,-1/2-jsqrt(3)/2)
        (r0)+      C --Wc---O-------O------------------------------------ C' (r10)+


      Aout_r = Ar'+Br'+Cr'
      Aout_i = Ai'+Bi'+Ci'
      Bout_r = Ar'-(Br'+Cr')/2+(Bi'-Ci')*sqrt(3)/2
      Bout_i = Ai'-(Bi'+Ci')/2-(Br'-Cr')*sqrt(3)/2
      Cout_r = Ar'-(Br'+Cr')/2-(Bi'-Ci')*sqrt(3)/2
      Cout_i = Ai'-(Bi'+Ci')/2+(Br'-Cr')*sqrt(3)/2
 where Ar', Ai', Br', Bi', Cr', and Ci' are calculated as follows:
      Ar' = Ar
      Ai' = Ai
      Br' = (Br*Wbr - Bi*Wbi)
      Bi' = (Br*Wbi + Bi*Wbr)
      Cr' = (Cr*Wcr - Ci*Wci)
      Ci' = (Cr*Wci + Ci*Wcr)

     Fixed scaling: Scale down bits(d15) = 2
     Auto scaling:  if #of leading zeros > 2, Scale down bits(d15) = 0
                    if #of leading zeros = 2, Scale down bits(d15) = 1
                    else, Scale down bits(d15) = 2

************************************************************************************
```

Figure 15. Middle Stage Radix-3 Butterfly Diagram

```
****************************************************************************************

     RADIX-3       LAST LOOPS
     =======       =======================================

     INPUT:      r0
     OUTPUT:     r8 r9 r10, offset between pointers is 4*N/3
     TWIDDLES:   r12:Wb r13:Wc


                                    (1,1,1)
       (r0)+      A -------O-------O------------------------------------ A' (r8)+
                            \\   //
                             \ x /
                             /\ /\  (1,-1/2-jsqrt(3)/2,-1/2+jsqrt(3)/2)
       (r0)+      B --Wb---O---x---O------------------------------------ B' (r9)+
                             \/ \/
                             / x \
                            //   \\ (1,-1/2+jsqrt(3)/2,-1/2-jsqrt(3)/2)
       (r0)+      C --Wc---O-------O------------------------------------ C' (r10)+


      Aout_r = Ar'+Br'+Cr'
      Aout_i = Ai'+Bi'+Ci'
      Bout_r = Ar'-(Br'+Cr')/2+(Bi'-Ci')*sqrt(3)/2
      Bout_i = Ai'-(Bi'+Ci')/2-(Br'-Cr')*sqrt(3)/2
      Cout_r = Ar'-(Br'+Cr')/2-(Bi'-Ci')*sqrt(3)/2
      Cout_i = Ai'-(Bi'+Ci')/2+(Br'-Cr')*sqrt(3)/2
 where Ar', Ai', Br', Bi', Cr', and Ci' are calculated as follows:
      Ar' = Ar
      Ai' = Ai
      Br' = (Br*Wbr - Bi*Wbi)
      Bi' = (Br*Wbi + Bi*Wbr)
      Cr' = (Cr*Wcr - Ci*Wci)
      Ci' = (Cr*Wci + Ci*Wcr)

     Fixed scaling: Scale down bits(d15) = 2
     Auto scaling:  if #of leading zeros > 2, Scale down bits(d15) = 0
                    if #of leading zeros =< 2, Scale down bits(d15) = 2-#of leading zeros

*****************************************************************************************
```

Figure 16. Last Stage Radix-3 Butterfly Diagram

```
***************************************************************************************

     RADIX-4       FIRST LOOPS
     =======       =======================================

     INPUT:      r0 r1 r2  r3 digital reversed address
     OUTPUT:     r8 r9 r10 r11 offset between pointers is 4*N/4
     TWIDDLES:   all equal to 1. no pointers to twiddles.


                                   A+C
       (r0)       A --------0---0---------O------O---------------- A' (r8)+
                             \ /+          \    /+
                              \             \  /
                             / \+  A-C       \/
       (r1)       C --------0---0---------O--/\--O---------------- B' (r9)+
                                -          \/  \/+
                                           /\  /\
                                +  B+D    /  \/  \+
       (r2)       B --------0---0--------O---/\---O--------------- C' (r10)+
                             \ /            /  \  -
                              \            /    \
                             / \+  B-D    /      \+
       (r3)       D --------0---0---(-j)-O--------O--------------- D' (r11)+
                                -                 -

     A' =A + C +  (B + D)
     B' =A - C - j(B - D)
     C' =A + C -  (B + D)
     D' =A - C + j(B - D)

     Fixed scaling: Scale down bits(d15) = 3
     Auto scaling:  if #of leading bits > 3, Scale up bits(d15) = #of leading bits - 3
                    else, Scale down bits(d15) = 3 - #of leading bits

*********************************************************************************
```

Figure 17. First Stage Radix-4 Butterfly Diagram

```
*********************************************************************************

     RADIX-4       MIDDLE LOOPS
     =======       =======================================

     INPUT:      r0 r1 n3 = 3
     OUTPUT:     r8 r9 r10 r11 offset between pointers is 4*N/4
     TWIDDLES:   r5:Wb r6:Wc r7:Wd n0:twiddle factor's offset


      (r0)+       A --------0---0---------O------O---------------- A' (r8)+
                             \ /+          \    /+
                              \             \  /
                             / \+            \/
      (r1)+       C --Wc----0---0---------O--/\--O---------------- B' (r9)+
                                -          \/  \/+
                                           /\  /\
                                +         /  \/  \+
      (r0)+n3     B --Wb----0---0--------O---/\---O--------------- C' (r10)+
                             \ /            /  \  -
                              \            /    \
                             / \+         /      \+
      (r1)+n3     D --Wd----0---0---(-j)-O--------O--------------- D' (r11)+
                                -                 -

     Ar' = Ar +(Cr * Wcr - Ci * Wci)+(Br * Wbr - Bi * Wbi)+(Dr * Wdr - Di * Wdi)
     Ai' = Ai +(Cr * Wci + Ci * Wcr)+(Br * Wbi + Bi * Wbr)+(Dr * Wdi + Di * Wdr)
     Br' = Ar -(Cr * Wcr - Ci * Wci)+(Br * Wbi + Bi * Wbr)-(Dr * Wdi + Di * Wdr)
     Bi' = Ai -(Cr * Wci + Ci * Wcr)-(Br * Wbr - Bi * Wbi)+(Dr * Wdr - Di * Wdi)
     Cr' = Ar +(Cr * Wcr - Ci * Wci)-(Br * Wbr - Bi * Wbi)-(Dr * Wdr - Di * Wdi)
     Ci' = Ai +(Cr * Wci + Ci * Wcr)-(Br * Wbi + Bi * Wbr)-(Dr * Wdi + Di * Wdr)
     Dr' = Ar -(Cr * Wcr - Ci * Wci)-(Br * Wbi + Bi * Wbr)+(Dr * Wdi + Di * Wdr)
     Di' = Ai -(Cr * Wci + Ci * Wcr)+(Br * Wbr - Bi * Wbi)-(Dr * Wdr - Di * Wdi)

     Fixed scaling: Scale down bits(d15) = 2
     Auto scaling:  if #of leading zeros > 3, Scale down bits(d15) = 0
                    if #of leading zeros = 3, Scale down bits(d15) = 2
                    else, Scale down bits(d15) = 2

*********************************************************************************
```

Figure 18. Middle Stage Radix-4 Butterfly Diagram

```
*************************************************************************************
     RADIX-5       MIDDLE LOOPS
     =======       =======================================
     INPUT:      r0,r1, n0 = 2;
     OUTPUT:     r8 r9 r10 r11 r12, offset between pointers is 4*N/5
     TWIDDLES:   r4:Wb r5:Wc r6:Wd r7:We
     Parameters: r14 a:b    r15 c:d
                                    ___________
                              A'|           |(1,1,1,1,1)
       (r0)+n0     A ---------|-O   R   O-|------------------------------- Aout (r8)+
                                |     a      |
                                |     d      |
                              B'|     i      |(1,a-jb,-c-jd,-c+jd,a+jb)
       (r1)+n0     B --Wb----|-O   x   O-|------------------------------- Bout (r9)+
                                |            |
                                |     5      |
                              C'|            |(1,-c-jd,a+jb,a-jb,-c+jd)
       (r0)+n0     C --Wc----|-O   B   O-|------------------------------- Cout (r10)+
                                |     u      |
                                |     t      |
                              D'|     t      |(1,-c+jd,a-jb,a+jb,-c-jd)
       (r1)        D --Wd----|-O   e   O-|------------------------------- Dout (r11)+
                                |     r      |
                                |     f      |
                              E'|     l      |(1,a+jb,-c+jd,-c-jd,a-jb)
       (r0)+       E --We----|-O   y   O-|------------------------------- Eout (r12)+
                                |___________|

       Aout_r = Ar'+Br'+Cr'+Dr'+Er'
       Aout_i = Ai'+Bi'+Ci'+Di'+Ei'
       Bout_r = Ar'+(a(Br'+Er')-c(Cr'+Dr'))+(b(Bi'-Ei')+d(Ci'-Di'))
       Bout_i = Ai'+(a(Bi'+Ei')-c(Ci'+Di'))-(b(Br'-Er')+d(Cr'-Dr'))
       Cout_r = Ar'-(c(Br'+Er')-a(Cr'+Dr'))+(d(Bi'-Ei')-b(Ci'-Di'))
       Cout_i = Ai'-(c(Bi'+Ei')-a(Ci'+Di'))-(d(Br'-Er')-b(Cr'-Dr'))
       Dout_r = Ar'-(c(Br'+Er')-a(Cr'+Dr'))-(d(Bi'-Ei')-b(Ci'-Di'))
       Dout_i = Ai'-(c(Bi'+Ei')-a(Ci'+Di'))+(d(Br'-Er')-b(Cr'-Dr'))
       Eout_r = Ar'+(a(Br'+Er')-c(Cr'+Dr'))-(b(Bi'-Ei')+d(Ci'-Di'))
       Eout_i = Ai'+(a(Bi'+Ei')-c(Ci'+Di'))+(b(Br'-Er')+d(Cr'-Dr'))
   where Ar', Ai', ...., Er', and Ei' are calculated as follows:
       Ar' = Ar
       Ai' = Ai
       Br' = (Br*Wbr - Bi*Wbi)
       Bi' = (Br*Wbi + Bi*Wbr)
       Cr' = (Cr*Wcr - Ci*Wci)
       Ci' = (Cr*Wci + Ci*Wcr)
       Dr' = (Dr*Wdr - Di*Wdi)
       Di' = (Dr*Wdi + Di*Wdr)
       Er' = (Er*Wer - Ei*Wei)
       Ei' = (Er*Wei + Ei*Wer)
     Fixed scaling: Scale down bits(d0) = 3
     Auto scaling:  if #of leading zeros > 3, Scale down bits(d15) = 0
                    else, Scale down bits(d0) = 3-#of leading zeros
*************************************************************************************
     RADIX-5       LAST LOOPS
     =======       =======================================
     Diagram is the same as that for MIDDLE LOOPS
     Fixed scaling: Scale down bits(d12) = 3
     Auto scaling:  Scale down bits(d12) = 2-#of leading zeros
*************************************************************************************
```

Figure 19. Middle and Last Stage Radix-5 Butterfly Diagram

5 Experimental Results

This section provide the performance results of the DFT/IDFT kernel running on SC3850.

The real-time cycle counts for the implemented DIT DFT and IDFT kernels are measured using the SC3850 cycle accurate simulator. CodeWarrior Integrated Development Environment (IDE) R10.0 is used to build the DFT kernels and the corresponding test harness. The cycle counts for the DIT DFT and IDFT kernels are shown in Figure 20 and Figure 21. Automatic scaling method finds the maximum magnitude data at each DFT stage and thus requires roughly 30% more computation as shown in the figure.

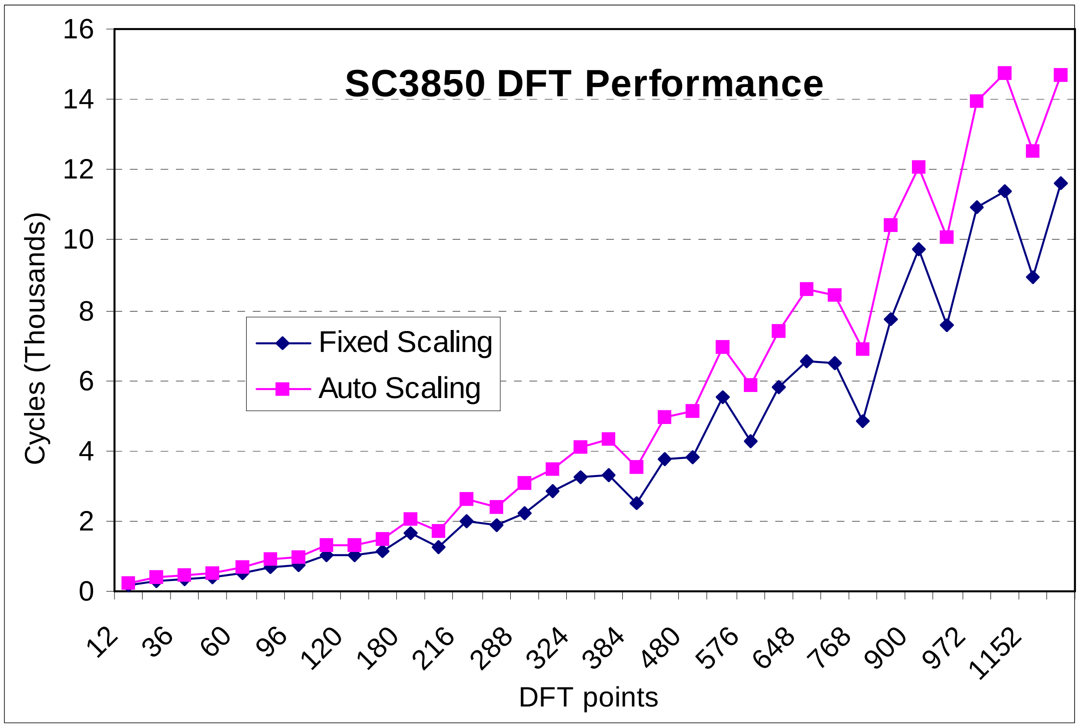


Figure 20. SC3850 DIT DFT Cycle Count at Different Points

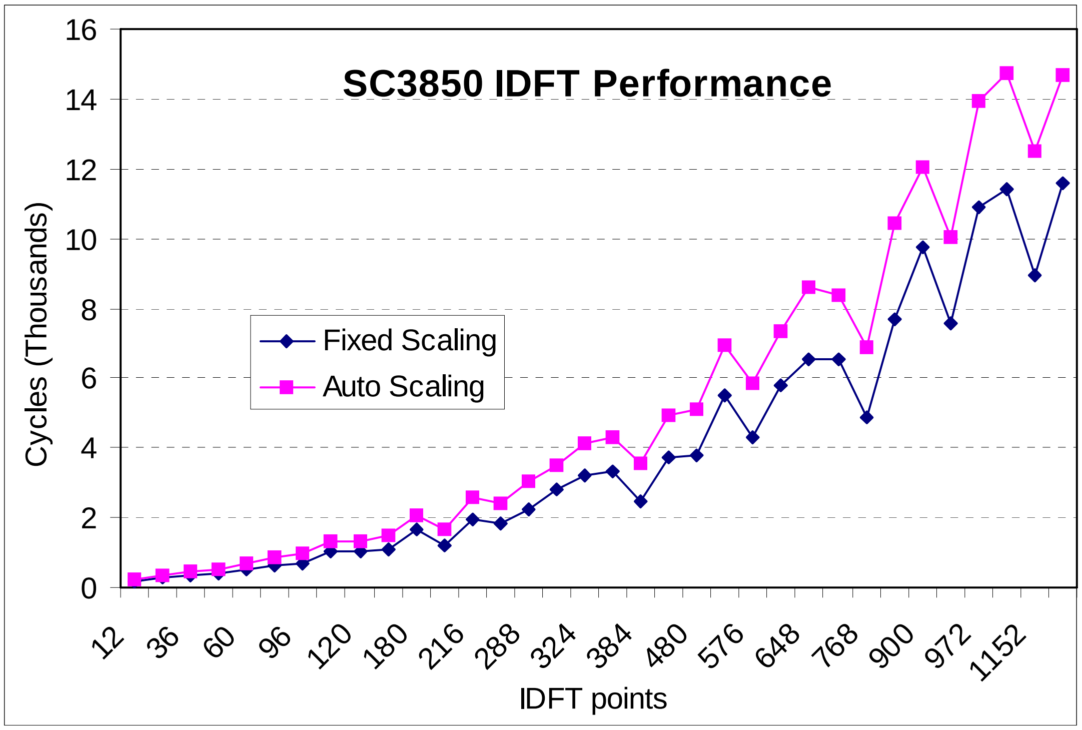


Figure 21. SC3850 DIT IDFT Cycle Count at Different Points

6 Conclusions

The purpose of this application note is to develop DIT DFT/IDFT algorithm implemented on the SC3850 core architecture. The DFT/IDFT is computed by a combination of Radix-2, 3, 4, and 5 FFTs/IFFTs to achieve good speed performance. The implementation benefits the following SC3850 features:

- VLES execution model, which utilizes maximum parallelism by allowing multiple address generation and data arithmetic logic units to execute multiple instructions in a single cycle
- Special SIMD instructions, such as MAX2, ABS2, SOD2ffcc, MAC2ffggR, and MAC2ffggI
- MOVER instructions with scaling, rounding, and limiting to avoid overflowing
- Memory access with multiple data, such as MOVE2.4F, MOVE2.2f, MOVERH.4f, and MOVERL.4F.

The SC3850 fixed-point code incurs very little loss of accuracy compared to the floating-point Matlab results. Fixed scaling and automatic scaling methods can be used to avoid the overflowing in the DFT calculation. In fixed scaling method, the scaling factors for the DFT stages are pre defined, and the algorithm scales the data without considering the data dynamic range. On the contrary, the automatic scaling method scales the data based on their dynamic range to obtain good bit precision, and thus produce results with higher SNR. But it needs to find the maximum magnitude of the input data to each stage, resulting in roughly 30% more computation.

The powerful architecture and instruction set of the SC3850 core permits flexible and compact coding of the algorithms in assembly language. The optimization of the DIT DFT/IDFT is done by taking special advantage of StarCore parallel computation capabilities, such as SIMD, parallel computing, and software pipelining.

7 References

1. Discrete-Time Signal Processing (Second Edition). Alan V. Oppenheim, Ronald W. Schafer, John R. Buck, Prentice Hall International, 1999.
2. The Fast Fourier Transform and Its Applications. E. Oran Brigham, Prentice Hall, 1988.
3. SC3850 DSP Core Reference Manual. Rev.A, Freescale Semiconductor, 08/2008.
4. Functional Differences Between the SC3400 and the SC3x50 Cores. Rev. C 06/2008.
5. SC3850 Programmer’s Guide. Rev. A, 06/2008.
6. Implementation of Radix-4 Fast Fourier Transform on StarCore DSP SC3400. AN3426 Rev. 0, 06/2007.
7. Implementation of Mixed Radix Fast Fourier Transform (FFT) and Inverse FFT on StarCore DSP SC3400. AN3443 Rev. A, 11/2007.
8. Software Optimization of FFTs and IFFTs Using the SC3850 Core. AN3666 Rev. C, 1/2009

NOTE

Freescale documents in the above list that use a letter for a revision designator are only available under NDA. Contact your local sales office or representative to access to these documents.

Appendix A DFT Reference Code

