Analog Devices公司DSP、处理器和开发工具的使用技术指南
联系我们的技术支持中心：dsp.support@analog.com 和 dsptools.support@analog.com
查看我们的在线资源：http://www.analog.com/ee-notes 和 http://www.analog.com/processors

TigerSHARC®处理器上定点FFT的并行实现

Boris Lernerr提供

第一版-2005年2月3日

引言

现代的高度并行式处理器的进展如Analog Devices的TigerSHARC®系列处理器，要求寻找更为有效的方法来并行实现许多标准算法的操作。该应用手记不仅解释最快的16位FFT如何在TigerSHARC上的实现，而且提供指导算法的开发，使你能够将同一技术施于其他算法。一般来说，大多数算法有几个层次的优化级，该手记将给予详细讨论。第一和最直截了当的优化级是指令的并行，这是处理器架构所允许的。这种工作是简单而又烦人的。优化的第二级是循环体展开（loop unrolling）和软件的流水线操作，以取得最大并行性，避免流水线停止工作。虽然比简单的一级并行复杂，但可按描述的步骤进行工作，无需深入了解算法，几乎没有独创性。第三级是重建算法的数学操作，仍产生有效的结果，但更适合处理器的架构。这需要彻底了解算法，不像软件的流水线操作那样，它没有既定的引导你走向最佳方案的步骤。这在写最佳代码中是最有趣的。

在实际应用中，常常不需要用到所有的优化层次。当所有的优化级需要时，最好以反向的顺序进行这些级的优化。在代码完全被流水化操作以后，再来尝试改变基本的底层算法已经太迟了。由此，程序师必须要先考虑算法结构，随后组织代码。然后通常将优化层次1和2（并行、展开以及流水线操作）同时进行。

本手记中所参考使用的代码由模拟器件公司提供。具体例子使用一个256-点FFT，但其中的数学算法和理念同样适于其它大小（不小于16点）的变换。

如同所见，重建的算法将FFT打散成更小的部分而后可被并行。在256点FFT（其代码列表在该应用手记的尾部）的情形下，FFT被分成一个个16点的FFT有16个，16点FFT以基数4（radix-4）的格式来完成（即每个只有两个阶）。如果我们做一个512=点 FFT，我们将不得不一次做16个32点的FFT（或者一次做32个16点的FFT）,每个32点FFT以基数4格式先完成前两个阶，最后一阶做成基数2格式。这种不同意味着书写FFT大小统配（FFT size-generic）的代码是困难的。虽然实现的算法是通用的，可同等地适于所有大小的FFT，但代码却不能这样，它必须针对每一种点大小FFT进行手工调整，以便能够完全达到最优化。带上这些一并考虑，让我们进入TigerSHARC园地里迷人的定点FFT世界吧。

标准的基数2 FFT算法

图1给出了一种输入经过位反转（bitreversed）后的标准16点基数-2 FFT实现方案。传统来看，在这一算法中，阶1和2与所要求的位反转结合成一个单一的优化循环。（由于两个阶不需要乘法运算，只进行加和减操作）。每个余留的阶通常将共享相同旋移因数的蝶形运算结合在一起组成群组（对于每一群组，旋移因子仅需读取一次）。

Copyright 2003, Analog Devices公司。版权全部所有。Analog Devices 公司不为用户的产品设计或用户产品的使用或应用以及由ADI协助所可能产生的对任何专利权或其他人权利的侵犯而承担任何责任。所有商标和标识所有权均属于其各自持有者。Analog Devices 公司应用和开发工具工程师所提供的资料均视为准确、可靠。但Analog Devices公司不为本公司的Engineer-to-Engineer 指南所提供内容的技术准确性承担任何责任。