```
 DEFINE FIX_SCALE '1'
 UNDEF FIX_SCALE

 IF @DEF('FIX_SCALE')
SCALE_RADIX_2 EQU 1
SCALE_RADIX_3 EQU 2
SCALE_RADIX_4 EQU 2
SCALE_RADIX_5 EQU 3
 MSG 'FIX_SCALE is ON'
 ELSE
 MSG 'FIX_SCALE is OFF'
 ENDIF

S51  equ 2   ; Scaling check for radix-5 stage (Last stage)
S52  equ 2   ; Scaling check for radix-5 stage (Last stage)
S53  equ 3   ; Scaling check for radix-5 stage (Middle stage)
S54  equ 3   ; Scaling check for radix-5 stage (Middle stage)
S31  equ 2   ; Scaling check for radix-3 stage (Last stage)
S32  equ 2   ; Scaling check for radix-3 stage (Last stage)
S33  equ 2   ; Scaling check for radix-3 stage (Middle stage)
S34  equ 2   ; Scaling check for radix-3 stage (Middle stage)
S41  equ 2   ; Scaling check for radix-4 stage (Middle stage)
S42  equ 2   ; Scaling check for radix-4 stage (Middle stage)
S43  equ 2   ; Scaling check for radix-4 stage (1st stage)
S44  equ 2   ; Scaling check for radix-4 stage (1st stage)
S21  equ 1   ; Scaling check for radix-2 stage (1st stage)
S22  equ 1   ; Scaling check for radix-2 stage (1st stage)

   section .data

radix5parameter
   align 256
      dcw $278E  ; radix-5 a
      dcw $79BC  ; radix-5 b
      dcw $678E  ; radix-5 c
      dcw $4B3D  ; radix-5 d
radix3parameter
      dcw $6ED9  ; radix-3 sqrt(3)/2

    ALIGN 8
my_OUTPUT_pointer:
    DS 8

   endsec


   section .text

   global _sc3850_dft_dit_complex_16x16_auto_scale_asm
   align  16

_sc3850_dft_dit_complex_16x16_auto_scale_asm
init:
```

```
push.2l d6:d7     push.2l r6:r7
[
clr d3                  ; Clear for status register stack
clr d4                  ; Clear stage counter
adda #72,sp,r4
move.l  (r0)+,r1        ; base_address
]
[
tfra r4,sp
move.l r0,(my_OUTPUT_pointer)    ;// --> psFft->psiOut
]

[
move.l  (r0)+,r14       ; out_address
move.l r1,(sp-8)        ; base_address
]
[
move.l  (r0)+,r2        ; num_radix
move.l r14,(sp-12)      ; out_address
]
[
move.l  (r0)+,r3        ; num_butterfly
move.l r2,(sp-16)       ; num_radix
]
[
move.l  (r0)+,r4        ; num_subgroup
move.l r3,(sp-20)       ; num_butterfly
]
[
move.l  (r0)+,r5        ; num_radix_offset
move.l r4,(sp-24)       ; num_subgroup
]
[
move.l  (r0)+,r6        ; digit_reversed_address
move.l r5,(sp-28)       ; num_radix_offset
]
[
move.l r6,(sp-32)       ; digit_reversed_address
move.l sr,d3            ; Status Register
]
[
move.l d3,(sp-36)       ; Status register
move.l #$0BE4008C,SR ; no scaling, Two's-complement rounding, 32 bit saturate,
                     ; 16 bit saturate, W20=0
]
[
move.l  (r0)+,r7        ; wb
move.l d4,(sp-40)       ; Initialize the stage counter
]
[
move.l r7,(sp-44)       ; wb
move.l  (r0)+,r7        ; wc
]
[
move.l r7,(sp-48)       ; wc
move.l  (r0)+,r7        ; wd
]
```

```
        [
        move.l r7,(sp-52)     ; wd
        move.l (r0)+,r7       ; we
        ]
        [
        move.l (r0)+,r1       ; DFT point
        move.w (r2),d0        ; num_radix
        ]
        [
        move.l (r0)+,r4       ; pliAutoScale
        move.l r7,(sp-56)     ; we
        ]
        [
        move.l r1,(sp-60)     ; DFT point
        move.l r4,(sp-68)     ; pliAutoScale
        ]
        move.w (r2),d0        ; num_radix[s]
        cmpeq.w #2,d0         ; num_radix[s]=0 or not
        jf radix4_dit_1_stage     ; Go to adix4_1st_stage if num_radix[s]!=2


    ;////////////////////////////////////////////////////////////////////////
    ; Radix-2 Stage:
    ;////////////////////////////////////////////////////////////////////////

    radix2_init:

       ;---------------------------------------------------------
       ; 1st stage (radix-2)
       ;---------------------------------------------------------
    radix2_dit_1_stage:

     IF @DEF('FIX_SCALE')
        [
        move.l (sp-60),r2    ; DFT point
        move.l (sp-40),r3    ; Stage counter
        ]
        [
        move.l (sp-32),r13   ; digit_reversed_address
        move.l (sp-68),r1    ; psiScale
        ]
        [
        move.l (sp-8),r0     ; A input pointer
        move.l (sp-12),r9    ; A output pointer
        ]
        [
        dosetup0 radix2_dit_1_stage_loop1
        move.l (sp-28),r4    ; num_radix_offset
        ]
        [
        move.w (r2),d14      ; DFT point
        move.w #SCALE_RADIX_2,d15
        ]
     ELSE     ;//IF @DEF('FIX_SCALE')

        ;-- Find max value (real & imag) & stage initialization
```

```
[
move.l (sp-8),r0     ; input base (A)
move.l (sp-60),r2    ; DFT point
]
[
move.w (r2),d15      ; DFT point
]
[
tfr d15,d14
sub #8,d15           ; d15-8
]
[
asrr #2,d15          ; (d15-8)/4
]
[
adda #8,r0,r1        ; Input address
move.w #2,n0
]
[
move.2l (r0)+n0,d0:d1
move.2l (r1)+n0,d2:d3
]
[
abs2 d0,d0   abs2 d1,d1
abs2 d2,d2   abs2 d3,d3
move.2l (r0)+n0,d4:d5
move.2l (r1)+n0,d6:d7
]
[
abs2 d4,d4   abs2 d5,d5
abs2 d6,d6   abs2 d7,d7
doensh0 d15          ; loop count for max search
]

; Loop for max (real,imag) search
loopstart0
    [
    max2 d0,d4   max2 d1,d5
    max2 d2,d6   max2 d3,d7
    move.2l (r0)+n0,d0:d1
    move.2l (r1)+n0,d2:d3
    ]
    [
    abs2 d0,d0   abs2 d1,d1
    abs2 d2,d2   abs2 d3,d3
    ]
loopend0

[
max2 d0,d4   max2 d1,d5
max2 d2,d6   max2 d3,d7
move.l (sp-40),r3      ; Stage counter
]
[
tfr d6,d0   tfr d7,d1
move.l (sp-20),r2      ; num_butterfly
move.l (sp-24),r4      ; num_subgroup
```

```
         ]
         [
         max2 d0,d4   max2 d1,d5
         ]
         [
         tfr d5,d0
         ]
         [
         clr d15
         max2 d0,d4
         move.l (sp-32),r13    ; digit_reversed_address
         ]
         [
         add #2,d15            ; scale down = 2
         sxt.w d4,d0
         adda r3,r2          ; num_butterfly[s]
         adda r3,r4          ; num_subgroup[s]
         ]
         [
         extract #16,#16,d4,d4
         move.l (sp-68),r1   ; psiScale
         ]
         [
         max d0,d4           ; Max real/imag
         move.l (sp-8),r0    ; A input pointer
         move.l (sp-12),r9   ; A output pointer
         ]
         [
         clb d4,d0           ; count reading bit
         move.w (r2),r14     ; num_butterfly[s]
         move.w (r4),r5      ; num_subgroup[s]
         ]
         [
         neg d0              ; negates
         dosetup0 radix2_dit_1_stage_loop1
         move.l (sp-28),r4   ; num_radix_offset
         ]
         [
         clr d4
         sub #16,d0          ; limit 16-bit
         move.w #S22,d2
         ]
         [
         add #1,d4
         cmpgt.w #S21,d0     ; Check if siNorm > S21
         move.w #S21,d1
         ]
         [
         ift clr d15         ; scale down = 0      if siNorm > S21+1
         ifa
         cmpeq.w #S21,d0     ; Check if siNorm == S21
         ]
         tfrt d4,d15         ; scale down = 1     if siNorm == S21+1

      ENDIF    ;// IF @DEF('FIX_SCALE')
         [
         move.l d15,(sp-64) ; Set norm to stack
```

```
move.w d15,(r1)+    ; psiScale <- norm
]
[
move.l r1,(sp-68)   ; psiScale
asrr #2,d14         ; DFT point/4
move.l d15,r5       ; scale down
]
[
doen0 d14           ; loop count
adda #2,r13,r14
]
[
adda r3,r4          ; num_radix_offset[s]
move.l SR,d3
]
[
move.w (r4),r1      ; num_radix_offset[s]
move.l (sp-32),r15  ; digit_reversed_address
]
[
move.l d3,r4
move.l #(_psiDigitReversedAddress_absolut_address),r15    ;///
]

; For next stage input/output
[
move.l r9,(sp-8)    ; base_address
move.l r0,(sp-12)   ; out_address
]
[
bmset #$1000,SR.L   ; W20=1
tfra r9,r8          ; A output
]
[
bmclr #$0080,SR.L   ; SM2=0
adda r1,r9          ; B output pointer (long)
]
[
bmclr #$0030,SR.L   ; SCM=00
tfra r9,r10         ; B output
]

;--- Start Radix-2 (1st stage)

[
move.l (r15)+,r0     ; digit_reversed_address(i+0)
cmpeqa.w #1,r5
]
[
ift bmset #$0010,SR.L     ; SCM=01
ifa
move.l (r15)+,r1     ; digit_reversed_address(i+1)
]
[
move.l (r15)+,r2     ; digit_reversed_address(i+2)
cmpeqa.w #2,r5
]
```

```
        [
        ift bmset #$0030,SR.L     ; SCM=11
        ifa
        move.l (r15)+,r3      ; digit_reversed_address(i+3)
        ]
        [
        MOVE2.2F (r0),d0      ; d0=Ar:Ai
        move.l (r15)+,r0      ; digit_reversed_address(i+0)
        ]
        [
        MOVE2.2F (r1),d1      ; d1=Br:Bi
        move.l (r15)+,r1      ; digit_reversed_address(i+1)
        ]
        [
        MOVE2.2F (r2),d4      ; d4=Ar:Ai
        move.l (r15)+,r2      ; digit_reversed_address(i+2)
        ]
        FALIGN
    radix2_dit_1_stage_loop1:
        loopstart0
            [ ; 01
            MOVE2.2F (r3),d5      ; d5=Br:Bi
            move.l (r15)+,r3      ; digit_reversed_address(i+3)
            ]
            [ ; 02
            sod2aaii d1,d0,d2     ; d2=(Ar+Br),(Ai+Bi)
            sod2ssii d1,d0,d6     ; d3=(Ar-Br),(Ai-Bi)
            sod2aaii d5,d4,d3     ; d6=(Ar+Br),(Ai+Bi)
            sod2ssii d5,d4,d7     ; d7=(Ar-Br),(Ai-Bi)
            MOVE2.2F (r0),d0      ; d0=Ar:Ai
            move.l (r15)+,r0      ; digit_reversed_address(i+0)
            ]
            [ ; 03
            MOVE2.2F (r1),d1      ; d1=Br:Bi
            move.l (r15)+,r1      ; digit_reversed_address(i+1)
            ]
            [ ; 04
            MOVER2.4F d2:d3,(r8)+     ; out:[Aout_r,Aout_i], out:[Aout_r,Aout_i]
            MOVE2.2F (r2),d4      ; d4=Ar:Ai
            ]
            [ ; 05
            MOVER2.4F d6:d7,(r9)+     ; out:[Bout_r,Bout_i], out:[Bout_r,Bout_i]
            move.l (r15)+,r2      ; digit_reversed_address(i+2)
            ]
        loopend0
        [
        move.l (sp-16),r2     ; num_radix
        move.l (sp-40),r3     ; Load the stage counter
        ]
        [
        move.l r4,d3
        adda #2,r3,r3         ; +2
        ]
        [
        move.l d3,SR
        adda r3,r2            ; num_butterfly[s]
        ]
        [
        move.w (r2),r0             ; num_radix[s]
        move.l r3,(sp-40)          ; Update the stage counter
        ]
        cmpeqa.w #0,r0             ; num_radix[s]=0 or not
        [
        jt radix3_dit_last_stage   ; Go to radix-3 Last stage processing if num_radix[s]=0
        cmpeqa.w #5,r0             ; num_radix[s]=5 or not
        ]
        [
        jt radix5_dit_m_stage      ; Go to radix-5 Middle stage processing if num_radix[s]=5
        cmpeqa.w #1,r0             ; num_radix[s]=1 or not
        ]
        [
        jt radix5_dit_last_stage      ; Go to radix-5 Last stage processing if num_radix[s]=1
        cmpeqa.w #4,r0             ; num_radix[s]=4 or not
        ]
        jt radix4_dit_m_stage     ; Go to radix-4 Middle stage processing if num_radix[s]=4

    ;////////////////////////////////////////////////////////////////////////
    ; Radix-3 Stages:
    ;////////////////////////////////////////////////////////////////////////

    radix3_init:

       ;---------------------------------------------------------
       ; Radix-3 (Middle stage)
       ;---------------------------------------------------------

    radix3_dit_m_stage:
     IF @DEF('FIX_SCALE')
        [
        move.l (sp-40),r3   ; Stage counter
        dosetup0 radix3_dit_m_stage_loop1
        ]
        [
        move.l (sp-20),r2      ; num_butterfly
        move.l (sp-24),r4      ; num_subgroup
        ]
        [
        move.l (sp-44),r5      ; r5 -> Wb
        move.l (sp-28),r12     ; num_radix_offse
        ]
        [
        move.l (sp-8),r0       ; A input pointer
        move.l (sp-12),r9      ; A output pointer
        ]
        [
        dosetup1 radix3_dit_m_stage_loop2
        dosetup2 radix3_dit_m_stage_loop2_2
        ]
        [
        adda r3,r2          ; num_butterfly[s]
        adda r3,r12         ; num_radix_offse[s]
        ]
        [
        move.w (r2),n0      ; num_butterfly[s],The stage twiddle factor's offset
```

```
            move.w (r2),r13    ; num_butterfly[s]
            ]
            [
            move.w (r12),r15   ; num_radix_offse[s]
            move.l (sp-68),r2  ; psiScale
            ]
            [
            adda r3,r4         ; num_subgroup[s]
            move.l (sp-48),r6  ; r6 -> Wc
            ]
            [
            move.w (r4),r14    ; num_subgroup[s]
            move.w #SCALE_RADIX_3,d15
            ]

         ELSE    ;//IF @DEF('FIX_SCALE')

            move.l (sp-40),r3  ; Stage counter

            ;-- Find max value (real & imag) & stage initialization
            [
            move.l (sp-8),r0    ; input base (A)
            move.l (sp-60),r2   ; DFT point
            ]
            [
            move.w (r2),d15     ; DFT point
            ]
            [
            sub #8,d15          ; d15-8
            ]
            [
            asrr #2,d15         ; (d15-8)/4
            ]
            [
            adda #8,r0,r1       ; Input address
            move.w #2,n0
            ]
            [
            move.2l (r0)+n0,d0:d1
            move.2l (r1)+n0,d2:d3
            ]
            [
            abs2 d0,d0   abs2 d1,d1
            abs2 d2,d2   abs2 d3,d3
            move.2l (r0)+n0,d4:d5
            move.2l (r1)+n0,d6:d7
            ]
            [
            abs2 d4,d4   abs2 d5,d5
            abs2 d6,d6   abs2 d7,d7
            doensh0 d15         ; loop count for max search
            ]

            ; Loop for max (real,imag) search
            loopstart0
                [
                max2 d0,d4   max2 d1,d5
```

```
    max2 d2,d6   max2 d3,d7
    move.2l (r0)+n0,d0:d1
    move.2l (r1)+n0,d2:d3
    ]
    [
    abs2 d0,d0   abs2 d1,d1
    abs2 d2,d2   abs2 d3,d3
    ]
loopend0

[
max2 d0,d4   max2 d1,d5
max2 d2,d6   max2 d3,d7
move.l (sp-20),r2      ; num_butterfly
move.l (sp-24),r4      ; num_subgroup
]
[
tfr d6,d0   tfr d7,d1
move.l (sp-44),r5      ; r5 -> Wb
move.l (sp-28),r12     ; num_radix_offse
]
[
max2 d0,d4   max2 d1,d5
dosetup0 radix3_dit_m_stage_loop1
dosetup1 radix3_dit_m_stage_loop2
]
[
tfr d5,d0
move.l (sp-8),r0       ; A input pointer
move.l (sp-12),r9      ; A output pointer
]
[
max2 d0,d4
dosetup2 radix3_dit_m_stage_loop2_2
]
[
sxt.w d4,d0
adda r3,r12        ; num_radix_offse[s]
]
[
extract #16,#16,d4,d4
adda r3,r2         ; num_butterfly[s]
adda r3,r4         ; num_subgroup[s]
]
[
max d0,d4          ; Max real/imag
move.l (sp-48),r6  ; r6 -> Wc
move.w (r12),r15   ; num_radix_offse[s]
]
[
clb d4,d0          ; count reading bit
move.w (r2),r13    ; num_butterfly[s]
move.w (r4),r14    ; num_subgroup[s]
]
[
clr d14
neg d0             ; negates
```

```
            move.l (sp-68),r2  ; psiScale
            ]
            [
            add #1,d14         ; d14 = 1
            sub #16,d0         ; limit 16-bit
            move.w #S34,d2     ; d2 = 2
            move.w #2,d15      ; scale down = 2;
            ]
            [
            cmpgt.w #S33,d0 ; Check if siNorm > S33
            move.w #S33,d1
            ]
            [
            ift clr d15         ; scale down = 0      if siNorm > S33+1
            ifa
            cmpeq.w #S33,d0 ; Check if siNorm == S33
            ]
            [
            tfrt d14,d15        ; scale down = 1      if siNorm == S33+1
            tfra r13,n0         ; The stage twiddle factor's offset
            ]
         ENDIF    ;// IF @DEF('FIX_SCALE')
            ; For next stage input/output
            [
            move.l r9,(sp-8)    ; base_address at the next stage
            move.l r0,(sp-12)   ; out_address  at the next stage
            ]
            [
            move.l d15,(sp-64)  ; Set norm to stack
            move.w d15,(r2)+    ; psiScale <- norm
            ]
            [
            move.l r2,(sp-68)   ; psiScale
            move.l d15,r11
            ]
            [
            tfra r9,r8            ; A output
            move.l SR,d4
            ]
            [
            adda r15,r9             ; B output pointer (long)
            move.l d4,r3
            ]
            [
            tfra r9,r10             ; B output
            move.f #$4000,d7         ; M2_3 = (1/2):0
            ]
            [
            tfra r13,r12
            bmset #$1000,SR.L    ; W20=1
            ]
            [
            adda r15,r10        ; C output pointer (long)
            bmclr #$0080,SR.L    ; SM2=0
            ]
            [
            asra r12
```

```
    bmclr #$0030,SR.L    ; SCM=00
    ]
    ;--- Start Radix-3 (Middle stage)
    [
    move.l #radix3parameter,r15  ; Point to the radix-3 parameter array
    cmpeqa.w #1,r11
    ]
    [
    ift bmset #$0010,SR.L    ; SCM=01
    ifa
    asra r14
    ]
    [
    doen0 r14
    cmpeqa.w #2,r11
    ]
    [
    ift bmset #$0030,SR.L    ; SCM=11
    ifa
    move.2f (r0)+,d4:d5      ; IAre:IAim=Ar:Ai
    ]
    [
    bmtsts #$0001,r13.l      ; if num_butterfly[s] is odd, T=1
    tfra r12,r13
    ]
    [
    suba #1,r13
    move.l (r0)+,d2          ; v_IB1=Br:Bi
    ]
    ; Intra loop
    FALIGN
radix3_dit_m_stage_loop1:    ; --> Butterflies/Subgroup
    loopstart0
        [
        doen1 r12
        move.l (r5)+n0,d9        ; v_WB1=Wbr:Wbi
        ]
        FALIGN
radix3_dit_m_stage_loop2:        ; --> Subgroup
        loopstart1
            [ ;01
            MPYRE d2,d9,d12        ; M1_1 = Br' = (Br*Wbr - Bi*Wbi)
            MPYIM d2,d9,d13        ; M1_2 = Bi' = (Br*Wbi + Bi*Wbr)
            move.l (r0)+,d8        ; v_IC1=Cr:Ci
            move.l (r6),d10        ; v_WC1=Wcr:Wci
            ]
            [ ;02
            mac2assar d8,d10,d12.H,d6   ; M2_1 = (Br'+Cr'):(Br'-Cr')
            mac2aassi d8,d10,d13.H,d13  ; M2_2 = (Bi'+Ci'):(Bi'-Ci')
            move.f (r15),d10            ; M3_4 = (sqrt(3)/2):(0)
            move.2f (r0)+,d14:d15       ; IAre:IAim=Ar:Ai
            ]
            ;stall
            [ ;04
           sod2aaii d4,d6,d0           ; Aout_r = Ar + M2_1_H    // Aout_r = Ar'+Br'+Cr'
           sod2aaii d5,d13,d1          ; Aout_i = Ai + M2_2_H    // Aout_i = Ai'+Bi'+Ci'
         mac -d6,d7,d4             ; M3_1_H = Ar - M2_1_H*1/2     // Ar' - (Br'+Cr')*1/2
```

```
               mac -d13,d7,d5             ; M3_2_H = Ai - M2_2_H*1/2       // Ai' - (Bi'+Ci')*1/2
                 move.l d13,d8
                 move.l (r0)+,d2               ; v_IB1=Br:Bi
                 ]
                 [ ;05
              mac2aassi d8,d10,d4.H,d4    ; Bout_r = Ar' - (Br'+Cr')*1/2 + (Bi'-Ci')*sqrt(3)/2
:Cout_r = Ar' - (Br'+Cr')*1/2 - (Bi'-Ci')*sqrt(3)/2
                 MPYRE d2,d9,d12               ; M1_1 = Br' = (Br*Wbr - Bi*Wbi)
                 MPYIM d2,d9,d13               ; M1_2 = Bi' = (Br*Wbi + Bi*Wbr)
                 move.l (r0)+,d8               ; v_IC1=Cr:Ci
                 move.l (r6),d10               ; v_WC1=Wcr:Wci
                 ]
                 [ ;06
                 mac2assar d8,d10,d12.H,d12   ; M2_1 = (Br'+Cr'):(Br'-Cr')
                 mac2aassi d8,d10,d13.H,d13   ; M2_2 = (Bi'+Ci'):(Bi'-Ci')
                 move.f (r15),d10              ; M3_4 = (sqrt(3)/2):(0)
                 move.l d6,d8
                 ]
                 ;stall
                 [ ;08
                sod2aaii d14,d12,d2          ; Aout_r = Ar + M2_1_H     // Aout_r = Ar'+Br'+Cr'
                sod2aaii d15,d13,d3          ; Aout_i = Ai + M2_2_H     // Aout_i = Ai'+Bi'+Ci'
               mac -d12,d7,d14            ; M3_1_H = Ar - M2_1_H*1/2       // Ar' - (Br'+Cr')*1/2
               mac2ssaai d8,d10,d5.H,d5    ; Bout_i=Ai' - (Bi'+Ci')*1/2 - (Br'-Cr')*sqrt(3)/2
:Cout_i=Ai' - (Bi'+Ci')*1/2 + (Br'-Cr')*sqrt(3)/2
                 move.l d13,d8
                 ]
                 [ ;09
                 mac2aassi d8,d10,d14.H,d14   ; Bout_r:Cout_r
               mac -d13,d7,d15            ; M3_2_H = Ai - M2_2_H*1/2       // Ai' - (Bi'+Ci')*1/2
                 move.l d12,d8
                 move.2f (r0)+,d12:d13         ; IAre:IAim=Ar:Ai
                 ]
                 [ ;10
                 mac2ssaai d8,d10,d15.H,d15   ; Bout_i:Cout_i
                 MOVERH.4F d0:d1:d2:d3,(r8)+  ; out:[Aout_r,Aout_i]
                 move.l (r0)+,d2               ; v_IB1=Br:Bi
                 ]
                 [ ;11
                 tfr d12,d4
                 tfr d13,d5
                 MOVERH.4F d4:d5:d14:d15,(r9)+     ; out:[Bout_r,Bout_i],[Bout_r,Bout_i]
                 MOVERL.4F d4:d5:d14:d15,(r10)+    ; out:[Cout_r,Cout_i],[Cout_r,Cout_i]
                 ]
              loopend1
             [
             iff addl2a n0,r6
             iff move.l (r5)+n0,d9   ; v_WB1=Wbr:Wbi
             ]
             [
             MPYRE d2,d9,d12         ; M1_1 = Br' = (Br*Wbr - Bi*Wbi)
             MPYIM d2,d9,d13         ; M1_2 = Bi' = (Br*Wbi + Bi*Wbr)
             move.l (r0)+,d8         ; v_IC1=Cr:Ci
             move.l (r6),d10         ; v_WC1=Wcr:Wci
             ]
             [
             mac2assar d8,d10,d12.H,d6  ; M2_1 = (Br'+Cr'):(Br'-Cr')
```

```
            mac2aassi d8,d10,d13.H,d13 ; M2_2 = (Bi'+Ci'):(Bi'-Ci')
            move.f    (r15),d10        ; M3_4 = (sqrt(3)/2):(0)
            move.2f (r0)+,d14:d15      ; IAre:IAim=Ar:Ai
            ]
            [
            ift addl2a n0,r6
            ift move.l (r5),d9        ; v_WB1=Wbr:Wbi
            ;stall
            ]
            [
            sod2aaii d4,d6,d0         ; Aout_r = Ar + M2_1_H    // Aout_r = Ar'+Br'+Cr'
            sod2aaii d5,d13,d1        ; Aout_i = Ai + M2_2_H    // Aout_i = Ai'+Bi'+Ci'
           mac -d6,d7,d4             ; M3_1_H = Ar - M2_1_H*1/2        // Ar' - (Br'+Cr')*1/2
           mac -d13,d7,d5            ; M3_2_H = Ai - M2_2_H*1/2        // Ai' - (Bi'+Ci')*1/2
            move.l d13,d8
            move.l (r0)+,d2           ; v_IB1=Br:Bi
            ]
            [
            mac2aassi d8,d10,d4.H,d4  ; Bout_r:Cout_r
            MPYRE d2,d9,d12           ; M1_1 = Br' = (Br*Wbr - Bi*Wbi)
            MPYIM d2,d9,d13           ; M1_2 = Bi' = (Br*Wbi + Bi*Wbr)
            move.l (r0)+,d8           ; v_IC1=Cr:Ci
            move.l (r6),d10           ; v_WC1=Wcr:Wci
            ]
            [
            mac2assar d8,d10,d12.H,d12    ; M2_1 = (Br'+Cr'):(Br'-Cr')
            mac2aassi d8,d10,d13.H,d13    ; M2_2 = (Bi'+Ci'):(Bi'-Ci')
            move.f (r15),d10              ; M3_4 = (sqrt(3)/2):(0)
            move.l d6,d8
            ]
            [
            ift addl2a n0,r5
            ift doen2 r12
            ifa
            sod2aaii d14,d12,d2         ; Aout_r = Ar + M2_1_H    // Aout_r = Ar'+Br'+Cr'
            sod2aaii d15,d13,d3         ; Aout_i = Ai + M2_2_H    // Aout_i = Ai'+Bi'+Ci'
            ;stall
            ]
            [
            iff doen2 r13
            ifa
           mac -d12,d7,d14             ; M3_1_H = Ar - M2_1_H*1/2        // Ar' - (Br'+Cr')*1/2
            mac2ssaai d8,d10,d5.H,d5    ; Bout_i:Cout_i
            move.l d13,d8
            ]
            [
            mac2aassi d8,d10,d14.H,d14 ; Bout_r:Cout_r
           mac -d13,d7,d15             ; M3_2_H = Ai - M2_2_H*1/2        // Ai' - (Bi'+Ci')*1/2
            move.l d12,d8
            move.2f (r0)+,d12:d13       ; IAre:IAim=Ar:Ai
            ]
            [
            mac2ssaai d8,d10,d15.H,d15  ; Bout_i:Cout_i
            moverh.4f d0:d1:d2:d3,(r8)+ ; out:[Aout_r,Aout_i]
            move.l (r0)+,d2             ; v_IB1=Br:Bi
            ]
    radix3_dit_m_stage_loop2_2:         ; --> Subgroup
```

```
loopstart2
    [ ;01
    tfr d12,d4
    tfr d13,d5
    MOVERH.4F d4:d5:d14:d15,(r9)+         ; out:[Bout_r,Bout_i],[Bout_r,Bout_i]
    MOVERL.4F d4:d5:d14:d15,(r10)+        ; out:[Cout_r,Cout_i],[Cout_r,Cout_i]
    ]
    [ ;02
    MPYRE d2,d9,d12         ; M1_1 = Br' = (Br*Wbr - Bi*Wbi)
    MPYIM d2,d9,d13         ; M1_2 = Bi' = (Br*Wbi + Bi*Wbr)
    move.l (r0)+,d8         ; v_IC1=Cr:Ci
    move.l (r6),d10         ; v_WC1=Wcr:Wci
    ]
    [ ;03
    mac2assar d8,d10,d12.H,d6     ; M2_1 = (Br'+Cr'):(Br'-Cr')
    mac2aassi d8,d10,d13.H,d13    ; M2_2 = (Bi'+Ci'):(Bi'-Ci')
    move.f (r15),d10              ; M3_4 = (sqrt(3)/2):(0)
    move.2f (r0)+,d14:d15         ; IAre:IAim=Ar:Ai
    ]
    ;STALL
    [ ;05
    sod2aaii d4,d6,d0         ; Aout_r = Ar + M2_1_H    // Aout_r = Ar'+Br'+Cr'
    sod2aaii d5,d13,d1        ; Aout_i = Ai + M2_2_H    // Aout_i = Ai'+Bi'+Ci'
  mac -d6,d7,d4            ; M3_1_H = Ar - M2_1_H*1/2        // Ar' - (Br'+Cr')*1/2
  mac -d13,d7,d5           ; M3_2_H = Ai - M2_2_H*1/2        // Ai' - (Bi'+Ci')*1/2
    move.l d13,d8
    move.l (r0)+,d2           ; v_IB1=Br:Bi
    ]
    [ ;06
    mac2aassi d8,d10,d4.H,d4  ; Bout_r:Cout_r
    MPYRE d2,d9,d12           ; M1_1 = Br' = (Br*Wbr - Bi*Wbi)
    MPYIM d2,d9,d13           ; M1_2 = Bi' = (Br*Wbi + Bi*Wbr)
    move.l (r0)+,d8           ; v_IC1=Cr:Ci
    move.l (r6),d10           ; v_WC1=Wcr:Wci
    ]
    [ ;07
    mac2assar d8,d10,d12.H,d12    ; M2_1 = (Br'+Cr'):(Br'-Cr')
    mac2aassi d8,d10,d13.H,d13    ; M2_2 = (Bi'+Ci'):(Bi'-Ci')
    move.f (r15),d10              ; M3_4 = (sqrt(3)/2):(0)
    move.l d6,d8
    ]
    ;stall
    [ ;09
    sod2aaii d14,d12,d2        ; Aout_r = Ar + M2_1_H    // Aout_r = Ar'+Br'+Cr'
    sod2aaii d15,d13,d3        ; Aout_i = Ai + M2_2_H    // Aout_i = Ai'+Bi'+Ci'
  mac -d12,d7,d14          ; M3_1_H = Ar - M2_1_H*1/2       // Ar' - (Br'+Cr')*1/2
    mac2ssaai d8,d10,d5.H,d5    ; Bout_i:Cout_i
    move.l d13,d8
    ]
    [ ;10
    mac2aassi d8,d10,d14.H,d14 ; Bout_r:Cout_r
  mac -d13,d7,d15          ; M3_2_H = Ai - M2_2_H*1/2       // Ai' - (Bi'+Ci')*1/2
    move.l d12,d8
    move.2f (r0)+,d12:d13      ; IAre:IAim=Ar:Ai
    ]
    [ ;11
    mac2ssaai d8,d10,d15.H,d15  ; Bout_i:Cout_i
```