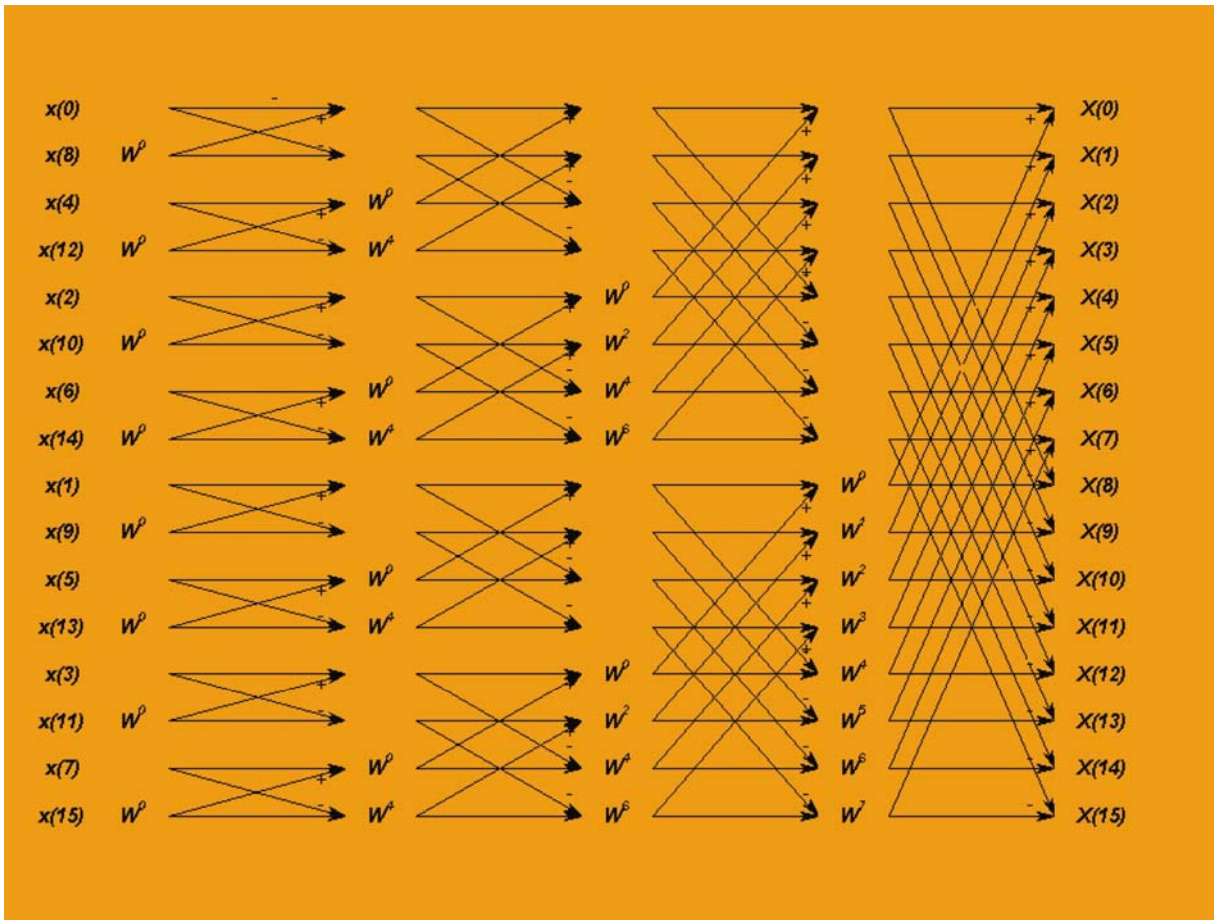


图1　16点FFT的标准结构

由于TigerSHARC处理器对打包数据提供矢量化的16位处理，因此我们需要把该算法并行到至少和TigerSHARC能处理的同等多的并行过程中。TigerSHARC的一个加/减指令（有助于进行一个基本蝶形的计算）可以以每周期8个16-bit值进行并行操作（TigerSHARC 处理器的每个compute block有4个）。由于数据为复数，这就相当于每周期有4个数据加/减操作。因此，我们需要把FFT打分成至少4个并行过程。见图1中的图表，显然，我们可以通过简单地把数据组合成块来进行，例如一次4点：

1st block = {x(0), x(8), x(4), x(12)}
2nd block = {x(2), x(10), x(6), x(14)}
3rd block = {x(1), x(9), x(5), x(13)}
4th block = {x(3), x(11), x(7), x(15)}

这些群组没有相互依赖性，针对FFT的头两个阶会并行得很好。之后我们将遇到麻烦，并行性失去。但是此时，我们可以将数据重组成不同的块以确保以后的工作中新块相互不干扰，从而做到可并行处理。仔细观察显示，所需要的重组是一种以新块为基础的交插（或解交插）操作，新块由如下给出：

1st block = {x(0), x(2), x(1), x(3)}
2nd block = {x(8), x(10), x(9), x(11)}
3rd block = {x(4), x(6), x(5), x(7)}
4th block = {x(12), x(14), x(13), x(15)}

对待这些新块的另一种方法是4x4矩阵变换（块定义矩阵的行）。当然，有严重的副作用——当数据重组后，最后的阶将并行，但会不以正

确的顺序产生数据。或许，我们以有别于我们以位反转为开始的一种顺序开始，可以对此有所补偿，但还是让我们把这里面的细节留给后面严密的数学分析吧。

这时，对16点FFT的分析似乎建议我们，通常在给定一个N点FFT的情况下，我们需要把它看成一个二维的数据 $\sqrt{N}\times\sqrt{N}$ 矩阵，进行行或列的并行处理，然后变换矩阵，再次并行处理行或列。由此引出的另一个要求是N必须是一个完全平方数。如所证，我们可以对这一要求进行处理，这将在后面讨论。此刻我们所关心的是256点FFT, 256 =16^2。

于是，我们将并行哪一个？是行还是列？答案在TigerSHARC处理器的向量架构中。当TigerSHARC处理器从存储器中获取数据时，它一次获取128位（即4个16位的复数数据点）的大块数据，并把它打包到四个或两个（对于SIMD取数来说）寄存器中。而后它沿四个或两个寄存器进行矢量化处理。这样，我们想要并行的是矩阵的列（即，我们需要构建我们的数学计算，以使矩阵的所有列相互无关联性）。

现在我们知道了我们想要数学计算带给我们什么，这时也是严格按数学语言做工作时候了。

算法的数学原理

下列符号将使用：

N =原FFT的点数（例中为256点），

$M=\sqrt{N}$,

$\hat{x}$ 将代表离散傅立叶变换（Discrete Fourier Transform，简称为DFT）。现在，已知信号x,

$$\hat{x}(n)=\sum_{k=0}^{N-1}x(k)e^{\frac{-2\pi nk}{N}}=\sum_{m=0}^{M-1}\sum_{l=0}^{M-1}x(Ml+m)e^{\frac{-2\pi n(Ml+m)}{N}}=$$
$$\sum_{m=0}^{M-1}e^{\frac{-2\pi nm}{N}}\sum_{l=0}^{M-1}x(Ml+m)e^{\frac{-2\pi nl}{M}}=\sum_{m=0}^{M-1}e^{\frac{-2\pi nm}{N}}\hat{x}_m(n)$$

where:

$$x_m(l):=x(Ml+m) \qquad (1)$$

这里 $\hat{x}_m$ 是这种*M*点DFT的函数，我们把输出指数n看作安排在 $M\ x\ M$ 矩阵中（即 $n=Ms+t,\ 0\le s,t<M-1$ ）。，由此，

$$\hat{x}(Ms+t)=\sum_{m=0}^{M-1}e^{\frac{-2\pi i(Ms+t)m}{N}}\hat{x}_m(Ms+t)=$$
$$\sum_{m=0}^{M-1}e^{\frac{-2\pi ism}{M}}e^{\frac{-2\pi itm}{N}}\hat{x}_m(t)$$

由于 $\hat{x}_m$ 是一个 *M*点DFT,它具有周期=*M*的周期性。因此，

$$\hat{x}(Ms+t)=\sum_{m=0}^{M-1}e^{\frac{-2\pi ism}{M}}x_t^*(m)=\hat{x}_t^*(s), \qquad (2)$$

where:

$$x_t^*(m):=e^{\frac{-2\pi itm}{N}}\hat{x}_m(t) \qquad (3)$$

这里 $\hat{x}_t^*$ 是这个*M*点DFT的函数。

算法的实现

方程式(1), (2) 和 (3)示出怎样用下列步骤计算x的DFT（这时我们回到我们的具体例子*N*=256, *M*=16中）：

1. 线性安排输入数据x(n)的256点，但把它看作一个16x16矩阵：

$$\begin{bmatrix} x(0) & x(1) & x(2) & \cdots & x(15) \\ x(16) & x(17) & x(18) & \cdots & x(31) \\ x(32) & x(33) & x(34) & \cdots & x(47) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x(240) & x(241) & x(242) & \cdots & x(255) \end{bmatrix}$$

2．用公式（1），再写为：

$$\begin{bmatrix} x_0(0) & x_1(0) & x_2(0) & \cdots & x_{15}(0) \\ x_0(1) & x_1(1) & x_2(1) & \cdots & x_{15}(1) \\ x_0(2) & x_1(2) & x_2(2) & \cdots & x_{15}(2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_0(15) & x_1(15) & x_2(15) & \cdots & x_{15}(15) \end{bmatrix}$$

3．我们现在按列计算并行FFT（如前所述，TigerSHARC处理器做此很有效）获得：

$$\begin{bmatrix} \hat{x}_0(0) & \hat{x}_1(0) & \hat{x}_2(0) & \cdots & \hat{x}_{15}(0) \\ \hat{x}_0(1) & \hat{x}_1(1) & \hat{x}_2(1) & \cdots & \hat{x}_{15}(1) \\ \hat{x}_0(2) & \hat{x}_1(2) & \hat{x}_2(2) & \cdots & \hat{x}_{15}(2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \hat{x}_0(15) & \hat{x}_1(15) & \hat{x}_2(15) & \cdots & \hat{x}_{15}(15) \end{bmatrix}$$