```
            MOVERH.4F d0:d1:d2:d3,(r8)+ ; out:[Aout_r,Aout_i]
            move.l (r0)+,d2             ; v_IB1=Br:Bi
            ]
         loopend2
        [
        tfr d12,d4
        tfr d13,d5
        MOVERH.4F d4:d5:d14:d15,(r9)+  ; out:[Bout_r,Bout_i],[Bout_r,Bout_i]
        MOVERL.4F d4:d5:d14:d15,(r10)+ ; out:[Cout_r,Cout_i],[Cout_r,Cout_i]
        ]
        addl2a n0,r6
    loopend0
    [
    move.l r3,d3
    move.l (sp-40),r3    ; Load the stage counter
    ]
    [
    move.l d3,SR
    move.l (sp-16),r2    ; num_radix
    ]
    adda #2,r3,r3        ; +2
    [
    move.l r3,(sp-40)    ; Update the stage counter
    adda r3,r2           ; num_butterfly[s]
    ]
    move.w (r2),r0             ; num_radix[s]
    cmpeqa.w #3,r0           ; num_radix[s]=3 or not
    [
   jt radix3_dit_m_stage      ; Go to radix-3 Middle stage processing if num_radix[s]=3
    cmpeqa.w #5,r0             ; num_radix[s]=5 or not
    ]
    [
    jt radix5_dit_m_stage      ; Go to radix-5 Middle stage processing if num_radix[s]=5
    cmpeqa.w #1,r0             ; num_radix[s]=1 or not
    ]
    jt radix5_dit_last_stage      ; Go to radix-5 Last stage processing if num_radix[s]=1

   ;---------------------------------------------------------
   ; Radix-3 (Last stage)
   ;---------------------------------------------------------
radix3_dit_last_stage:

 IF @DEF('FIX_SCALE')
    [
    move.l (sp-40),r3          ; Stage counter
    move.l (sp-28),r2          ; num_radix_offse
    ]
    [
    move.l (sp-24),r4          ; num_subgroup
    move.l (sp-44),r12         ; r12 -> Wb
    ]
    [
    move.l (sp-12),r8          ; A output pointer
    move.l (sp-48),r13         ; r13 -> Wc
    ]
    [
    clr d15
```

```
            move.l (sp-8),r0         ; A input pointer
            move.l (sp-68),r1        ; psiScale
            ]
            [
            add #SCALE_RADIX_3,d15
            adda r3,r2         ; num_radix_offse[s]
            dosetup0 radix3_dit_last_stage_loop1
            ]
            [
            move.w (r2),r15    ; num_radix_offse[s]
            adda r3,r4         ; num_subgroup[s]
            ]
            [
            move.w (r4),r14    ; num_subgroup[s]
            tfra r8,r9            ; A output
            ]
            [
            move.l d15,(sp-64)    ; Set norm to stack
            adda r15,r9            ; B output pointer (long)
            ]
            [
            tfra r9,r10            ; B output
            move.w d15,(r1)+    ; psiScale <- norm
            ]
            [
            adda r15,r10        ; C output pointer (long)
            move.l #radix3parameter,r15  ; Point to the radix-3 parameter array
            ]
         ELSE    ;//IF @DEF('FIX_SCALE')
            ;-- Find max value (real & imag) & stage initialization
            [
            move.l (sp-8),r0    ; input base (A)
            move.l (sp-60),r2   ; DFT point
            ]
            [
            move.w (r2),d15     ; DFT point
            ]
            [
            sub #8,d15          ; d15-8
            ]
            [
            asrr #2,d15         ; (d15-8)/4
            ]
            [
            adda #8,r0,r1       ; Input address
            move.w #2,n0
            ]
            [
            move.2l (r0)+n0,d0:d1
            move.2l (r1)+n0,d2:d3
            ]
            [
            abs2 d0,d0   abs2 d1,d1
            abs2 d2,d2   abs2 d3,d3
            move.2l (r0)+n0,d4:d5
            move.2l (r1)+n0,d6:d7
            ]
```

```
[
abs2 d4,d4   abs2 d5,d5
abs2 d6,d6   abs2 d7,d7
doensh0 d15         ; loop count for max search
]

; Loop for max (real,imag) search
loopstart0
    [
    max2 d0,d4   max2 d1,d5
    max2 d2,d6   max2 d3,d7
    move.2l (r0)+n0,d0:d1
    move.2l (r1)+n0,d2:d3
    ]
    [
    abs2 d0,d0   abs2 d1,d1
    abs2 d2,d2   abs2 d3,d3
    ]
loopend0

[
max2 d0,d4   max2 d1,d5
max2 d2,d6   max2 d3,d7
move.l (sp-20),r2      ; num_butterfly
move.l (sp-24),r4      ; num_subgroup
]
[
tfr d6,d0   tfr d7,d1
move.l (sp-40),r3      ; Stage counter
move.l (sp-28),r2      ; num_radix_offse
]
[
max2 d0,d4   max2 d1,d5
move.l (sp-44),r12        ; r12 -> Wb
move.l (sp-48),r13        ; r13 -> Wc
]
[
tfr d5,d0
move.l (sp-8),r0       ; A input pointer
move.l (sp-12),r8      ; A output pointer
]
[
max2 d0,d4
move.l (sp-68),r1   ; psiScale
]
[
sxt.w d4,d0
]
[
extract #16,#16,d4,d4
]
[
max d0,d4           ; Max real/imag
adda r3,r2          ; num_radix_offse[s]
adda r3,r4          ; num_subgroup[s]
]
[
        clb d4,d0           ; count reading bit
        dosetup0 radix3_dit_last_stage_loop1
        move.w (r2),r15     ; num_radix_offse[s]
        ]
        [
        neg d0              ; negates
        move.w (r4),r14     ; num_subgroup[s]
        tfra r8,r9          ; A output
        ]
        [
        sub #16,d0          ; limit 16-bit
        move.w #S32,d2
        adda r15,r9         ; B output pointer (long)
        ]
        [
        clr d11
        cmpgt.w #S31,d0     ; Check if norm > S or norm <= S
        move.w #S31,d1
        tfra r9,r10         ; B output
        ]
        [
        iff sub d0,d1,d15   ; scale down = S - d0 if norm <=S
        ift sub d0,d2,d15   ; scale up   = S - d0 if norm > S
        ]
        [
        max d11,d15             ; shift shouldn't be to the left
        adda r15,r10          ; C output pointer (long)
        move.l #radix3parameter,r15  ; Point to the radix-3 parameter array
        ]
        [
        move.l d15,(sp-64)     ; Set norm to stack
        move.w d15,(r1)+     ; psiScale <- norm
        ]
     ENDIF    ;// IF @DEF('FIX_SCALE')
        [
        move.l r1,(sp-68)     ; psiScale
        move.l d15,r1
        ]

        ;--- Start Radix-3 (Last stage)

        [
        asra r14
        move.l SR,d3
        ]
        [
        move.l d3,r3
        bmset #$1000,SR.L        ; W20=1
        ]
        [
        doen0 r14                ; set the loop
        ]
        ; Inner loop
        [
        move.2f (r0)+,d4:d5        ; IAre:IAim=Ar:Ai
        bmclr #$0080,SR.L        ; SM2=0
        ]
        [
        move.w (r4),r14              ; num_subgroup[s]
        bmclr #$0030,SR.L          ; SCM=00
        ]
        [
        cmpeqa.w #1,r1
        move.l (r0)+,d12           ; v_IB1=Br:Bi
        ]
        [
        ift bmset #$0010,SR.L      ; SCM=01
        ifa
        move.2l (r12)+,d8:d9       ; v_WB1=Wbr:Wbi
        ]
        [
        MPYRE d12,d8,d12           ; M1_1 = Br' = (Br*Wbr - Bi*Wbi)
        MPYIM d12,d8,d13           ; M1_2 = Bi' = (Br*Wbi + Bi*Wbr)
        move.l (r0)+,d8              ; v_IC1=Cr:Ci
        move.l (r13)+,d10          ; v_WC1=Wcr:Wci
        ]
        [
        mac2assar d8,d10,d12.H,d6      ; M2_1 = (Br'+Cr'):(Br'-Cr')
        move.f #$4000,d7           ; M2_3 = (1/2):0
        cmpeqa.w #2,r1
        ]
        [
        ift bmset #$0030,SR.L      ; SCM=11
        ifa
        mac2aassi d8,d10,d13.H,d13     ; M2_2 = (Bi'+Ci'):(Bi'-Ci')
        move.f (r15),d10           ; M3_4 = (sqrt(3)/2):(0)
        ]
        [
        bmtsts #$0001,r14.l          ; if num_subgroup[s] is odd, T=1
        move.2f (r0)+,d14:d15      ; IAre:IAim=Ar:Ai
        ]
        FALIGN
    radix3_dit_last_stage_loop1:
        loopstart0
            [ ;01
            sod2aaii d4,d6,d0          ; Aout_r = Ar + M2_1_H    // Aout_r = Ar'+Br'+Cr'
            sod2aaii d5,d13,d1         ; Aout_i = Ai + M2_2_H    // Aout_i = Ai'+Bi'+Ci'
            mac -d6,d7,d4              ; M3_1_H = Ar - M2_1_H*1/2        // Ar' - (Br'+Cr')*1/2
            mac -d13,d7,d5             ; M3_2_H = Ai - M2_2_H*1/2        // Ai' - (Bi'+Ci')*1/2
            move.l d13,d8
            move.l (r0)+,d12           ; v_IB1=Br:Bi
            ]
            [ ;02
            mac2aassi d8,d10,d4.H,d4 ; Bout_r:Cout_r
            MPYRE d12,d9,d12           ; M1_1 = Br' = (Br*Wbr - Bi*Wbi)
            MPYIM d12,d9,d13           ; M1_2 = Bi' = (Br*Wbi + Bi*Wbr)
            move.l (r0)+,d8            ; v_IC1=Cr:Ci
            move.2l (r13)+,d10:d11     ; v_WC1=Wcr:Wci
            ]
            [ ;03
            mac2assar d8,d10,d12.H,d12     ; M2_1 = (Br'+Cr'):(Br'-Cr')
            mac2aassi d8,d10,d13.H,d13     ; M2_2 = (Bi'+Ci'):(Bi'-Ci')
            tfr d6,d9
            move.f (r15),d10           ; M3_4 = (sqrt(3)/2):(0)
            ]
            [ ;04
            sod2aaii d14,d12,d2         ; Aout_r = Ar + M2_1_H    // Aout_r = Ar'+Br'+Cr'
            sod2aaii d15,d13,d3         ; Aout_i = Ai + M2_2_H    // Aout_i = Ai'+Bi'+Ci'
            move.l d13,d8
            ]
            [ ;05
            tfr d5,d0
           mac -d12,d7,d14            ; M3_1_H = Ar - M2_1_H*1/2        // Ar' - (Br'+Cr')*1/2
           mac -d13,d7,d15            ; M3_2_H = Ai - M2_2_H*1/2        // Ai' - (Bi'+Ci')*1/2
            MOVERH.4F d0:d1:d2:d3,(r8)+     ; out:[Aout_r,Aout_i]
            move.2f (r0)+,d2:d3         ; IAre:IAim=Ar:Ai
            ]
            [ ;06
            mac2ssaai d9,d10,d0.H,d5    ; Bout_i:Cout_i
            mac2aassi d8,d10,d14.H,d14 ; Bout_r:Cout_r
            move.l d12,d8
            move.l (r0)+,d12            ; v_IB1=Br:Bi
            ]
            [ ;07
            mac2ssaai d8,d10,d15.H,d15 ; Bout_i:Cout_i
            tfr d11,d10
            move.l (r12)+,d9            ; v_WB1=Wbr:Wbi
            move.l (r0)+,d8             ; v_IC1=Cr:Ci
            ]
            [ ;08
            MPYRE d12,d9,d12          ; M1_1 = Br' = (Br*Wbr - Bi*Wbi)
            MPYIM d12,d9,d13          ; M1_2 = Bi' = (Br*Wbi + Bi*Wbr)
            move.l (r12)+,d9          ; v_WB1=Wbr:Wbi
            ]
            [ ;09
            tfr d2,d4
            tfr d3,d5
            mac2assar d8,d10,d12.H,d6      ; M2_1 = (Br'+Cr'):(Br'-Cr')
            mac2aassi d8,d10,d13.H,d13     ; M2_2 = (Bi'+Ci'):(Bi'-Ci')
            MOVERH.4F d4:d5:d14:d15,(r9)+ ; out:[Bout_r,Bout_i],[Bout_r,Bout_i]
            MOVERL.4F d4:d5:d14:d15,(r10)+; out:[Cout_r,Cout_i],[Cout_r,Cout_i]
            ]
            [ ;10
            move.f (r15),d10          ; M3_4 = (sqrt(3)/2):(0)
            move.2f (r0)+,d14:d15     ; IAre:IAim=Ar:Ai
            ;stall
            ]
        loopend0
        [
        sod2aaii d4,d12,d0          ; Aout_r = Ar + M2_1_H    // Aout_r = Ar'+Br'+Cr'
        sod2aaii d5,d13,d1          ; Aout_i = Ai + M2_2_H    // Aout_i = Ai'+Bi'+Ci'
        mac -d12,d7,d4              ; M3_1_H = Ar - M2_1_H*1/2        // Ar' - (Br'+Cr')*1/2
        mac -d13,d7,d5              ; M3_2_H = Ai - M2_2_H*1/2        // Ai' - (Bi'+Ci')*1/2
        move.l d13,d8
        ]
        [
        mac2aassi d8,d10,d4.H,d4     ; Bout_r:Cout_r
        move.l d12,d8
        ]
        [
        ift     MOVERH.4F d0:d1:d2:d3,(r8)+     ; out:[Aout_r,Aout_i]
        ifa
        mac2ssaai d8,d10,d5.H,d5     ; Bout_i:Cout_i
        move.l r3,d6
        ]
        [
        ift    MOVERH.4F d4:d5:d14:d15,(r9)+     ; out:[Bout_r,Bout_i],[Bout_r,Bout_i]
        ift    MOVERL.4F d4:d5:d14:d15,(r10)+     ; out:[Cout_r,Cout_i],[Cout_r,Cout_i]
        ]
        [
        jmp exit_mixed_radix
        move.l d6,SR
        ]

    ;////////////////////////////////////////////////////////////////////////
    ; Radix-4 Stages:
    ;////////////////////////////////////////////////////////////////////////

    radix4_init:

       ;---------------------------------------------------------
       ; Radix-4 (1st stage)
       ;---------------------------------------------------------

    radix4_dit_1_stage:

     IF @DEF('FIX_SCALE')
        [
        clr d15
        move.l (sp-20),r15      ; num_butterfly
        move.l (sp-28),r4       ; num_radix_offset
        ]
        [
        add #SCALE_RADIX_4,d15
        move.l (sp-8),r0        ; A input pointer
        move.l (sp-12),r8       ; A output pointer
        ]
        [
        dosetup0 radix4_dit_1_stage_loop1
        move.l (sp-68),r2       ; psiScale
        ]
        [
        move.l d15,(sp-64)      ; Set norm to stack
        move.l d15,r5           ; psiScale <- norm
        ]
        [
        move.w (r4),r14         ; num_radix_offset[s]
        move.w (r15),r15        ; num_butterfly[s]
        ]
        ; For next stage input/output
        [
        move.l r8,(sp-8)        ; base_address at the next stage
        move.l r0,(sp-12)       ; out_address  at the next stage
        ]
        [
        tfra r8,r9              ; A output
        move.w d15,(r2)+        ; psiScale <- norm
        ]
```

```
         [
         move.l r2,(sp-68)      ; psiScale
         adda r14,r9            ; B output
         ]
         [
         tfra r9,r10            ; B output
         suba #1,r15
         ]
         [
         adda r14,r10        ; C output
         doen0 r15           ; loop0 = num_butterfly[s]
         ]
         [
         tfra r10,r11        ; C output
         move.l #(_psiDigitReversedAddress_absolut_address),r15
         ]
      ELSE    ;//IF @DEF('FIX_SCALE')
         ;-- Find max value (real & imag) & stage initialization
         [
         move.l (sp-8),r0    ; input base (A)
         move.l (sp-60),r2   ; DFT point
         ]
         move.w #2,n0
         [
         move.w (r2),d15     ; DFT point
         adda #8,r0,r1       ; Input address
         ]
         [
         sub #8,d15          ; d15-8
         move.2l (r0)+n0,d0:d1
         move.2l (r1)+n0,d2:d3
         ]
         [
         asrr #2,d15         ; (d15-8)/4
         move.l (sp-20),r2   ; num_butterfly
         move.l (sp-24),r4   ; num_subgroup
         ]
         [
         abs2 d0,d0   abs2 d1,d1
         abs2 d2,d2   abs2 d3,d3
         move.2l (r0)+n0,d4:d5
         move.2l (r1)+n0,d6:d7
         ]
         [
         abs2 d4,d4   abs2 d5,d5
         abs2 d6,d6   abs2 d7,d7
         doensh0 d15         ; loop count for max search
         ]

         ; Loop for max (real,imag) search
         loopstart0
             [
             max2 d0,d4   max2 d1,d5
             max2 d2,d6   max2 d3,d7
             move.2l (r0)+n0,d0:d1
             move.2l (r1)+n0,d2:d3
             ]
```

```
    [
    abs2 d0,d0   abs2 d1,d1
    abs2 d2,d2   abs2 d3,d3
    ]
loopend0
[
max2 d0,d4   max2 d1,d5
max2 d2,d6   max2 d3,d7
]
[
tfr d6,d0   tfr d7,d1
dosetup0 radix4_dit_1_stage_loop1
]
[
max2 d0,d4   max2 d1,d5
move.l (sp-8),r0        ; A input pointer
move.l (sp-12),r9       ; A output pointer
]
[
tfr d5,d0
]
[
max2 d0,d4
move.w (r4),r5          ; num_subgroup[s]
move.l (sp-28),r4       ; num_radix_offset
]
[
sxt.w d4,d0
move.w (r2),r15         ; num_butterfly[s]
move.w #S44,d2
]
[
extract #16,#16,d4,d4
move.l (sp-68),r2     ; psiScale
]
[
max d0,d4             ; Max real/imag
move.w #S43,d1
]
[
clb d4,d0           ; count leading bit
]
[
neg d0              ; negates
move.w (r4),r14     ; num_radix_offset[s]
suba #1,r15
]
[
clr d14
sub #16,d0          ; limit 16-bit
suba #1,r5          ; r5 - 1
tfra r9,r8          ; A output
]
[
add #1,d14
cmpgt.w #S43,d0 ; Check if siNorm > S43
move.w #2,d15        ; scale down = 2
```