4. 我们对此作逐点（point-wise）矩阵乘

$$\left[e^{\frac{-2\pi itm}{256}} \right]_{0 \le t,m \le 15}$$

得到

$$\begin{bmatrix} \hat{x}_0(0)e^{\frac{-2\pi i0}{256}} & \hat{x}_1(0)e^{\frac{-2\pi i0}{256}} & \hat{x}_2(0)e^{\frac{-2\pi i0}{256}} & \cdots & \hat{x}_{15}(0)e^{\frac{-2\pi i0}{256}} \\ \hat{x}_0(1)e^{\frac{-2\pi i0}{256}} & \hat{x}_1(1)e^{\frac{-2\pi i1}{256}} & \hat{x}_2(1)e^{\frac{-2\pi i2}{256}} & \cdots & \hat{x}_{15}(1)e^{\frac{-2\pi i15}{256}} \\ \hat{x}_0(2)e^{\frac{-2\pi i0}{256}} & \hat{x}_1(2)e^{\frac{-2\pi i2}{256}} & \hat{x}_2(2)e^{\frac{-2\pi i4}{256}} & \cdots & \hat{x}_{15}(2)e^{\frac{-2\pi i30}{256}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \hat{x}_0(15)e^{\frac{-2\pi i0}{256}} & \hat{x}_1(15)e^{\frac{-2\pi i15}{256}} & \hat{x}_2(15)e^{\frac{-2\pi i30}{256}} & \cdots & \hat{x}_{15}(15)e^{\frac{-2\pi i225}{256}} \end{bmatrix}$$

这按照公式（3），它正好是

$$\begin{bmatrix} x_0^*(0) & x_0^*(1) & x_0^*(2) & \cdots & x_0^*(15) \\ x_1^*(0) & x_1^*(1) & x_1^*(2) & \cdots & x_1^*(15) \\ x_2^*(0) & x_2^*(1) & x_2^*(2) & \cdots & x_2^*(15) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{15}^*(0) & x_{15}^*(1) & x_{15}^*(2) & \cdots & x_{15}^*(15) \end{bmatrix}$$

5. 现在我们来计算 $x_t^*(m)$ 的16点FFT，但它们在排列上并行为行而非列。我们必须变换以得到

$$\begin{bmatrix} x_0^*(0) & x_1^*(0) & x_2^*(0) & \cdots & x_{15}^*(0) \\ x_0^*(1) & x_1^*(1) & x_2^*(1) & \cdots & x_{15}^*(1) \\ x_0^*(2) & x_1^*(2) & x_2^*(2) & \cdots & x_{15}^*(2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_0^*(15) & x_1^*(15) & x_2^*(15) & \cdots & x_{15}^*(15) \end{bmatrix}$$

6. 我们按列计算并行FFT，使用公式（2）得到

$$\begin{bmatrix} \hat{x}(0) & \hat{x}(1) & \hat{x}(2) & \cdots & \hat{x}(15) \\ \hat{x}(16) & \hat{x}(17) & \hat{x}(18) & \cdots & \hat{x}(31) \\ \hat{x}(32) & \hat{x}(33) & \hat{x}(34) & \cdots & \hat{x}(47) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \hat{x}(240) & \hat{x}(241) & \hat{x}(242) & \cdots & \hat{x}(255) \end{bmatrix}$$

这是我们想要的FFT结果，它的次序正确！现在数学计算已完成，我们准备开始考虑编程实现。在下列讨论中，我们将参照上面所概括的步骤1到6描述的六步。

编程实现

我们将从头到尾检查前面部分的步骤，一次一步。步骤1和2不需要进行编程。输入数据已经以适当的顺序安排好。

步骤3需要我们按输入矩阵的列计算16个并行的

16点FFT。如前所述，由于其向量处理功能，TigerSHARC可一次轻易并行4个FFT，这样我们可一次做4个FFT，并重复这一过程4次，便可计算全部16个FFT。16点FFT以基数4的方式完成可做得非常有效，结果有2阶，每阶有4个蝶形运算。为了使内部开销最小化，4组4个FFT中，每组都先仅计算第一阶更有效。然后计算4组4个FFT中的每个第二阶（不是顺序计算每组的两阶）。

步骤4是一个逐点式复数乘法（共256次相乘）运算， 步骤5 是一个矩阵变换。这两步可以结合----在进行数据点的乘法运算时，我们可以以转置矩阵的方式存放它们。步骤6 与步骤3一样---我们必须按我们的新输入矩阵的列计算16个并行16 点FFT。这部分不需要书写。我们可以简单地分支到步骤3的代码，记住，一旦FFT完成(而不是像以往继续进入步骤4)，记住退出例行程序。

图2 代表包含数据的缓冲器，并且箭头对应于数据在缓冲器之间的变换。

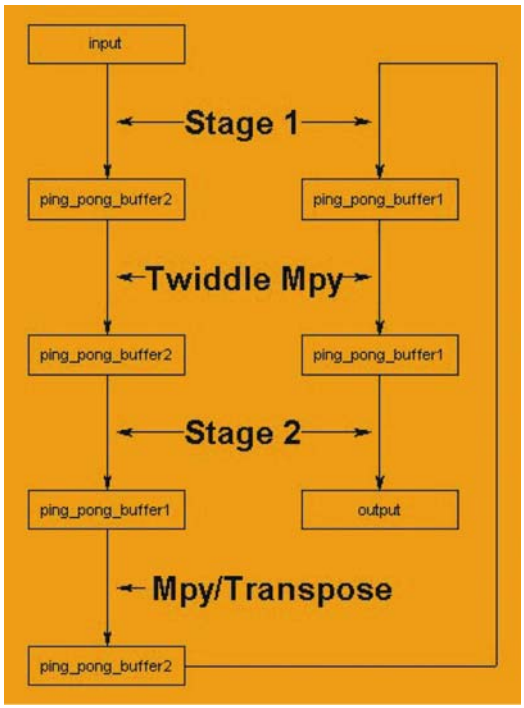


图2 代码的结构图

算法的流水线操作—第一阶段

让我们先集中于步骤3。

记忆存储器	*操作*
F1	*取4个蝶形的4个复数 Input1 =4个 32-bit 值*
F2	*取4个蝶形的4个复数 Input2 =4个 32-bit 值*
F3	*取4个蝶形的4个复数 Input3 =4个 32-bit 值*
F4	*取4个蝶形的4个复数 Input=4个32-bit 值*
AS1	*F1+/-F2*
AS2	*F3+/-F4*
MPY1	*(F1-F2)*(-i) 的前半部分注意我们每个周期只能做2个复数mpys*
M1	*将MPY1移入计算模块寄存器*
MPY2	*(F1-F2)*(-i) 的第二部分注意我们每个周期只能做2个复数mpys*
M2	*将MPY2移入计算模块寄存器*
AS3	*(F3 + F4)+/-(F1+F2) =4个蝶形的Output1和Output3*
AS4	*(F3 - F4)+/-(F1-F2)*(-i) =4个蝶形的Output2和Output 4*
S1	*存储4个蝶形 (Output1) =4个32-bit值*
S2	*存储4个蝶形 (Output2) =4个32-bit值*
S3	*存储4个蝶形 (Output3) =4个32-bit值*
S4	*存储4个蝶形 (Output4) =4个32-bit值*

表1 线性完成阶段1的单一蝶形——逻辑实现

表1列出了以TigerSHARC处理器的向量形式进行阶段1四个并行的基数-4 复数蝶形运算所必要的操作。实际上，这个部分对各阶段来说是相同的，但除非其它阶段在蝶形开始时也要求一个复数的旋移相乘。这使得其它阶段更为复杂，它们将在下一节内容涉及。

Cycle/ Operation	JALU	KALU	MAC	ALU
1	F1	S4---		AS3--
2	F2	S1--	MPY1-	AS2-
3	F3	S2--	M1-, MPY2-	AS4--
4	F4	S3--	M2-	AS1
5	F1+	S4--		AS3-
6	F2+	S1-	MPY1	AS2
7	F3+	S2-	M1, MPY2	AS4-
8	F4+	S3-	M2	AS1+
9	F1++	S4-		AS3
10	F2++	S1	MPY1+	AS2+
11	F3++	S2	M1+, MPY2+	AS4
12	F4++	S3	M2+	AS1++
13	F1+++	S4		AS3+
14	F2+++	S1+	MPY1++	AS2++
15	F3+++	S2+	M1++, MPY2++	AS4+
16	F4+++	S3+	M2++	AS1+++

表2　流水线操作的蝶形——阶段1

表2示出流水线操作的蝶形。操作中的一个"+"符号表示对应于下一组蝶形的操作，一个"-"号对应于上一组蝶形中的操作。所有指令都是并行的，并且没有阻塞（stall）。

算法的流水线操作– 阶段2

做一个阶段2的蝶形运算，必须进行和阶段1相同的计算，除非开始时16 (4个并行的x 4 点) 个输入中的每一个都存在一个附加的复数相乘操作。这会产生一个问题。原有的蝶形中已有二个SIMD 复数相乘运算。当ALU 、取数和存储保留在4时，又增加8 个便构成10 个复数相乘。这会使算法失衡，太多的乘法和太少的其他计算单元将不会很好地并行。这样做的最好的办法是每蝶形向量10个周期。结果，两个原有乘运算(是乘以-i) 中的每一个便可由一个短的ALU (逻辑“非”)和一个长循环替代。另外，需要二个寄存器移位，以保证数据返回到长寄存器中被打包，以进行随后的并行加/减运算。对向量式蝶形运算来说，这会留下共计8个乘法、6 个ALU (4 个加/减 和2个逻辑“非”)以及2个移位(循环移)——对于一个8 周期的操作执行是完美的平衡。还有，4取数和4存储为寄存器移位留下了大量的空间。

表3 列出了在TigerSHARC 处理器进行第二阶段向量式(即四个并行) 基数-4 复数蝶形运算所需要的操作。事情变得大大地复杂化了!

表4示出流水线操作的蝶形。一个"+"符号表示对应于下一组蝶形的操作，一个“-”号对应于上一组蝶形的操作。

记忆存储器	操作
F1	取4个蝶形中的4个复数 Input1 =4个32-bit 值
F2	取4个蝶形中的4个复数 Input2 =4个32-bit 值
F3	取4个蝶形中的4个复数 Input3 =4个32-bit 值
F4	取4个蝶形中的4个复数 Input4=4个32-bit 值
MPY1	F1*旋移的第一个一半
M1	移动MPY1到计算模块寄存器
MPY2	F1*旋移的第二个一半

M2	移动MPY2到计算模块寄存器
MPY3	F2*旋移的第一个一半
M3	移动MPY3到计算模块寄存器
MPY4	F2*旋移的第二个一半
M4	移动MPY4到计算模块寄存器
MPY5	F3*旋移的第一个一半
M5	移动MPY5到计算模块寄存器
MPY6	F3*旋移的第二个一半
M6	移动MPY6到计算模块寄存器
MPY7	F4*旋移的第一个一半
M7	移动MPY7到计算模块寄存器
MPY8	F4*旋移的第二个一半
M8	移动MPY8到计算模块寄存器
AS1	M1,M2+/-M3,M4
AS2	M5,M6+/-M7,M8
A1	逻辑非(M1-M3)
MV1	移动(M1-M3)到一对含有A1的寄存器中
R1	旋动A1的长结果，MV1-现在低寄存器含有(M1-M3)*(-i)
A2	逻辑非 (M2-M4)
MV2	移动 (M2-M4)到一对含A2的寄存器中
R2	旋动A1的长结果，MV2－现在低寄存器含有 (M2-M4)*(-i)
AS3	(F3+F4)+/-(F1+F2)
AS4	(F3 - F4)+/ - (F1-F2)*(-i)这里 (F1-F2)*(-i)由R1和R2获得
S1	Store 1 存储4个蝶形=4个32-bit值
S2	Store 2 存储4个蝶形=4个32-bit值
S3	Store3 存储4个蝶形=4个 32-bit值
S4	Store4 存储4个蝶形=4个 32-bit值

表3 线性完成阶段2的单一蝶形——逻辑实现

所有指令都是并行的，并且没有阻塞。仍有一个旋移取数问题存在，未得到解决，但还有许多JALU 和KALU 指令时段（slot）可利用，依进度安排旋移取数不会造成任何问题(它们实际上具有相同的单位向量值，因此广播读取将带给它们高效率）。

Cycle/ Operation	JALU	KALU	MAC	ALU
1	F1	MV1--	M3-, MPY4-	A1--
2	F2		M4-, MPY5-	A2--
3		S3---	M5-, MPY6-	R1--
4		S4---	M6-, MPY7-	R2--
5	F3	MV2--	M7-, MPY8-	AS3--
6	F4		M8-, MPY1	AS1-
7		S1--	M1, MPY2	AS4--
8		S2--	M2, MPY3	AS2-
9	F1+	MV1-	M3, MPY4	A1-
10	F2+		M4, MPY5	A2-
11		S3--	M5, MPY6	R1-
12		S4--	M6, MPY7	R2-
13	F3+	MV2-	M7, MPY8	AS3-
14	F4+		M8, MPY1+	AS1
15		S1-	M1+, MPY2+	AS4-
16		S2-	M2+, MPY3+	AS2
17	F1++	MV1	M3+, MPY4+	A1
18	F2++		M4+, MPY5+	A2
19		S3-	M5+, MPY6+	R1
20		S4-	M6+, MPY7+	R2
21	F3++	MV2	M7+, MPY8+	AS3
22	F4++		M8+, MPY1++	AS1+