```
          adda r14,r9          ; B output
          ]
          [
          ift clr d15          ; scale down = 0      if siNorm > S43+1
          ifa
          cmpeq.w #S43,d0 ; Check if siNorm = S43
          tfra r9,r10          ; B output
          ]
          [
          tfrt d14,d15         ; scale down = 1      if siNorm = S43
          adda r14,r10         ; C output
          ]
          [
          move.w d15,(r2)+     ; psiScale <- norm
          tfra r10,r11         ; C output
          ]
          [
          move.l d15,(sp-64)   ; Set norm to stack
          move.l r2,(sp-68)    ; psiScale
          ]
          [
          doen0 r15            ; loop0 = num_butterfly[s]
          move.l d15,r5        ; psiScale <- norm
          ]
          ; For next stage input/output
          [
          move.l r8,(sp-8)     ; base_address at the next stage
          move.l r0,(sp-12)    ; out_address  at the next stage
          ]
          move.l #(_psiDigitReversedAddress_absolut_address),r15
       ENDIF    ;// IF @DEF('FIX_SCALE')
          [
          adda r14,r11         ; D output

          ;--- Start Radix-4 (1st stage)

          move.l (r15)+,r0     ; digit_reversed_address(i+0)
          ]
          [
          move.l SR,d3
          move.l (r15)+,r1     ; digit_reversed_address(i+1)
          ]
          [
          move.l d3,r4
          move.l (r15)+,r2     ; digit_reversed_address(i+2)
          ]
          [
          bmset #$1000,SR.L    ; W20=1
          move.l (r15)+,r3     ; digit_reversed_address(i+3)
          ]
          [
          bmclr #$0080,SR.L    ; SM2=0
          MOVE2.2F (r0),d0     ; d0=Ar:Ai
          ]
          [
          bmclr #$0030,SR.L    ; SCM=00
          move.l (r15)+,r0     ; digit_reversed_address(i+0)
```

```
        ]
        [
        cmpeqa.w #1,r5
        MOVE2.2F (r1),d2    ; d2=Br:Bi
        ]
        [
        ift bmset #$0010,SR.L    ; SCM=01
        ifa
        move.l (r15)+,r1    ; digit_reversed_address(i+1)
        ]
        [
        cmpeqa.w #2,r5
        MOVE2.2F (r2),d1    ; d1=Cr:Ci
        ]
        [
        ift bmset #$0030,SR.L    ; SCM=11
        ifa
        move.l (r15)+,r2    ; digit_reversed_address(i+2)
        ]
        [
        MOVE2.2F (r3),d3    ; d3=Dr:Di
        move.l (r15)+,r3    ; digit_reversed_address(i+3)
        ]
        [
        sod2aaii d1,d0,d4    ; d0=(Ar+Cr),(Ai+Ci)
        sod2ssii d1,d0,d5    ; d1=(Ar-Cr),(Ai-Ci)
        sod2aaii d3,d2,d6    ; d2=(Br+Dr),(Bi+Di)
        sod2ssii d3,d2,d7    ; d3=(Br-Dr),(Bi-Di)
        MOVE2.2F (r0),d0     ; d0=Ar:Ai
        ]
        FALIGN
    radix4_dit_1_stage_loop1:
       loopstart0
            [
            MOVE2.2F (r1),d2    ; d2=Br:Bi
            move.l (r15)+,r0    ; digit_reversed_address(i+0)
            ]
            [
            sod2aaii d6,d4,d4   ; d4=(Ar':Ai')=((Ar+Br)+(Cr+Dr),(Ai+Bi)+(Ci+Di))
            sod2saxx d7,d5,d5   ; d5=(Br':Bi')=((Ar-Cr)+(Bi-Di),(Ai-Ci)-(Br-Dr))
            sod2ssii d6,d4,d6   ; d6=(Cr':Ci')=((Ar+Cr)-(Br+Dr),(Ai+Ci)-(Bi+Di))
            sod2asxx d7,d5,d7   ; d7=(Dr':Di')=((Ar-Cr)-(Bi-Di),(Ai-Ci)+(Br-Dr))
            MOVE2.2F (r2),d1    ; d1=Cr:Ci
            move.l (r15)+,r1    ; digit_reversed_address(i+1)
            ]
            [
            MOVER2.2F d4,(r8)+  ; out:[Aout_r,Aout_i], out:[Bout_r,Bout_i]
            move.l (r15)+,r2    ; digit_reversed_address(i+2)
            ]
            [
            MOVER2.2F d5,(r9)+  ; out:[Aout_r,Aout_i], out:[Bout_r,Bout_i]
            MOVE2.2F (r3),d3    ; d3=Dr:Di
            ]
            [
            MOVER2.2F d6,(r10)+ ; out:[Cout_r,Cout_i], out:[Dout_r,Dout_i]
            move.l (r15)+,r3    ; digit_reversed_address(i+3)
            ]
            [
            sod2aaii d1,d0,d4    ; d0=(Ar+Cr),(Ai+Ci)
            sod2ssii d1,d0,d5    ; d1=(Ar-Cr),(Ai-Ci)
            sod2aaii d3,d2,d6    ; d2=(Br+Dr),(Bi+Di)
            sod2ssii d3,d2,d7    ; d3=(Br-Dr),(Bi-Di)
            MOVER2.2F d7,(r11)+ ; out:[Cout_r,Cout_i], out:[Dout_r,Dout_i]
            MOVE2.2F (r0),d0     ; d0=Ar:Ai
            ]
        loopend0
        [
        sod2aaii d6,d4,d4    ; d4=(Ar':Ai')=((Ar+Br)+(Cr+Dr),(Ai+Bi)+(Ci+Di))
        sod2saxx d7,d5,d5    ; d5=(Br':Bi')=((Ar-Cr)+(Bi-Di),(Ai-Ci)-(Br-Dr))
        sod2ssii d6,d4,d6    ; d6=(Cr':Ci')=((Ar+Cr)-(Br+Dr),(Ai+Ci)-(Bi+Di))
        sod2asxx d7,d5,d7    ; d7=(Dr':Di')=((Ar-Cr)-(Bi-Di),(Ai-Ci)+(Br-Dr))
        move.l (sp-16),r2    ; num_radix
        move.l (sp-40),r3    ; Load the stage counter
        ]
        [
        MOVER2.2F d4,(r8)+     ; out:[Aout_r,Aout_i]
        MOVER2.2F d5,(r9)+     ; out:[Bout_r,Bout_i]
        ]
        [
        MOVER2.2F d6,(r10)+     ; out:[Cout_r,Cout_i]
        move.l r4,d3
        ]
        [
        MOVER2.2F d7,(r11)+     ; out:[Dout_r,Dout_i]
        adda #2,r3,r3        ; +2
        ]
        [
        move.l r3,(sp-40)    ; Update the stage counter
        adda r3,r2           ; num_butterfly[s]
        ]
        [
        move.w (r2),r0             ; num_radix[s]
        move.l d3,SR
        ]
        [
        cmpeqa.w #0,r0             ; num_radix[s]=0 or not
        ]
        [
        jt radix3_dit_last_stage   ; Go to radix-3 Last stage processing if num_radix[s]=0
        cmpeqa.w #3,r0             ; num_radix[s]=3 or not
        ]
        [
       jt radix3_dit_m_stage     ; Go to radix-3 Middle stage processing if num_radix[s]=3
        cmpeqa.w #5,r0             ; num_radix[s]=5 or not
        ]
        [
        jt radix5_dit_m_stage      ; Go to radix-5 Middle stage processing if num_radix[s]=5
        cmpeqa.w #1,r0             ; num_radix[s]=1 or not
        ]
        jt radix5_dit_last_stage   ; Go to radix-5 Last stage processing if num_radix[s]=1
        ;----------------------------------------------------------
        ; Radix-4 (Middle stage)
        ;----------------------------------------------------------
    radix4_dit_m_stage:
```

```
IF @DEF('FIX_SCALE')
   [
   clr d15
   move.l (sp-40),r3        ; Stage counter
   dosetup0 radix4_dit_m_stage_loop1    ; radix4 loop1 (outer loop)
   ]
   [
   add #SCALE_RADIX_4,d15
   move.l (sp-20),r2        ; num_butterfly
   move.l (sp-24),r4        ; num_subgroup
   ]
   [
   move.l (sp-44),r5      ; r5 -> Wb
   move.l (sp-48),r6      ; r6 -> Wc
   ]
   [
   move.l (sp-28),r12     ; num_radix_offset
   move.l (sp-52),r7      ; r7 -> Wd
   ]
   [
   dosetup1 radix4_dit_m_stage_loop2_1    ; radix4 loop2_1 (inner loop)
   move.l (sp-8),r0    ; A input pointer
   ]
   [
   adda r3,r2          ; num_butterfly[s]
   adda r3,r4          ; num_subgroup[s]
   ]
   [
   move.w (r2),r13     ; num_butterfly[s]
   move.l (sp-68),r2   ; psiScale
   ]
   [
   move.w (r4),r14     ; num_subgroup[s]
   move.l (sp-12),r9   ; A output pointer
   ]
   [
   dosetup2 radix4_dit_m_stage_loop2_2    ; radix4 loop2_2 (inner loop)
   adda r3,r12         ; num_radix_offset[s]
   ]
   [
   move.w #3,n3        ; output offset for radix-4
   ]
ELSE    ;//IF @DEF('FIX_SCALE')
   [
   move.l (sp-40),r3     ; Stage counter
   ]
   ;-- Find max value (real & imag) & stage initialization
   [
   move.l (sp-8),r0     ; input base (A)
   move.l (sp-60),r2    ; DFT point
   ]
   [
   move.w (r2),d15      ; DFT point
   ]
   [
   sub #8,d15           ; d15-8
   ]
```

```
[
asrr #2,d15          ; (d15-8)/4
]
[
adda #8,r0,r1        ; Input address
move.w #2,n3
]
[
move.2l (r0)+n3,d0:d1
move.2l (r1)+n3,d2:d3
]
[
abs2 d0,d0   abs2 d1,d1
abs2 d2,d2   abs2 d3,d3
move.2l (r0)+n3,d4:d5
move.2l (r1)+n3,d6:d7
]
[
abs2 d4,d4   abs2 d5,d5
abs2 d6,d6   abs2 d7,d7
doensh0 d15          ; loop count for max search
]

; Loop for max (real,imag) search
loopstart0
    [
    max2 d0,d4   max2 d1,d5
    max2 d2,d6   max2 d3,d7
    move.2l (r0)+n3,d0:d1
    move.2l (r1)+n3,d2:d3
    ]
    [
    abs2 d0,d0   abs2 d1,d1
    abs2 d2,d2   abs2 d3,d3
    ]
loopend0

[
max2 d0,d4   max2 d1,d5
max2 d2,d6   max2 d3,d7
move.l (sp-20),r2       ; num_butterfly
move.l (sp-24),r4       ; num_subgroup
]
[
tfr d6,d0   tfr d7,d1
move.l (sp-44),r5       ; r5 -> Wb
move.l (sp-48),r6       ; r6 -> Wc
]
[
max2 d0,d4   max2 d1,d5
move.l (sp-28),r12      ; num_radix_offset
move.l (sp-52),r7       ; r7 -> Wd
]
[
tfr d5,d0
dosetup0 radix4_dit_m_stage_loop1    ; radix4 loop1 (outer loop)
]
```

```
    [
    max2 d0,d4
    dosetup1 radix4_dit_m_stage_loop2_1  ; radix4 loop2_1 (inner loop)
    ]
    [
    move.l (sp-8),r0   ; A input pointer
    ]
    [
    sxt.w d4,d0
    dosetup2 radix4_dit_m_stage_loop2_2    ; radix4 loop2_2 (inner loop)
    ]
    [
    extract #16,#16,d4,d4
    move.l (sp-12),r9  ; A output pointer
    ]
    [
    max d0,d4          ; Max real/imag
    adda r3,r2         ; num_butterfly[s]
    adda r3,r4         ; num_subgroup[s]
    ]
    [
    clb d4,d0          ; count reading bit
    move.w (r2),r13    ; num_butterfly[s]
    move.w (r4),r14    ; num_subgroup[s]
    ]
    [
    move.l (sp-68),r2  ; psiScale
    ]
    [
    neg d0             ; negates
    adda r3,r12        ; num_radix_offset[s]
    ]
    [
    clr d14
    sub #16,d0         ; limit 16-bit
    move.w #S42,d2
    move.w #3,n3       ; output offset for radix-4
    ]
    [
    add #2,d14
    cmpgt.w #(S41+1),d0  ; Check if siNorm > S41+1
    move.w #2,d15        ; scale down = 2
    ]
    [
    ift clr d15          ; scale down = 0      if siNorm > S41+1
    ifa
    cmpeq.w #(S41+1),d0  ; Check if siNorm = S41+1
    ]
    [
    tfrt d14,d15         ; scale down = 2      if siNorm = S41+1
    move.w #S41,d1
    ]
 ENDIF    ;// IF @DEF('FIX_SCALE')
    [
    move.l d15,(sp-64) ; Set norm to stack
    move.w d15,(r2)+   ; psiScale <- norm
    ]
```

```
[
move.l r2,(sp-68)  ; psiScale
move.w (r12),r15   ; num_radix_offset[s]
]
[
tfra r13,n0        ; The stage twiddle factor's offset
tfra r14,n1
]
; For next stage input/output
[
move.l r9,(sp-8)       ; base_address
move.l r0,(sp-12)      ; out_address
]
[
tfra r9,r8             ; A output
adda r15,r9            ; B output pointer (long)
]
[
move.l d15,r1          ; psiScale <- norm
tfra r9,r10            ; B output
]

;--- Start Radix-4 (Middle stage)

[
adda r15,r10           ; C output pointer (long)
move.l SR,d3
]
[
move.l d3,r4
bmset #$1000,SR.L    ; W20=1
]
[
tfra r10,r11         ; C output
bmclr #$0080,SR.L    ; SM2=0
]
[
adda r15,r11         ; D output pointer (long)
bmclr #$0030,SR.L    ; SCM=00
]
[
move.l (sp-44),r5        ; r5 -> Wb
cmpeqa.w #1,r1
]
[
ift bmset #$0010,SR.L    ; SCM=01
ifa
move.l (sp-48),r6        ; r6 -> Wc
]
[
move.l (sp-52),r7        ; r7 -> Wd
cmpeqa.w #2,r1
]
[
ift bmset #$0030,SR.L    ; SCM=11
ifa
move.l (r6)+n0,d2        ; d14=Wcr:Wci
```

```
    ]
    [
    tfra r13,n0
    bmtsts #$0001,r13.l
    ]
    [
    asra r13
    asra r14
    ]
    [
    adda #-1,r13,r12
    adda #8,r0,r1                   ; Input address
    ]
    [
    MOVE2.2F (r0)+,d0               ; d0=Ar:Ai
    MOVE2.2F (r1)+,d1               ; d1=Cr:Ci
    ]
    [ ;01
    mac2assar  d1,d2,d0.H,d12       ; M1_1.H = IA[re] + IC[re] * WC[re] - IC[im] * WC[im]
                                    ; M1_1.L = IA[re] - IC[re] * WC[re] + IC[im] * WC[im]
    mac2aassi  d1,d2,d0.L,d13       ; M1_2.H = IA[im] + IC[re] * WC[im] + IC[im] * WC[re]
                                    ; M1_2.L = IA[im] - IC[re] * WC[im] - IC[im] * WC[re]
    MOVE2.2F (r0)+n3,d8             ; d8=Br:Bi
    move.l (r5)+n0,d10              ; d10=Wbr:Wbi
    ]
    [ ;02
    mac2assar  d8,d10,d12.H,d12     ; M2_1.H = M1_1.H + IB[re] * WB[re] - IB[im] * WB[im]
                                    ; M2_1.L = M1_1.H - IB[re] * WB[re] + IB[im] * WB[im]
    mac2aassi  d8,d10,d12.L,d4      ; M2_2.H = M1_1.L + IB[re] * WB[im] + IB[im] * WB[re]
                                    ; M2_2.L = M1_1.L - IB[re] * WB[im] - IB[im] * WB[re]
    mac2aassi  d8,d10,d13.H,d13     ; M2_3.H = M1_2.H + IB[re] * WB[im] + IB[im] * WB[re]
                                    ; M2_3.L = M1_2.H - IB[re] * WB[im] - IB[im] * WB[re]
    mac2saasr  d8,d10,d13.L,d5      ; M2_4.H = M1_2.L - IB[re] * WB[re] + IB[im] * WB[im]
                                    ; M2_4.L = M1_2.L + IB[re] * WB[re] - IB[im] * WB[im]
    move.l (r7)+n0,d11              ; d11=Wdr:Wdi
    doen0 r14                       ; loop0 = num_subgroup[s]
    ]
    FALIGN
radix4_dit_m_stage_loop1:           ; --> Butterflies/Subgroup
    loopstart0
        [
        doen1 r13                         ; loop1 = num_butterfly[s]
        MOVE2.2F (r1)+n3,d9               ; d9=Dr:Di
        ]
        ;/// Inter loop
        FALIGN
radix4_dit_m_stage_loop2_1                 ; --> Subgroup
        loopstart1
            [ ;03
          mac2assar  d9,d11,d12       ; OA[re] = M2_1.H + ID[re] * WD[re] - ID[im] * WD[im]
                                     ; OC[re] = M2_1.L - ID[re] * WD[re] + ID[im] * WD[im]
          mac2ssaai  d9,d11,d4        ; OB[re] = M2_2.H - ID[re] * WD[im] - ID[im] * WD[re]
                                     ; OD[re] = M2_2.L + ID[re] * WD[im] + ID[im] * WD[re]
          mac2aassi  d9,d11,d13       ; OA[im] = M2_3.H + ID[re] * WD[im] + ID[im] * WD[re]
                                     ; OC[im] = M2_3.L - ID[re] * WD[im] - ID[im] * WD[re]
          mac2assar  d9,d11,d5        ; OB[im] = M2_4.H + ID[re] * WD[re] - ID[im] * WD[im]
                                     ; OD[im] = M2_4.L - ID[re] * WD[re] + ID[im] * WD[im]
```