23		S1	M1++, MPY2++	AS4
24		S2	M2++, MPY3++	AS2+
25	F1+++	MV1+	M3++, MPY4++	A1+
26	F2+++		M4++, MPY5++	A2+
27		S3	M5++, MPY6++	R1+
28		S4	M6++, MPY7++	R2+

表4 流水线操作的的蝶形——阶段2

步骤4 、5 的相乘和变换对于流水线操作来说是非常简单的。它们仅涉及取数、 相乘和存储，因此算法中的这些部分的流水线操作在这里不加详细讨论。

代码

现在，写代码就变得容易了。ADSPTS201 TigerSHARC 处理器是很灵活的，它可排除各种挑战。只需遵循表2和表4 的流水线操作，代码就可以完成。最终代码示于列表1 中。

现在，最低限度---周期计数得到了多大的改进？ 表5 列出了新旧两种16位复数输入FFT的实现方案。如表所示，周期计数大为改善。

点N	旧的实现方案	新的实现方案
64	302	147
256	886	585
1024	3758	2725
2048	7839	5776
4096	16600	12546

表5 N 点16 位复数FFT的内核时钟周期

使用规则

C可调用复数FFT例行程序称为

```
fft256pt(&(input),&(ping_pong_buffer1),
        &(ping_pong_buffer2), &(output));
```

这里:

输入-> FFT 输入缓冲器，

输出-> FFT 输出缓冲器，

ping_pong_bufferx 是乒乓缓冲（ping pong）器。

所有缓冲器是以正常(没有位反转的) 的顺序进行复数值打包。

ping_pong_buffer1和ping_pong_buffer2必须是二个不同的缓冲器。但是，根据用户要求，进行一些内存优化是可能的。ping_pong_buffer1 在*input*不需要保留时可以同*input*一样施为。同理，*output*可以和ping_pong_buffer2一样施为。下面是最少内存使用情况下的例行程序用法例子:

```
fft256pt (&(input), & (输入),
        & (产品), &(output));
```

为消除存储块访问冲突，input 和ping_pong_buffer1 必须驻留在与ping_pong_buffer2及output不同的存储块中，而旋转因子必须驻留在一个与ping-pong缓冲器不同的存储块中。当然，所有代码必须同样驻留在一个与所有数据缓冲器都不同的数据块中。

后记

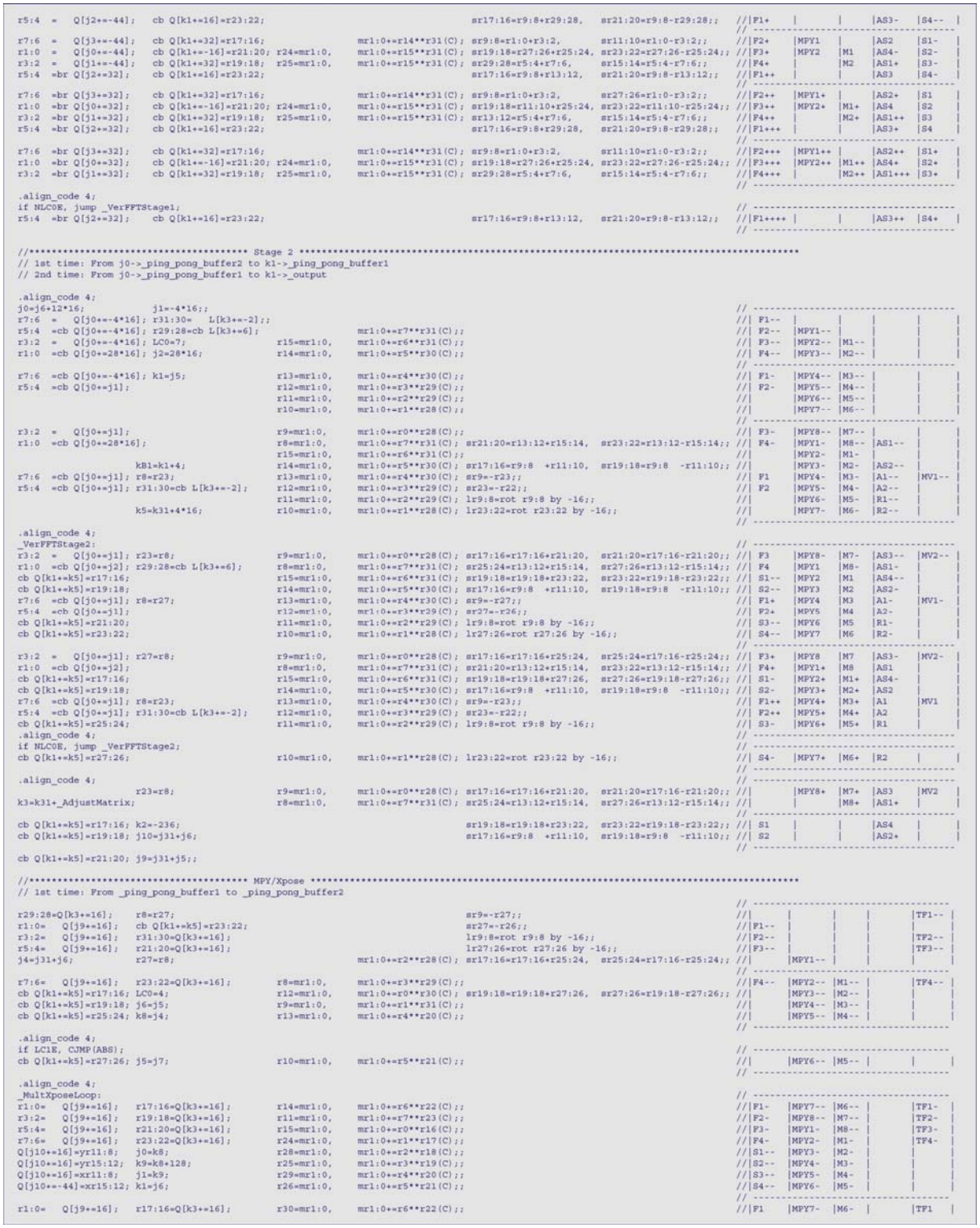
文中所考察的例子是一个256点FFT。在撰写本手记时，使用上述算法的64点、1024点、2048点和4096点FFT的例子已书写完成。在这些例子中，FFT被分别视作8x8，32x32、32x64和64x64矩阵来讨论。32点FFT以基数-4为基础完成（所有通向最后一阶的方式），最后一阶以传统的基数2方式完成。