```
      MOVE2.2F (r0)+,d0             ; d0=Ar:Ai
      MOVE2.2F (r1)+,d8             ; d8=Cr:Ci
      ]
      [ ;01
   mac2assar  d8,d2,d0.H,d6    ; M1_1.H = IA[re] + IC[re] * WC[re] - IC[im] * WC[im]
                              ; M1_1.L = IA[re] - IC[re] * WC[re] + IC[im] * WC[im]
   mac2aassi  d8,d2,d0.L,d7    ; M1_2.H = IA[im] + IC[re] * WC[im] + IC[im] * WC[re]
                              ; M1_2.L = IA[im] - IC[re] * WC[im] - IC[im] * WC[re]
      MOVE2.2F (r0)+n3,d8           ; d8=Br:Bi
      MOVE2.2F (r1)+n3,d9           ; d9=Dr:Di
      ]
      [ ;02
   mac2assar  d8,d10,d6.H,d6    ; M2_1.H = M1_1.H + IB[re] * WB[re] - IB[im] * WB[im]
                              ; M2_1.L = M1_1.H - IB[re] * WB[re] + IB[im] * WB[im]
   mac2aassi  d8,d10,d6.L,d14   ; M2_2.H = M1_1.L + IB[re] * WB[im] + IB[im] * WB[re]
                              ; M2_2.L = M1_1.L - IB[re] * WB[im] - IB[im] * WB[re]
   mac2aassi  d8,d10,d7.H,d7    ; M2_3.H = M1_2.H + IB[re] * WB[im] + IB[im] * WB[re]
                              ; M2_3.L = M1_2.H - IB[re] * WB[im] - IB[im] * WB[re]
   mac2saasr  d8,d10,d7.L,d15   ; M2_4.H = M1_2.L - IB[re] * WB[re] + IB[im] * WB[im]
                              ; M2_4.L = M1_2.L + IB[re] * WB[re] - IB[im] * WB[im]
      MOVE2.2F (r0)+,d0              ; d0=Ar:Ai,
      MOVE2.2F (r1)+,d1              ; d1=Cr:Ci
      ]
      [ ;03
   mac2assar  d9,d11,d6        ; OA[re] = M2_1.H + ID[re] * WD[re] - ID[im] * WD[im]
                              ; OC[re] = M2_1.L - ID[re] * WD[re] + ID[im] * WD[im]
   mac2ssaai  d9,d11,d14       ; OB[re] = M2_2.H - ID[re] * WD[im] - ID[im] * WD[re]
                              ; OD[re] = M2_2.L + ID[re] * WD[im] + ID[im] * WD[re]
   mac2aassi  d9,d11,d7        ; OA[im] = M2_3.H + ID[re] * WD[im] + ID[im] * WD[re]
                              ; OC[im] = M2_3.L - ID[re] * WD[im] - ID[im] * WD[re]
   mac2assar  d9,d11,d15       ; OB[im] = M2_4.H + ID[re] * WD[re] - ID[im] * WD[im]
                              ; OD[im] = M2_4.L - ID[re] * WD[re] + ID[im] * WD[im]
      MOVE2.2F (r0)+n3,d8            ; d8=Br:Bi,
      MOVE2.2F (r1)+n3,d9            ; d9=Dr:Di
      ]
      [ ;01
   mac2assar  d1,d2,d0.H,d12    ; M1_1.H = IA[re] + IC[re] * WC[re] - IC[im] * WC[im]
                              ; M1_1.L = IA[re] - IC[re] * WC[re] + IC[im] * WC[im]
   mac2aassi  d1,d2,d0.L,d13    ; M1_2.H = IA[im] + IC[re] * WC[im] + IC[im] * WC[re]
                              ; M1_2.L = IA[im] - IC[re] * WC[im] - IC[im] * WC[re]
      MOVERH.4F d12:d13:d6:d7,(r8)+     ; save OA1:OA2
      MOVERL.4F d12:d13:d6:d7,(r10)+    ; save OC1:OC2
      ]
      [ ;02
   mac2assar  d8,d10,d12.H,d12  ; M2_1.H = M1_1.H + IB[re] * WB[re] - IB[im] * WB[im]
                              ; M2_1.L = M1_1.H - IB[re] * WB[re] + IB[im] * WB[im]
   mac2aassi  d8,d10,d12.L,d4   ; M2_2.H = M1_1.L + IB[re] * WB[im] + IB[im] * WB[re]
                              ; M2_2.L = M1_1.L - IB[re] * WB[im] - IB[im] * WB[re]
   mac2aassi  d8,d10,d13.H,d13  ; M2_3.H = M1_2.H + IB[re] * WB[im] + IB[im] * WB[re]
                              ; M2_3.L = M1_2.H - IB[re] * WB[im] - IB[im] * WB[re]
   mac2saasr  d8,d10,d13.L,d5   ; M2_4.H = M1_2.L - IB[re] * WB[re] + IB[im] * WB[im]
                              ; M2_4.L = M1_2.L + IB[re] * WB[re] - IB[im] * WB[im]
      MOVERH.4F d4:d5:d14:d15,(r9)+     ; save OB1:OB2
      MOVERL.4F d4:d5:d14:d15,(r11)+    ; save OD1:OD2
      ]
  loopend1
  [
  iff doen2 r12                  ; loop1 = num_butterfly[s]
  iff move.l (r6)+n0,d2          ; d2=Wcr:Wci
  ifa
  tfr d9,d3
  ]
  [ ;01
  iff move.l (r5)+n0,d10         ; d10=Wbr:Wbi
  iff move.l (r7)+n0,d11         ; d11=Wdr:Wdi
  ifa
 mac2assar  d1,d2,d0.H,d12      ; M1_1.H = IA[re] + IC[re] * WC[re] - IC[im] * WC[im]
                             ; M1_1.L = IA[re] - IC[re] * WC[re] + IC[im] * WC[im]
 mac2aassi  d1,d2,d0.L,d13      ; M1_2.H = IA[im] + IC[re] * WC[im] + IC[im] * WC[re]
                             ; M1_2.L = IA[im] - IC[re] * WC[im] - IC[im] * WC[re]
  ]
  [ ;02
 mac2assar  d8,d10,d12.H,d12    ; M2_1.H = M1_1.H + IB[re] * WB[re] - IB[im] * WB[im]
                             ; M2_1.L = M1_1.H - IB[re] * WB[re] + IB[im] * WB[im]
 mac2aassi  d8,d10,d12.L,d4     ; M2_2.H = M1_1.L + IB[re] * WB[im] + IB[im] * WB[re]
                             ; M2_2.L = M1_1.L - IB[re] * WB[im] - IB[im] * WB[re]
 mac2aassi  d8,d10,d13.H,d13    ; M2_3.H = M1_2.H + IB[re] * WB[im] + IB[im] * WB[re]
                             ; M2_3.L = M1_2.H - IB[re] * WB[im] - IB[im] * WB[re]
 mac2saasr  d8,d10,d13.L,d5     ; M2_4.H = M1_2.L - IB[re] * WB[re] + IB[im] * WB[im]
                             ; M2_4.L = M1_2.L + IB[re] * WB[re] - IB[im] * WB[im]
  MOVE2.2F (r0)+,d0              ; d0=Ar:Ai,
  MOVE2.2F (r1)+,d1              ; d1=Cr:Ci
  ]
  [ ;03
 mac2assar  d9,d11,d12          ; OA[re] = M2_1.H + ID[re] * WD[re] - ID[im] * WD[im]
                             ; OC[re] = M2_1.L - ID[re] * WD[re] + ID[im] * WD[im]
 mac2ssaai  d9,d11,d4           ; OB[re] = M2_2.H - ID[re] * WD[im] - ID[im] * WD[re]
                             ; OD[re] = M2_2.L + ID[re] * WD[im] + ID[im] * WD[re]
 mac2aassi  d9,d11,d13          ; OA[im] = M2_3.H + ID[re] * WD[im] + ID[im] * WD[re]
                             ; OC[im] = M2_3.L - ID[re] * WD[im] - ID[im] * WD[re]
 mac2assar  d9,d11,d5           ; OB[im] = M2_4.H + ID[re] * WD[re] - ID[im] * WD[im]
                             ; OD[im] = M2_4.L - ID[re] * WD[re] + ID[im] * WD[im]
  MOVE2.2F (r0)+n3,d8            ; d8=Br:Bi,
  MOVE2.2F (r1)+n3,d9            ; d9=Dr:Di
  ]
  [
  ift doen2 r13                  ; loop1 = num_butterfly[s]
  ift move.l (r6)+n0,d2          ; d14=Wcr:Wci
  ]
  [ ;01
  ift move.l (r5)+n0,d10         ; d10=Wbr:Wbi
  ift move.l (r7)+n0,d11         ; d11=Wdr:Wdi
  ifa
 mac2assar  d1,d2,d0.H,d6       ; M1_1.H = IA[re] + IC[re] * WC[re] - IC[im] * WC[im]
                             ; M1_1.L = IA[re] - IC[re] * WC[re] + IC[im] * WC[im]
 mac2aassi  d1,d2,d0.L,d7       ; M1_2.H = IA[im] + IC[re] * WC[im] + IC[im] * WC[re]
                             ; M1_2.L = IA[im] - IC[re] * WC[im] - IC[im] * WC[re]
  ]
  [ ;02
  ifa
 mac2assar  d8,d10,d6.H,d6      ; M2_1.H = M1_1.H + IB[re] * WB[re] - IB[im] * WB[im]
                             ; M2_1.L = M1_1.H - IB[re] * WB[re] + IB[im] * WB[im]
 mac2aassi  d8,d10,d6.L,d14     ; M2_2.H = M1_1.L + IB[re] * WB[im] + IB[im] * WB[re]
                             ; M2_2.L = M1_1.L - IB[re] * WB[im] - IB[im] * WB[re]
```

```
                mac2aassi  d8,d10,d7.H,d7    ; M2_3.H = M1_2.H + IB[re] * WB[im] + IB[im] * WB[re]
                                             ; M2_3.L = M1_2.H - IB[re] * WB[im] - IB[im] * WB[re]
                mac2saasr  d8,d10,d7.L,d15   ; M2_4.H = M1_2.L - IB[re] * WB[re] + IB[im] * WB[im]
                                             ; M2_4.L = M1_2.L + IB[re] * WB[re] - IB[im] * WB[im]
                 MOVE2.2F (r0)+,d0            ; d0=Ar:Ai,
                 MOVE2.2F (r1)+,d1            ; d1=Cr:Ci
                 ]
                 FALIGN
        radix4_dit_m_stage_loop2_2                ; --> Subgroup
                 loopstart2
                     [ ;03
                   mac2assar  d9,d11,d6       ; OA[re] = M2_1.H + ID[re] * WD[re] - ID[im] * WD[im]
                                             ; OC[re] = M2_1.L - ID[re] * WD[re] + ID[im] * WD[im]
                   mac2ssaai  d9,d11,d14      ; OB[re] = M2_2.H - ID[re] * WD[im] - ID[im] * WD[re]
                                             ; OD[re] = M2_2.L + ID[re] * WD[im] + ID[im] * WD[re]
                   mac2aassi  d9,d11,d7       ; OA[im] = M2_3.H + ID[re] * WD[im] + ID[im] * WD[re]
                                             ; OC[im] = M2_3.L - ID[re] * WD[im] - ID[im] * WD[re]
                   mac2assar  d9,d11,d15      ; OB[im] = M2_4.H + ID[re] * WD[re] - ID[im] * WD[im]
                                             ; OD[im] = M2_4.L - ID[re] * WD[re] + ID[im] * WD[im]
                     MOVE2.2F (r0)+n3,d8          ; d8=Br:Bi,
                     MOVE2.2F (r1)+n3,d9          ; d9=Dr:Di
                     ]
                     [ ;01
                   mac2assar  d1,d2,d0.H,d12   ; M1_1.H = IA[re] + IC[re] * WC[re] - IC[im] * WC[im]
                                             ; M1_1.L = IA[re] - IC[re] * WC[re] + IC[im] * WC[im]
                   mac2aassi  d1,d2,d0.L,d13   ; M1_2.H = IA[im] + IC[re] * WC[im] + IC[im] * WC[re]
                                             ; M1_2.L = IA[im] - IC[re] * WC[im] - IC[im] * WC[re]
                     MOVERH.4F d12:d13:d6:d7,(r8)+     ; save OA1:OA2
                     MOVERL.4F d12:d13:d6:d7,(r10)+    ; save OC1:OC2
                     ]
                     [ ;02
                     mac2assar  d8,d10,d12.H,d12    ; M2_1.H = M1_1.H + IB[re] * WB[re] -
                                                    ; IB[im] * WB[im]
                                                    ; M2_1.L = M1_1.H - IB[re] * WB[re] +
                                                    ; IB[im] * WB[im]
                     mac2aassi  d8,d10,d12.L,d4     ; M2_2.H = M1_1.L + IB[re] * WB[im] +
                                                    ; IB[im] * WB[re]
                                                    ; M2_2.L = M1_1.L - IB[re] * WB[im] -
                                                    ; IB[im] * WB[re]
                     mac2aassi  d8,d10,d13.H,d13    ; M2_3.H = M1_2.H + IB[re] * WB[im] +
                                                    ; IB[im] * WB[re]
                                                    ; M2_3.L = M1_2.H - IB[re] * WB[im] -
                                                    ; IB[im] * WB[re]
                     mac2saasr  d8,d10,d13.L,d5     ; M2_4.H = M1_2.L - IB[re] * WB[re] +
                                                    ; IB[im] * WB[im]
                                                    ; M2_4.L = M1_2.L + IB[re] * WB[re] -
                                                    ; IB[im] * WB[im]
                     MOVERH.4F d4:d5:d14:d15,(r9)+     ; save OB1:OB2
                     MOVERL.4F d4:d5:d14:d15,(r11)+    ; save OD1:OD2
                     ]
                     [ ;03
                   mac2assar  d9,d11,d12      ; OA[re] = M2_1.H + ID[re] * WD[re] - ID[im] * WD[im]
                                             ; OC[re] = M2_1.L - ID[re] * WD[re] + ID[im] * WD[im]
                   mac2ssaai  d9,d11,d4       ; OB[re] = M2_2.H - ID[re] * WD[im] - ID[im] * WD[re]
                                             ; OD[re] = M2_2.L + ID[re] * WD[im] + ID[im] * WD[re]
                   mac2aassi  d9,d11,d13      ; OA[im] = M2_3.H + ID[re] * WD[im] + ID[im] * WD[re]
                                             ; OC[im] = M2_3.L - ID[re] * WD[im] - ID[im] * WD[re]
```