2048点FFT按32列、64列的矩阵进行排列。32个64点FFT中每一个都按列进行并行运算。使用一个逐点式相乘和变换，得到一个64列、32行的矩阵。按列并行运算64个32点FFT中的每一个，完成算法。惟一的副作用是代码的并行FFT部分不能够被重用（记住，算法需要这样做两次），因为行和列的数量不再相同。这将导致更长的源代码，但周期计数效率刚刚一样好。

附录

优化FFT的完整源代码

```
/*****************************************************************************************************************************

        fft256pt.asm

        Prelim rev.         August 10, 2004     BL

        This is assembly routine for the complex C-callable 256-point 16-bit FFT on
        TigerSHARC family of DSPs.

        I. Description of Calling.

                1. Inputs:
                        j4 -> input
                        j5 -> ping_pong_buffer1
                        j6 -> ping_pong_buffer2
                        j7 -> output

                2. C-Calling Example:
                        Fft256pt(&(input), &(ping_pong_buffer1), &(ping_pong_buffer2), &(output));

                3. Limitations:
                        a. All buffers must be aligned on memory boundary which is a multiple of 4.
                        b. Buffers input.and ping_pong_buffer2 must be aligned on memory boundary
                           which is a multiple of 256.
                        c. If memory space savings are required and input does not have to be
                           preserved, ping_pong_buffer1 can be the same buffer as input with no
                           degradation in performance.
                        d. If memory space savings are required, output can be the same buffer
                           as ping_pong_buffer2 with no degradation in performance.

                4. For the code to yield optimal performance, the following must be observed:
                        a. Buffer input must have been cached previously. This is reasonable to
                           assume since any engine that would have brought the data into internal
                           memory, such as a DMA, would also have cached it.
                        b. input and ping_pong_buffer2 must be located in different memory blocks.
                        c. ping_pong_buffer1 and ping_pong_buffer2 must be located in different
                           memory blocks.
                        d. ping_pong_buffer1 and output must be located in different memory blocks.
                        e. twiddles and input must be located in different memory blocks.
                        f. AdjustMatrix and ping_pong_buffer1 must be located in different memory
                           blocks.

        II. Description of the FFT algorithm.

                1. All data is treated as complex packed data.
                2. An application note will be provided for the description of the math of
                   the algorithm.

*****************************************************************************************************************************/

//************************ Includes ******************************************************************************************

#include <defTS201.h>

//****************************************************************************************************************************
.section data6a;
.align 4;                                                                                                   // allign to quad
.var _AdjustMatrix[256] = "MatrixCoeffs.dat";

.align 4;                                                                                                   // allign to quad
.var _twiddles16[32] = "Twiddles16.dat";

//****************************************************************************************************************************
.section program;
.global _fft256pt;

//************************************ Start of code ************************************************************************
_fft256pt:

j2=j4+64;;
j0=j4+0;                kl=j6;;
j3=j4+(128+64);         LC1=2;;                                                                             // ------------------------------------
r5:4  =br Q[j2+=32];    k3=k31+(_twiddles16+2);;                                                            //|F1--  |      |     |      |      |
j1=j4+128;;                                                                                                 // ------------------------------------

.align_code 4;
_VerticalLoop:

//************************************ Stage 1 ******************************************************************************
// 1st time: From j0,j1,j2,j3->_input to k1->_ping_pong_buffer2
// 2nd time: From _ping_pong_buffer2 to _ping_pong_buffer1
                                                                                                            // ------------------------------------
r7:6  =br Q[j3+=32];    kL1=k31+252;;                                                                       //|F2--  |      |     |      |      |
r1:0  =br Q[j0+=32];    r31=0x80000000;;                                                                    //|F3--  |      |     |      |      |
r3:2  =br Q[j1+=32];    kB3=k31+_twiddles16;                                sr13:12=r5:4+r7:6,     sr15:14=r5:4-r7:6;;  //|F4--  |      |     |AS1-- |      |
r5:4  =br Q[j2+=32];    kB1=k1+4;;                                                                          //|F1-  |      |     |      |      |
                                                                                                            // ------------------------------------
r7:6  =br Q[j3+=32];    jL0=252;                    mr1:0+=r14**r31(C); sr9:8=r1:0+r3:2,       sr11:10=r1:0-r3:2;;  //|F2-   |MPY1-- |     |AS2-- |      |
r1:0  =br Q[j0+=32];    kL3=k31+32;         r24=mr1:0,    mr1:0+=r15**r31(C);;                              //|F3-   |MPY2-- |M1-- |      |      |
r3:2  =br Q[j1+=32];    LC0=4;              r25=mr1:0,    mr1:0+=r15**r31(C); sr29:28=r5:4+r7:6,     sr15:14=r5:4-r7:6;;  //|F4-   |       |M2-- |AS1-  |      |
r5:4  =br Q[j2+=32];    jB0=kB1;                                            sr17:16=r9:8+r13:12,   sr21:20=r9:8-r13:12;;//|F1    |       |     |AS3-- |      |
                                                                                                            // ------------------------------------
.align_code 4;
_VerFFTStage1:                                                                                              // ------------------------------------
r7:6  =br Q[j3+=32];    cb Q[k1+=32]=r17:16;        mr1:0+=r14**r31(C); sr9:8=r1:0+r3:2,       sr27:26=r1:0-r3:2;;  //|F2    |MPY1-  |     |AS2-  |S1--  |
r1:0  =br Q[j0+=32];    cb Q[k1+=-16]=r21:20; r24=mr1:0,    mr1:0+=r15**r31(C); sr19:18=r11:10+r25:24, sr23:22=r11:10-r25:24;; //|F3    |MPY2-  |M1-  |AS4-- |S2--  |
r3:2  =br Q[j1+=32];    cb Q[k1+=32]=r19:18;  r25=mr1:0,    mr1:0+=r15**r31(C); sr13:12=r5:4+r7:6,     sr15:14=r5:4-r7:6;;  //|F4    |       |M2-  |AS1   |S3--  |
```



```
r5:4  =   Q[j2+=-44];   cb Q[k1+=16]=r23:22;                                    sr17:16=r9:8+r29:28,   sr21:20=r9:8-r29:28;;  //|F1+    |       |     |AS3-   |S4--  |
                                                                                                                            // ------------------------------------
r7:6  =   Q[j3+=-44];   cb Q[k1+=32]=r17:16;              mr1:0+=r14**r31(C); sr9:8=r1:0+r3:2,       sr11:10=r1:0-r3:2;;    //|F2+    |MPY1   |     |AS2    |S1-   |
r1:0  =   Q[j0+=-44];   cb Q[k1+=-16]=r21:20; r24=mr1:0,  mr1:0+=r15**r31(C); sr19:18=r27:26+r25:24, sr23:22=r27:26-r25:24;; //|F3+    |MPY2   |M1   |AS4-   |S2-   |
r3:2  =   Q[j1+=-44];   cb Q[k1+=32]=r19:18;  r25=mr1:0,  mr1:0+=r15**r31(C); sr29:28=r5:4+r7:6,     sr15:14=r5:4-r7:6;;    //|F4+    |       |M2   |AS1+   |S3-   |
r5:4  =br Q[j2+=32];    cb Q[k1+=16]=r23:22;                                    sr17:16=r9:8+r13:12,   sr21:20=r9:8-r13:12;;  //|F1++   |       |     |AS3    |S4-   |
                                                                                                                            // ------------------------------------
r7:6  =br Q[j3+=32];    cb Q[k1+=32]=r17:16;              mr1:0+=r14**r31(C); sr9:8=r1:0+r3:2,       sr27:26=r1:0-r3:2;;    //|F2++   |MPY1+  |     |AS2+   |S1    |
r1:0  =br Q[j0+=32];    cb Q[k1+=-16]=r21:20; r24=mr1:0,  mr1:0+=r15**r31(C); sr19:18=r11:10+r25:24, sr23:22=r11:10-r25:24;; //|F3++   |MPY2+  |M1+  |AS4    |S2    |
r3:2  =br Q[j1+=32];    cb Q[k1+=32]=r19:18;  r25=mr1:0,  mr1:0+=r15**r31(C); sr13:12=r5:4+r7:6,     sr15:14=r5:4-r7:6;;    //|F4++   |       |M2+  |AS1++  |S3    |
r5:4  =br Q[j2+=32];    cb Q[k1+=16]=r23:22;                                    sr17:16=r9:8+r29:28,   sr21:20=r9:8-r29:28;;  //|F1+++  |       |     |AS3+   |S4    |
                                                                                                                            // ------------------------------------
r7:6  =br Q[j3+=32];    cb Q[k1+=32]=r17:16;              mr1:0+=r14**r31(C); sr9:8=r1:0+r3:2,       sr11:10=r1:0-r3:2;;    //|F2+++  |MPY1++ |     |AS2++  |S1+   |
r1:0  =br Q[j0+=32];    cb Q[k1+=-16]=r21:20; r24=mr1:0,  mr1:0+=r15**r31(C); sr19:18=r27:26+r25:24, sr23:22=r27:26-r25:24;; //|F3+++  |MPY2++ |M1++ |AS4+   |S2+   |
r3:2  =br Q[j1+=32];    cb Q[k1+=32]=r19:18;  r25=mr1:0,  mr1:0+=r15**r31(C); sr29:28=r5:4+r7:6,     sr15:14=r5:4-r7:6;;    //|F4+++  |       |M2++ |AS1+++ |S3+   |
                                                                                                                            // ------------------------------------

.align_code 4;
if NLC0E, jump _VerFFTStage1;                                                                                               // ------------------------------------
r5:4  =br Q[j2+=32];    cb Q[k1+=16]=r23:22;                                    sr17:16=r9:8+r13:12,   sr21:20=r9:8-r13:12;;  //|F1+++  |       |     |AS3++  |S4+   |
                                                                                                                            // ------------------------------------

//************************************ Stage 2 ***************************************************************************
// 1st time: From j0->_ping_pong_buffer2 to k1->_ping_pong_buffer1
// 2nd time: From j0->_ping_pong_buffer1 to k1->_output

.align_code 4;
j0=j6+12*16;            j1=-4*16;;                                                                                          // ------------------------------------
r7:6  =   Q[j0+=-4*16]; r31:30=   L[k3+=-2];;                                                                               //|F1--   |       |     |       |      |
r5:4  =cb Q[j0+=-4*16]; r29:28=cb L[k3+=6];               mr1:0+=r7**r31(C);;                                               //|F2--   |MPY1-- |     |       |      |
r3:2  =   Q[j0+=-4*16]; LC0=7;                r15=mr1:0,  mr1:0+=r6**r31(C);;                                               //|F3--   |MPY2-- |M1-- |       |      |
r1:0  =cb Q[j0+=28*16]; j2=28*16;             r14=mr1:0,  mr1:0+=r5**r30(C);;                                               //|F4--   |MPY3-- |M2-- |       |      |
                                                                                                                            // ------------------------------------
r7:6  =cb Q[j0+=-4*16]; k1=j5;                r13=mr1:0,  mr1:0+=r4**r30(C);;                                               //|F1-    |MPY4-- |M3-- |       |      |
r5:4  =cb Q[j0+=j1];                          r12=mr1:0,  mr1:0+=r3**r29(C);;                                               //|F2-    |MPY5-- |M4-- |       |      |
                                              r11=mr1:0,  mr1:0+=r2**r29(C);;                                               //|       |MPY6-- |M5-- |       |      |
                                              r10=mr1:0,  mr1:0+=r1**r28(C);;                                               //|       |MPY7-- |M6-- |       |      |
                                                                                                                            // ------------------------------------
r3:2  =   Q[j0+=j1];                          r9=mr1:0,   mr1:0+=r0**r28(C);;                                               //|F3-    |MPY8-- |M7-- |       |      |
r1:0  =cb Q[j0+=28*16];                       r8=mr1:0,   mr1:0+=r7**r31(C); sr21:20=r13:12+r15:14,  sr23:22=r13:12-r15:14;; //|F4-    |MPY1-  |M8-- |AS1--  |      |
                                              r15=mr1:0,  mr1:0+=r6**r31(C);;                                               //|       |MPY2-  |M1-  |       |      |
                      kB1=k1+4;               r14=mr1:0,  mr1:0+=r5**r30(C); sr17:16=r9:8  +r11:10,  sr19:18=r9:8  -r11:10;; //|       |MPY3-  |M2-  |AS2--  |      |
r7:6  =cb Q[j0+=j1]; r8=r23;                  r13=mr1:0,  mr1:0+=r4**r30(C); sr9=-r23;;                                     //|F1     |MPY4-  |M3-  |A1--   |MV1-- |
r5:4  =cb Q[j0+=j1]; r31:30=cb L[k3+=-2];     r12=mr1:0,  mr1:0+=r3**r29(C); sr23=-r22;;                                     //|F2     |MPY5-  |M4-  |A2--   |      |
                                              r11=mr1:0,  mr1:0+=r2**r29(C); lr9:8=rot r9:8 by -16;;                        //|       |MPY6-  |M5-  |R1--   |      |