```
      mac2assar  d9,d11,d5       ; OB[im] = M2_4.H + ID[re] * WD[re] - ID[im] * WD[im]
                                ; OD[im] = M2_4.L - ID[re] * WD[re] + ID[im] * WD[im]
        MOVE2.2F (r0)+,d0           ; d0=Ar:Ai,
        MOVE2.2F (r1)+,d1           ; d1=Cr:Ci
        ]
        [ ;01
      mac2assar  d1,d2,d0.H,d6    ; M1_1.H = IA[re] + IC[re] * WC[re] - IC[im] * WC[im]
                                ; M1_1.L = IA[re] - IC[re] * WC[re] + IC[im] * WC[im]
      mac2aassi  d1,d2,d0.L,d7    ; M1_2.H = IA[im] + IC[re] * WC[im] + IC[im] * WC[re]
                                ; M1_2.L = IA[im] - IC[re] * WC[im] - IC[im] * WC[re]
        MOVE2.2F (r0)+n3,d8         ; d8=Br:Bi,
        MOVE2.2F (r1)+n3,d9         ; d9=Dr:Di
        ]
        [ ;02
     mac2assar  d8,d10,d6.H,d6    ; M2_1.H = M1_1.H + IB[re] * WB[re] - IB[im] * WB[im]
                                ; M2_1.L = M1_1.H - IB[re] * WB[re] + IB[im] * WB[im]
     mac2aassi  d8,d10,d6.L,d14   ; M2_2.H = M1_1.L + IB[re] * WB[im] + IB[im] * WB[re]
                                ; M2_2.L = M1_1.L - IB[re] * WB[im] - IB[im] * WB[re]
     mac2aassi  d8,d10,d7.H,d7    ; M2_3.H = M1_2.H + IB[re] * WB[im] + IB[im] * WB[re]
                                ; M2_3.L = M1_2.H - IB[re] * WB[im] - IB[im] * WB[re]
     mac2saasr  d8,d10,d7.L,d15   ; M2_4.H = M1_2.L - IB[re] * WB[re] + IB[im] * WB[im]
                                ; M2_4.L = M1_2.L + IB[re] * WB[re] - IB[im] * WB[im]
        MOVE2.2F (r0)+,d0            ; d0=Ar:Ai,
        MOVE2.2F (r1)+,d1            ; d1=Cr:Ci
        ]
    loopend2
    [ ;03
   mac2assar  d9,d11,d6          ; OA[re] = M2_1.H + ID[re] * WD[re] - ID[im] * WD[im]
                                ; OC[re] = M2_1.L - ID[re] * WD[re] + ID[im] * WD[im]
   mac2ssaai  d9,d11,d14         ; OB[re] = M2_2.H - ID[re] * WD[im] - ID[im] * WD[re]
                                ; OD[re] = M2_2.L + ID[re] * WD[im] + ID[im] * WD[re]
   mac2aassi  d9,d11,d7          ; OA[im] = M2_3.H + ID[re] * WD[im] + ID[im] * WD[re]
                                ; OC[im] = M2_3.L - ID[re] * WD[im] - ID[im] * WD[re]
   mac2assar  d9,d11,d15         ; OB[im] = M2_4.H + ID[re] * WD[re] - ID[im] * WD[im]
                                ; OD[im] = M2_4.L - ID[re] * WD[re] + ID[im] * WD[im]
    MOVE2.2F (r0)+n3,d8             ; d8=Br:Bi
    move.l (r5)+n0,d10              ; d10=Wbr:Wbi
    ]
    [
    move.l (r6)+n0,d2               ; d2 =Wcr:Wci
    move.l (r7)+n0,d11              ; d11=Wdr:Wdi
    ]
    [ ;01
   mac2assar  d1,d2,d0.H,d12      ; M1_1.H = IA[re] + IC[re] * WC[re] - IC[im] * WC[im]
                                ; M1_1.L = IA[re] - IC[re] * WC[re] + IC[im] * WC[im]
   mac2aassi  d1,d2,d0.L,d13      ; M1_2.H = IA[im] + IC[re] * WC[im] + IC[im] * WC[re]
                                ; M1_2.L = IA[im] - IC[re] * WC[im] - IC[im] * WC[re]
    MOVERH.4F d12:d13:d6:d7,(r8)+ ; save OA1:OA2
    MOVERL.4F d12:d13:d6:d7,(r10)+ ; save OC1:OC2
    ]
    [ ;02
   mac2assar  d8,d10,d12.H,d12   ; M2_1.H = M1_1.H + IB[re] * WB[re] - IB[im] * WB[im]
                                ; M2_1.L = M1_1.H - IB[re] * WB[re] + IB[im] * WB[im]
   mac2aassi  d8,d10,d12.L,d4    ; M2_2.H = M1_1.L + IB[re] * WB[im] + IB[im] * WB[re]
                                ; M2_2.L = M1_1.L - IB[re] * WB[im] - IB[im] * WB[re]
   mac2aassi  d8,d10,d13.H,d13   ; M2_3.H = M1_2.H + IB[re] * WB[im] + IB[im] * WB[re]
                                ; M2_3.L = M1_2.H - IB[re] * WB[im] - IB[im] * WB[re]
```

```
            mac2saasr  d8,d10,d13.L,d5    ; M2_4.H = M1_2.L - IB[re] * WB[re] + IB[im] * WB[im]
                                         ; M2_4.L = M1_2.L + IB[re] * WB[re] - IB[im] * WB[im]
             MOVERH.4F d4:d5:d14:d15,(r9)+ ; save OB1:OB2
             MOVERL.4F d4:d5:d14:d15,(r11)+ ; save OD1:OD2
             ]
         loopend0
         [
         move.l (sp-16),r2        ; num_radix
         move.l (sp-40),r3        ; Load the stage counter
         ]
         move.l r4,d3
         [
         move.l d3,SR
         adda #2,r3,r3             ; +2
         ]
         [
         move.l r3,(sp-40)           ; Update the stage counter
         adda r3,r2                  ; num_butterfly[s]
         ]
         move.w (r2),r0              ; num_radix[s]
         cmpeqa.w #0,r0              ; num_radix[s]=0 or not
         [
         jt radix3_dit_last_stage  ; Go to radix-3 Last stage processing if num_radix[s]=0
         cmpeqa.w #3,r0              ; num_radix[s]=3 or not
         ]
         [
        jt radix3_dit_m_stage     ; Go to radix-3 Middle stage processing if num_radix[s]=3
         cmpeqa.w #1,r0              ; num_radix[s]=1 or not
         ]
         jt radix5_dit_last_stage  ; Go to radix-5 Last stage processing if num_radix[s]=1
         cmpeqa.w #4,r0              ; num_radix[s]=4 or not
         ]
        jt radix4_dit_m_stage     ; Go to radix-4 Middle stage processing if num_radix[s]=4

;////////////////////////////////////////////////////////////////////////
; Radix-5 Stages:
;////////////////////////////////////////////////////////////////////////

radix5_init:

   ;---------------------------------------------------------
   ; Radix-5 (Middle stage)
   ;---------------------------------------------------------

radix5_dit_m_stage:
 IF @DEF('FIX_SCALE')
    [
    clr d0
    move.l (sp-20),r2       ; num_butterfly
    move.l (sp-24),r4       ; num_subgroup
    ]
    [
    add #SCALE_RADIX_5,d0
    move.l (sp-40),r3       ; Stage counter
    move.l (sp-44),r5       ; r5 -> Wb
    ]
    [
    move.w #2,n0
    dosetup0 radix5_dit_m_stage_loop1
    ]
    [
    move.l (sp-48),r6       ; r6 -> Wc
    move.l (sp-28),r1       ; num_radix_offset
    ]
    [
    move.l (sp-52),r7       ; r7 -> Wd
    move.l (sp-56),r8       ; r8 -> We
    ]
    [
    adda r3,r2             ; num_butterfly[s]
    adda r3,r4             ; num_subgroup[s]
    ]
    [
    move.w (r2),r14         ; num_butterfly[s] ; butterflies/subgroup
    move.w (r4),r13         ; num_subgroup[s]  ; subgroups/stage
    ]
    [
    move.l (sp-12),r9     ; A output pointer
    move.l (sp-68),r4     ; psiScale
    ]
    [
    move.l (sp-8),r0     ; A input pointer
    adda r3,r1             ; num_radix_offset[s]
    ]
    [
    move.w (r1),r15         ; num_radix_offset[s]
    ]
    ; For next stage input/output
    [
    move.l r9,(sp-8)     ; base_address
    move.l r0,(sp-12)     ; out_address
    ]
    [
    tfra r14,r2             ; Middle stage twiddle factor's offset
    tfra r9,r8                 ; A output
    ]
 ELSE    ;//IF @DEF('FIX_SCALE')
    ;-- Find max value (real & imag) & stage initialization
    [
    move.l (sp-8),r0     ; input base (A)
    move.l (sp-60),r2    ; DFT point
    ]
    ; 3 STALLS on r2
    [
    move.w (r2),d15      ; DFT point
    move.l (sp-40),r3     ; Stage counter
    ]
    [
    sub #8,d15           ; d15-8
    ]
    [
    asrr #2,d15          ; (d15-8)/4
    ]
    [
    adda #8,r0,r1       ; Input address
    move.w #2,n0
    ]
    [
    move.2l (r0)+n0,d0:d1
    move.2l (r1)+n0,d2:d3
    ]
    [
    abs2 d0,d0   abs2 d1,d1
    abs2 d2,d2   abs2 d3,d3
    move.2l (r0)+n0,d4:d5
    move.2l (r1)+n0,d6:d7
    ]
    [
    abs2 d4,d4   abs2 d5,d5
    abs2 d6,d6   abs2 d7,d7
    doensh0 d15         ; loop count for max search
    ]

    ; Loop for max (real,imag) search
    loopstart0
        [
        max2 d0,d4   max2 d1,d5
        max2 d2,d6   max2 d3,d7
        move.2l (r0)+n0,d0:d1
        move.2l (r1)+n0,d2:d3
        ]
        [
        abs2 d0,d0   abs2 d1,d1
        abs2 d2,d2   abs2 d3,d3
        ]
    loopend0
    [
    max2 d0,d4   max2 d1,d5
    max2 d2,d6   max2 d3,d7
    move.l (sp-20),r2      ; num_butterfly
    move.l (sp-24),r4      ; num_subgroup
    ]
    [
    tfr d6,d0   tfr d7,d1
    move.l (sp-44),r5      ; r5 -> Wb
    dosetup0 radix5_dit_m_stage_loop1
    ]
    [
    max2 d0,d4   max2 d1,d5
    move.l (sp-48),r6      ; r6 -> Wc
    move.l (sp-28),r1      ; num_radix_offset
    ]
    [
    tfr d5,d0
    move.l (sp-52),r7      ; r7 -> Wd
    move.l (sp-56),r8      ; r8 -> We
    ]
    [
    max2 d0,d4
    adda r3,r2            ; num_butterfly[s]
    adda r3,r4            ; num_subgroup[s]
```

```
        ]
        [
        sxt.w d4,d0
        move.w (r2),r14         ; num_butterfly[s] ; butterflies/subgroup
        move.w (r4),r13         ; num_subgroup[s]  ; subgroups/stage
        ]
        [
        extract #16,#16,d4,d4
        move.l (sp-8),r0        ; A input pointer
        move.l (sp-12),r9       ; A output pointer
        ]
        [
        max d0,d4             ; Max real/imag
        move.l (sp-68),r4     ; psiScale
        adda r3,r1            ; num_radix_offset[s]
        ]
        ; For next stage input/output
        [
        clb d4,d0               ; count reading bit
        move.w #S54,d2
        move.w (r1),r15         ; num_radix_offset[s]
        ]
        [
        neg d0                ; negates
        move.l r9,(sp-8)      ; base_address
        move.l r0,(sp-12)     ; out_address
        ]
        [
        sub #16,d0            ; limit 16-bit
        tfra r14,r2           ; Middle stage twiddle factor's offset
        tfra r9,r8            ; A output
        ]
        [
        cmpgt.w #S53,d0       ; Check if norm > S or norm <= S
        move.w #S53,d1
        ]
        [
        iff sub d0,d1,d0      ; scale down = S - d0 if norm <= S
        ift clr d0            ; scale down = 0      if norm >  S
        ]
     ENDIF    ;// IF @DEF('FIX_SCALE')
        [
        move.l d0,(sp-64)     ; Set norm to stack
        move.w d0,(r4)+       ; psiScale <- norm
        ]
        [
        move.l r4,(sp-68)     ; psiScale
        adda r15,r9           ; B output pointer (long)
        ]
        [
        tfra r9,r10           ; B output
        move.l (sp-44),r4     ; r4 -> Wb
        ]
        [
        adda r15,r10          ; C output pointer (long)
        move.l (sp-48),r5     ; r5 -> Wc
        ]
        [
        tfra r10,r11          ; C output
        adda #4,r0,r1         ; A input
        ]
        [
        adda r15,r11          ; D output pointer (long)
        dosetup1 radix5_dit_m_stage_loop2
        ]
        [
        tfra r11,r12          ; D output
        move.l #radix5parameter,r14  ; Point to the radix-5 parameter array
        ]
        [
        adda r15,r12          ; E output pointer (long)
        move.l #radix5parameter+4,r15  ; Point to the radix-5 parameter array
        ]
        ;--- Start Radix-5 (Middle stage)
        [ ;
        move.2f (r0)+n0,d12:d13    ; d12.H = Ar, d13.H = Ai
        move.l (r1)+n0,d14         ; d14 = Br:Bi
        ]
        [ ;
        move.l (r0)+n0,d15         ; d15 = Cr:Ci
        move.l (sp-64),d8          ; Scaling factor
        ]
        [ ;
        asrr2 d8,d14               ; d14 = Br:Bi>>S
        asrr2 d8,d15               ; d15 = Cr:Ci>>S
        move.l (sp-52),r6          ; r6 -> Wd
        move.l (sp-56),r7          ; r7 -> We
        ]
        [ ;
        asrr d8,d12                ; d12  Ar>>S
        move.l (r0)+,d9            ; d9 = Er:Ei
        doen0 r13                  ; num_subgroup ; subgroups/stage
        ]
        FALIGN
    radix5_dit_m_stage_loop1:      ; --> Butterflies/Subgroup
        loopstart0
            [
            doen1 r2               ; num_butterfly[s]
            move.l (r4),d2         ; d2 = Wbr:Wbi
            ]
            FALIGN
    radix5_dit_m_stage_loop2:      ; --> Subgroup
            loopstart1
                [ ; 01
                asrr2 d8,d9                 ; d9 = Er:Ei>>S
                MPYRE d2,d14,d6             ; d6  = Br' = (Br*Wbr - Bi*Wbi)
                MPYIM d2,d14,d7             ; d7  = Bi' = (Br*Wbi + Bi*Wbr)
                tfr d8,d1                   ; Scaling factor
                move.l (r1)+n0,d8           ; d8 = Dr:Di
                move.l (r5),d10             ; d10 = Wcr:Wci
                ]
                [ ; 02
                asrr2 d1,d8                 ; d8 = Dr:Di>>S
                pack.2f d7,d6,d0            ; d0 = Br':Bi'
                MPYRE d10,d15,d14           ; d14  = Cr' = (Cr*Wcr - Ci*Wci)
                MPYIM d10,d15,d15           ; d15  = Ci' = (Cr*Wci + Ci*Wcr)
                move.l (r7),d3              ; d3 = Wer:Wei
                ]
                [ ; 03
                asrr d1,d13                 ; d13 = Ai>>S
                mac2assar d9,d3,d0.h,d6     ; M2_1 = (Br'+Er'):(Br'-Er')
                mac2aassi d9,d3,d0.l,d7     ; M2_2 = (Bi'+Ei'):(Bi'-Ei')
                pack.2f d15,d14,d0          ; d0 = Cr':Ci'
                move.l (r6),d11             ; d11 = Wdr:Wdi
                ]
                [ ; 04
                sod2aaii d12,d6,d4          ; Ar'+Br'+Er'
                sod2aaii d13,d7,d5          ; Ai'+Bi'+Ei'
                mac2assar d8,d11,d0.h,d14   ; M2_3 = (Cr'+Dr'):(Cr'-Dr')
                mac2aassi d8,d11,d0.l,d15   ; M2_4 = (Ci'+Di'):(Ci'-Di')
                move.2f (r14),d0:d1         ; d0:d1 = [a,b], Radix-5 parameters
                move.2f (r15),d8:d9         ; d8:d9 = [c,d], Radix-5 parameters
                ]
                [ ; 05
                move.l d6,d2
                move.l d7,d3
                sod2aaii d4,d14,d4         ; Aout_r = Ar'+Br'+Cr'+Dr'+Er'
                sod2aaii d5,d15,d5         ; Aout_i = Ai'+Bi'+Ci'+Di'+Ei'
                ]
                [ ; 06
                mac +d0,d2,d12             ; Ar' +  a*(Br'+Er')
                mac +d0,d3,d13             ; Ai' +  a*(Bi'+Ei')
                tfr d12,d14
                tfr d13,d15
                move.l d14,d10
                move.l d15,d11
                ]
                [ ; 07
                mac -d8,d10,d12            ; Ar' + (a*(Br'+Er')-c*(Cr'+Dr'))
                mac -d8,d11,d13            ; Ai' + (a*(Bi'+Ei')-c*(Ci'+Di'))
                mac -d8,d2,d14             ; Ar' -  c*(Br'+Er')
                mac -d8,d3,d15             ; Ai' -  c*(Bi'+Ei')
                adda #4,r0,r1              ; A input
                moves.2f d4:d5,(r8)+       ; out:[Aout_r,Aout_i]
                ]
                [ ; 08
                pack.2f d13,d12,d0
                mac +d0,d10,d14             ; Ar' - (c*(Br'+Er')-a*(Cr'+Dr'))
                mac +d0,d11,d15             ; Ai' - (c*(Bi'+Ei')-a*(Ci'+Di'))
                move.l (sp-64),d8           ; Scaling factor
                move.2f (r0)+n0,d12:d13     ; d12.H = Ar, d13.H = Ai
                ]
                [ ; 09
              mac2aassi d1,d3,d0.h,d4    ; M5_1.H = Ar'+(a(Br'+Er')-c(Cr'+Dr'))+(b(Bi'-Ei')),
                                          ; M5_1.L = Ar'+(a(Br'+Er')-
                                          ; c(Cr'+Dr'))-(b(Bi'-Ei'))
                mac2ssaai d1,d2,d0.l,d5     ; M5_2.H = Ai'+(a(Bi'+Ei')-
                                           ; c(Ci'+Di'))-(b(Br'-Er')),
                                           ; M5_2.L = Ai'+(a(Bi'+Ei')-
                                           ; c(Ci'+Di'))+(b(Br'-Er'))
                pack.2f d15,d14,d0
```

```
            move.l (r1)+n0,d14        ; d14 = Br:Bi
            move.l (r0)+n0,d15        ; d15 = Cr:Ci
            ]
            [ ; 10
            mac2aassi d9,d11,d4        ; Bout_r = Ar'+(a(Br'+Er')-c(Cr'+Dr'))+
                                       ; (b(Bi'-Ei')+d(Ci'-Di')),
                                       ; Eout_r = Ar'+(a(Br'+Er')-
                                       ; c(Cr'+Dr'))-(b(Bi'-Ei')+d(Ci'-Di'))
            mac2ssaai d9,d10,d5        ; Bout_i = Ai'+(a(Bi'+Ei')-c(Ci'+Di'))-
                                       ; (b(Br'-Er')+d(Cr'-Dr')), Eout_i =
                                       ; Ai'+(a(Bi'+Ei')-c(Ci'+Di'))+
                                       ; (b(Br'-Er')+d(Cr'-Dr'))
          mac2aassi d9,d3,d0.h,d6    ; M5_3.H = Ar'-(c(Br'+Er')-a(Cr'+Dr'))+(d(Bi'-Ei')),
                                      ; M5_3.L = Ar'-(c(Br'+Er')-a(Cr'+Dr'))-
                                      ;(d(Bi'-Ei'))
          mac2ssaai d9,d2,d0.l,d7    ; M5_4.H = Ai'-(c(Bi'+Ei')-a(Ci'+Di'))-(d(Br'-Er')),
                                      ; M5_4.L = Ai'-(c(Bi'+Ei')-a(Ci'+Di'))+
                                      ; (d(Br'-Er'))
            move.l (r0)+,d9            ; d9 = Er:Ei
            move.l (r4),d2             ; d2 = Wbr:Wbi
            ]
            [ ; 11
            asrr2 d8,d14             ; d14 = Br:Bi>>S
            asrr2 d8,d15             ; d15 = Cr:Ci>>S
            mac2saasi d1,d11,d6      ; Cout_r = Ar'-(c(Br'+Er')-a(Cr'+Dr'))+
                                     ;(d(Bi'-Ei')-b(Ci'-Di')), Dout_r =
                                     ; Ar'-(c(Br'+Er')-a(Cr'+Dr'))-
                                     ; (d(Bi'-Ei')-b(Ci'-Di'))
            mac2assai d1,d10,d7      ; Cout_i = Ai'-(c(Bi'+Ei')-a(Ci'+Di'))-
                                     ; (d(Br'-Er')-b(Cr'-Dr')), Dout_i =
                                     ; Ai'-(c(Bi'+Ei')-a(Ci'+Di'))+
                                     ; (d(Br'-Er')-b(Cr'-Dr'))
            moves.2f d4:d5,(r9)+     ; out:[Bout_r,Bout_i]
            move.2w d4:d5,(r12)+     ; out:[Eout_r,Eout_i]
            ]
            [ ; 12
            asrr d8,d12              ; d12 = Ar>>S
            moves.2f d6:d7,(r10)+    ; out:[Cout_r,Cout_i]
            move.2w d6:d7,(r11)+     ; out:[Dout_r,Dout_i]
            ]
        loopend1
         [
         addl2a r2,r4                ; updata twiddle factor pointers
         addl2a r2,r5
         ]
         [
         addl2a r2,r6
         addl2a r2,r7
         ]
    loopend0
    [
    move.l (sp-16),r2    ; num_radix
    move.l (sp-40),r3    ; Load the stage counter
    ]
    ; 3 STALLS on r3
    adda #2,r3,r3        ; +2
    [
    move.l r3,(sp-40)    ; Update the stage counter
    adda r3,r2           ; num_butterfly[s]
    ]
    move.w (r2),r0             ; num_radix[s]
    cmpeqa.w #5,r0             ; num_radix[s]=5 or not
    [
    jt radix5_dit_m_stage      ; Go to radix-5 Middle stage processing if num_radix[s]=5
    cmpeqa.w #1,r0             ; num_radix[s]=1 or not
    ]
    jt radix5_dit_last_stage   ; Go to radix-5 Last stage processing if num_radix[s]=1

    ;---------------------------------------------------------
    ; Radix-5 (Last stage)
    ;---------------------------------------------------------
radix5_dit_last_stage:

 IF @DEF('FIX_SCALE')
    [
    clr d12
    move.l (sp-40),r3         ; Stage counter
    move.l (sp-28),r15        ; num_radix_offset[0]
    ]
    [
    add #SCALE_RADIX_5,d12
    move.l (sp-24),r14        ; num_subgroup
    move.l (sp-12),r9         ; A output pointer
    ]
    [
    dosetup0 radix5_dit_last_stage_loop1
    move.l (sp-8),r0          ; A input pointer
    ]
    [
    move.w #2,n0
    move.l (sp-68),r2         ; psiScale
    ]
    [
    adda r3,r15               ; num_radix_offset[s]
    move.l (sp-44),r4         ; r4 -> Wb
    ]
    [
    adda r3,r14               ; siNumSubgroup[s]
    move.w (r15),r15          ; num_radix_offset[s]
    ]
    [
    adda #4,r0,r1             ; A input
    move.l (sp-48),r5         ; r5 -> Wc
    ]
    [
    move.l (sp-52),r6         ; r6 -> Wd
    move.l (sp-56),r7         ; r7 -> We
    ]
    [
    tfra r9,r8                ; A output
    move.w (r14),d5           ; siNumSubgroup[s]
    ]
    [
    adda r15,r9               ; B output pointer (long)
    doen0 d5                 ; Loop for num_butterfly
    ]
    [
    tfra r9,r10                  ; B output
    move.l #radix5parameter,r14  ; Point to the radix-5 parameter array
    ]
    [
    adda r15,r10             ; C output pointer (long)
    move.l d12,(sp-64)       ; Set norm to stack
    ]
    [
    tfra r10,r11             ; C output
    move.w d12,(r2)+         ; psiScale <- norm
    ]
    [
    adda r15,r11             ; D output pointer (long)
    move.l r2,(sp-68)        ; psiScale
    ]
    [
    tfra r11,r12             ; D output
    ]
    [
    adda r15,r12             ; E output pointer (long)
    move.l #radix5parameter+4,r15  ; Point to the radix-5 parameter array
    ]
 ELSE    ;//IF @DEF('FIX_SCALE')
    ;-- Find max value (real & imag) & stage initialization
    [
    move.l (sp-8),r0    ; input base (A)
    move.l (sp-60),r2   ; DFT point
    ]
    ; 3 STALLS on r2
    [
    move.w (r2),d15     ; DFT point
    move.l (sp-40),r3   ; Stage counter
    ]
    [
    sub #8,d15          ; d15-8
    move.l (sp-28),r2   ; num_radix_offset[0]
    move.l (sp-24),r14  ; num_subgroup
    ]
    [
    asrr #2,d15         ; (d15-8)/4
    dosetup0 radix5_dit_last_stage_loop1
    ]
    [
    adda #8,r0,r1       ; Input address
    move.w #2,n0
    ]
    [
    move.2l (r0)+n0,d0:d1
    move.2l (r1)+n0,d2:d3
    ]
    [
    abs2 d0,d0    abs2 d1,d1
    abs2 d2,d2    abs2 d3,d3
    move.2l (r0)+n0,d4:d5
```

```
move.2l (r1)+n0,d6:d7
]
[
abs2 d4,d4   abs2 d5,d5
abs2 d6,d6   abs2 d7,d7
doensh0 d15         ; loop count for max search
]

; Loop for max (real,imag) search
loopstart0
    [
    max2 d0,d4   max2 d1,d5
    max2 d2,d6   max2 d3,d7
    move.2l (r0)+n0,d0:d1
    move.2l (r1)+n0,d2:d3
    ]
    [
    abs2 d0,d0   abs2 d1,d1
    abs2 d2,d2   abs2 d3,d3
    ]
loopend0
[
max2 d0,d4   max2 d1,d5
max2 d2,d6   max2 d3,d7
adda r3,r2               ; num_radix_offset[s]
adda r3,r14              ; siNumSubgroup[s]
]
[
tfr d6,d0
tfr d7,d1
move.w (r2),r15          ; num_radix_offset[s]
move.l (sp-12),r9        ; A output pointer
]
[
max2 d0,d4
max2 d1,d5
move.l (sp-8),r0         ; A input pointer
move.l (sp-68),r2        ; psiScale
]
[
tfr d5,d0
move.l (sp-44),r4        ; r4 -> Wb
move.l (sp-48),r5        ; r5 -> Wc
]
[
max2 d0,d4
move.l (sp-52),r6        ; r6 -> Wd
move.l (sp-56),r7        ; r7 -> We
]
[
sxt.w d4,d0
tfra r9,r8               ; A output
adda r15,r9              ; B output pointer (long)
]
[
extract #16,#16,d4,d4
adda #4,r0,r1            ; B input
```

```
        tfra r9,r10                 ; B output
        ]
        [
        max d0,d4                   ; Max real/imag
        move.w (r14),d5             ; siNumSubgroup[s]
        adda r15,r10                ; C output pointer (long)
        ]
        [
        clb d4,d0                   ; count reading bit
        tfra r10,r11                ; C output
        doen0 d5                    ; Loop for num_butterfly
        ]
        [
        neg d0                       ; negates
        adda r15,r11                 ; D output pointer (long)
        move.l #radix5parameter,r14   ; Point to the radix-5 parameter array
        ]
        [
        sub #16,d0                   ; limit 16-bit
        move.w #S52,d2
        tfra r11,r12                 ; D output
        ]
        [
        cmpgt.w #S51,d0              ; Check if norm > S or norm <= S
        move.w #S51,d1
        adda r15,r12                 ; E output pointer (long)
        ]
        [
        iff sub d0,d1,d12           ; scale down = S - d0 if norm <=  S
        ift sub d0,d2,d12           ; scale up   = S - d0 if norm > S
        ]
        [
        move.l d12,(sp-64)          ; Set norm to stack
        move.w d12,(r2)+            ; psiScale <- norm
        ]
        [
        move.l r2,(sp-68)           ; psiScale
        move.l #radix5parameter+4,r15   ; Point to the radix-5 parameter array
        ]
     ENDIF     ;// IF @DEF('FIX_SCALE')

        ;--- Start Radix-5 (Last stage)

        [ ;
        move.2f (r0)+n0,d12:d13      ; d12.H = Ar, d13.H = Ai
        move.l (r1)+n0,d14           ; d14 = Br:Bi
        ]
        [ ;
        move.l (r0)+n0,d15           ; d15 = Cr:Ci
        move.l (sp-64),d8            ; Scaling factor
        ]
        [ ;
        asrr2 d8,d14               ; d14 = Br:Bi>>S
        asrr2 d8,d15               ; d15 = Cr:Ci>>S
        ]
        [ ;
        asrr d8,d12                   ; d12 = Ar>>S
```

```
    move.l (r0)+,d9             ; d9  = Er:Ei
    move.l (r4)+,d2             ; d2  = Wbr:Wbi
    ]
    FALIGN
radix5_dit_last_stage_loop1:     ; --> Butterflies/Subgroup
    loopstart0
        [ ; 01
        asrr2 d8,d9                 ; d9 = Er:Ei>>S
        MPYRE d2,d14,d6             ; d6  = Br' = (Br*Wbr - Bi*Wbi)
        MPYIM d2,d14,d7             ; d7  = Bi' = (Br*Wbi + Bi*Wbr)
        tfr d8,d1                   ; Scaling factor
        move.l (r1)+n0,d8           ; d8 = Dr:Di
        move.l (r5)+,d10            ; d10 = Wcr:Wci
        ]
        [ ; 02
        asrr2 d1,d8                 ; d8 = Dr:Di>>S
        pack.2f d7,d6,d0
        MPYRE d10,d15,d14           ; d14 = Cr' = (Cr*Wcr - Ci*Wci)
        MPYIM d10,d15,d15           ; d15 = Ci' = (Cr*Wci + Ci*Wcr)
        move.l (r7)+,d3             ; d3  = Wer:Wei
        ]
        [ ; 03
        asrr d1,d13                 ; d13 = Ai>>S
        mac2assar d9,d3,d0.h,d6     ; M2_1 = (Br'+Er'):(Br'-Er')
        mac2aassi d9,d3,d0.l,d7     ; M2_2 = (Bi'+Ei'):(Bi'-Ei')
        pack.2f d15,d14,d0
        move.l (r6)+,d11            ; d11 = Wdr:Wdi
        ]
        [ ; 04
        sod2aaii d12,d6,d4          ; Ar'+Br'+Er'
        sod2aaii d13,d7,d5          ; Ai'+Bi'+Ei'
        mac2assar d8,d11,d0.h,d14   ; M2_3 = (Cr'+Dr'):(Cr'-Dr')
        mac2aassi d8,d11,d0.l,d15   ; M2_4 = (Ci'+Di'):(Ci'-Di')
        move.2f (r14),d0:d1         ; d0:d1 = [a,b], Radix-5 parameters
        move.2f (r15),d8:d9         ; d8:d9 = [c,d], Radix-5 parameters
        ]
        [ ; 05
        move.l d6,d2
        move.l d7,d3
        sod2aaii d4,d14,d4         ; Aout_r = Ar'+Br'+Cr'+Dr'+Er'
        sod2aaii d5,d15,d5         ; Aout_i = Ai'+Bi'+Ci'+Di'+Ei'
        ]
        [ ; 06
        mac +d0,d2,d12             ; Ar' +  a*(Br'+Er')
        mac +d0,d3,d13             ; Ai' +  a*(Bi'+Ei')
        tfr d12,d14
        tfr d13,d15
        move.l d14,d10
        move.l d15,d11
        ]
        [ ; 07
        mac -d8,d10,d12            ; Ar' + (a*(Br'+Er')-c*(Cr'+Dr'))
        mac -d8,d11,d13            ; Ai' + (a*(Bi'+Ei')-c*(Ci'+Di'))
        mac -d8,d2,d14             ; Ar' -  c*(Br'+Er')
        mac -d8,d3,d15             ; Ai' -  c*(Bi'+Ei')
        adda #4,r0,r1              ; B input
        moves.2f d4:d5,(r8)+       ; out:[Aout_r,Aout_i]
        ]
        [ ; 08
        pack.2f d13,d12,d0
        mac +d0,d10,d14           ; Ar' - (c*(Br'+Er')-a*(Cr'+Dr'))
        mac +d0,d11,d15           ; Ai' - (c*(Bi'+Ei')-a*(Ci'+Di'))
        move.l (sp-64),d8         ; Scaling factor
        move.2f (r0)+n0,d12:d13   ; d12.H = Ar, d13.H = Ai
        ]
        [ ; 09
        mac2aassi d1,d3,d0.h,d4   ; M5_1.H = Ar'+(a(Br'+Er')-c(Cr'+Dr'))+(b(Bi'-Ei')),
                                  ; M5_1.L = Ar'+(a(Br'+Er')-c(Cr'+Dr'))-(b(Bi'-Ei'))
        mac2ssaai d1,d2,d0.l,d5   ; M5_2.H = Ai'+(a(Bi'+Ei')-c(Ci'+Di'))-(b(Br'-Er')),
                                  ; M5_2.L = Ai'+(a(Bi'+Ei')-c(Ci'+Di'))+(b(Br'-Er'))
        pack.2f d15,d14,d0
        move.l (r1)+n0,d14        ; d14 = Br:Bi
        move.l (r0)+n0,d15        ; d15 = Cr:Ci
        ]
        [ ; 10
        mac2aassi d9,d11,d4        ; Bout_r =
                                   ; Ar'+(a(Br'+Er')-c(Cr'+Dr'))+
                                   ; (b(Bi'-Ei')+d(Ci'-Di')), Eout_r =
                                   ; Ar'+(a(Br'+Er')-c(Cr'+Dr'))-
                                   ; (b(Bi'-Ei')+d(Ci'-Di'))
        mac2ssaai d9,d10,d5        ; Bout_i =
                                   ; Ai'+(a(Bi'+Ei')-c(Ci'+Di'))-
                                   ; (b(Br'-Er')+d(Cr'-Dr')), Eout_i =
                                   ; Ai'+(a(Bi'+Ei')-c(Ci'+Di'))+
                                   ; (b(Br'-Er')+d(Cr'-Dr'))
        mac2aassi d9,d3,d0.h,d6    ; M5_3.H = Ar'-(c(Br'+Er')-a(Cr'+Dr'))+(d(Bi'-Ei')),
                                   ; M5_3.L = Ar'-(c(Br'+Er')-a(Cr'+Dr'))-(d(Bi'-Ei'))
        mac2ssaai d9,d2,d0.l,d7    ; M5_4.H = Ai'-(c(Bi'+Ei')-a(Ci'+Di'))-(d(Br'-Er')),
                                   ; M5_4.L = Ai'-(c(Bi'+Ei')-a(Ci'+Di'))+(d(Br'-Er'))
        move.l (r0)+,d9            ; d9 = Er:Ei
        move.l (r4)+,d2            ; d2 = Wbr:Wbi
        ]
        [ ; 11
        asrr2 d8,d14               ; d14 = Br:Bi>>S
        asrr2 d8,d15               ; d15 = Cr:Ci>>S
        mac2saasi d1,d11,d6        ; Cout_r =
                                   ; Ar'-(c(Br'+Er')-a(Cr'+Dr'))+
                                   ; (d(Bi'-Ei')-b(Ci'-Di')), Dout_r =
                                   ; Ar'-(c(Br'+Er')-a(Cr'+Dr'))-
                                   ; (d(Bi'-Ei')-b(Ci'-Di'))
        mac2assai d1,d10,d7        ; Cout_i =
                                   ; Ai'-(c(Bi'+Ei')-a(Ci'+Di'))-
                                   ; (d(Br'-Er')-b(Cr'-Dr')), Dout_i =
                                   ; Ai'-(c(Bi'+Ei')-a(Ci'+Di'))+
                                   ; (d(Br'-Er')-b(Cr'-Dr'))
        moves.2f d4:d5,(r9)+       ; out:[Bout_r,Bout_i]
        move.2w d4:d5,(r12)+       ; out:[Eout_r,Eout_i]
        ]
        [ ; 12
        asrr d8,d12                ; d12 = Ar>>S
        moves.2f d6:d7,(r10)+      ; out:[Cout_r,Cout_i]
        move.2w d6:d7,(r11)+       ; out:[Dout_r,Dout_i]
        ]
    loopend0
```

```
exit_mixed_radix:
    [
    move.l (my_OUTPUT_pointer),r14 ;// --> psFft->psiOut
    move.l (sp-12),d3              ; out_address psRadix.psiOut
    ]
[
move.l (sp-36),d4
adda #-72,sp,r5
]
[
tfra r5,sp
]

    pop.2l d6:d7   pop.2l r6:r7
F_sc3850_dft_dit_complex_16x16_auto_scale_asm_end
[
rtsd
move.l d4,sr
]
move.l d3,(r14)                    ; psFft->psiOut = psRadix.psiOut;
endsec
```

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or
+1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064
Japan
0120 191014 or
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 010 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
+1-800 441-2447 or
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Document Number: AN3680
Rev. 0
11/2010