                      k5=k31+4*16;            r10=mr1:0,  mr1:0+=r1**r28(C); lr23:22=rot r23:22 by -16;;                    //|       |MPY7-  |M6-  |R2--   |      |
                                                                                                                            // ------------------------------------

.align_code 4;
_VerFFTStage2:                                                                                                              // ------------------------------------
r3:2  =   Q[j0+=j1]; r23=r8;                  r9=mr1:0,   mr1:0+=r0**r28(C); sr17:16=r17:16+r21:20,  sr21:20=r17:16-r21:20;; //|F3    |MPY8-  |M7-  |AS3--  |MV2-- |
r1:0  =cb Q[j0+=j2]; r29:28=cb L[k3+=6];      r8=mr1:0,   mr1:0+=r7**r31(C); sr25:24=r13:12+r15:14,  sr27:26=r13:12-r15:14;; //|F4    |MPY1   |M8-  |AS1-   |      |
cb Q[k1+=k5]=r17:16;                          r15=mr1:0,  mr1:0+=r6**r31(C); sr19:18=r19:18+r23:22,  sr23:22=r19:18-r23:22;; //|S1--  |MPY2   |M1   |AS4--  |      |
cb Q[k1+=k5]=r19:18;                          r14=mr1:0,  mr1:0+=r5**r30(C); sr17:16=r9:8  +r11:10,  sr19:18=r9:8  -r11:10;; //|S2--  |MPY3   |M2   |AS2-   |      |
r7:6  =cb Q[j0+=j1]; r8=r27;                  r13=mr1:0,  mr1:0+=r4**r30(C); sr9=-r27;;                                     //|F1+   |MPY4   |M3   |A1-    |MV1-  |
r5:4  =cb Q[j0+=j1];                          r12=mr1:0,  mr1:0+=r3**r29(C); sr27=-r26;;                                    //|F2+   |MPY5   |M4   |A2-    |      |
cb Q[k1+=k5]=r21:20;                          r11=mr1:0,  mr1:0+=r2**r29(C); lr9:8=rot r9:8 by -16;;                        //|S3--  |MPY6   |M5   |R1-    |      |
cb Q[k1+=k5]=r23:22;                          r10=mr1:0,  mr1:0+=r1**r28(C); lr27:26=rot r27:26 by -16;;                    //|S4--  |MPY7   |M6   |R2-    |      |
                                                                                                                            // ------------------------------------
r3:2  =   Q[j0+=j1]; r27=r8;                  r9=mr1:0,   mr1:0+=r0**r28(C); sr17:16=r17:16+r25:24,  sr25:24=r17:16-r25:24;; //|F3+   |MPY8   |M7   |AS3-   |MV2-  |
r1:0  =cb Q[j0+=j2];                          r8=mr1:0,   mr1:0+=r7**r31(C); sr21:20=r13:12+r15:14,  sr23:22=r13:12-r15:14;; //|F4+   |MPY1+  |M8   |AS1    |      |
cb Q[k1+=k5]=r17:16;                          r15=mr1:0,  mr1:0+=r6**r31(C); sr19:18=r19:18+r27:26,  sr27:26=r19:18-r27:26;; //|S1-   |MPY2+  |M1+  |AS4-   |      |
cb Q[k1+=k5]=r19:18;                          r14=mr1:0,  mr1:0+=r5**r30(C); sr17:16=r9:8  +r11:10,  sr19:18=r9:8  -r11:10;; //|S2-   |MPY3+  |M2+  |AS2    |      |
r7:6  =cb Q[j0+=j1]; r8=r23;                  r13=mr1:0,  mr1:0+=r4**r30(C); sr9=-r23;;                                     //|F1++  |MPY4+  |M3+  |A1     |MV1   |
r5:4  =cb Q[j0+=j1]; r31:30=cb L[k3+=-2];     r12=mr1:0,  mr1:0+=r3**r29(C); sr23=-r22;;                                     //|F2++  |MPY5+  |M4+  |A2     |      |
cb Q[k1+=k5]=r25:24;                          r11=mr1:0,  mr1:0+=r2**r29(C); lr9:8=rot r9:8 by -16;;                        //|S3-   |MPY6+  |M5+  |R1     |      |
.align_code 4;                                                                                                              // ------------------------------------
if NLC0E, jump _VerFFTStage2;                                                                                               // ------------------------------------
cb Q[k1+=k5]=r27:26;                          r10=mr1:0,  mr1:0+=r1**r28(C); lr23:22=rot r23:22 by -16;;                    //|S4-   |MPY7+  |M6+  |R2     |      |
                                                                                                                            // ------------------------------------
.align_code 4;                                                                                                              // ------------------------------------
                      r23=r8;                 r9=mr1:0,   mr1:0+=r0**r28(C); sr17:16=r17:16+r21:20,  sr21:20=r17:16-r21:20;; //|       |MPY8+  |M7+  |AS3    |MV2   |
k3=k31+_AdjustMatrix;                         r8=mr1:0,   mr1:0+=r7**r31(C); sr25:24=r13:12+r15:14,  sr27:26=r13:12-r15:14;; //|       |       |M8+  |AS1+   |      |
                                                                                                                            // ------------------------------------
cb Q[k1+=k5]=r17:16; k2=-236;                                                  sr19:18=r19:18+r23:22,  sr23:22=r19:18-r23:22;; //|S1     |       |     |AS4    |      |
cb Q[k1+=k5]=r19:18; j10=j31+j6;                                              sr17:16=r9:8  +r11:10,  sr19:18=r9:8  -r11:10;; //|S2     |       |     |AS2+   |      |
                                                                                                                            // ------------------------------------
cb Q[k1+=k5]=r21:20; j9=j31+j5;;

//************************************ MPY/Xpose *************************************************************************
// 1st time: From _ping_pong_buffer1 to _ping_pong_buffer2
                                                                                                                            // ------------------------------------
r29:28=Q[k3+=16];    r8=r27;                                                  sr9=-r27;;                                    //|       |       |     |       |TF1-- |
r1:0=   Q[j9+=16];   cb Q[k1+=k5]=r23:22;                                     sr27=-r26;;                                   //|F1--   |       |     |       |      |
r3:2=   Q[j9+=16];   r31:30=Q[k3+=16];                                        lr9:8=rot r9:8 by -16;;                       //|F2--   |       |     |       |TF2-- |
r5:4=   Q[j9+=16];   r21:20=Q[k3+=16];                                        lr27:26=rot r27:26 by -16;;                   //|F3--   |       |     |       |TF3-- |
j4=j31+j6;           r27=r8;                                  mr1:0+=r2**r28(C); sr17:16=r17:16+r25:24,  sr25:24=r17:16-r25:24;; //|       |MPY1-- |     |       |      |
                                                                                                                            // ------------------------------------
r7:6=   Q[j9+=16];   r23:22=Q[k3+=16];    r8=mr1:0,   mr1:0+=r3**r29(C);;                                                   //|F4--   |MPY2-- |M1-- |       |TF4-- |
cb Q[k1+=k5]=r17:16; LC0=4;               r12=mr1:0,  mr1:0+=r0**r30(C); sr19:18=r19:18+r27:26,  sr27:26=r19:18-r27:26;; //|       |MPY3-- |M2-- |       |      |
cb Q[k1+=k5]=r19:18; j6=j5;                r9=mr1:0,   mr1:0+=r1**r31(C);;                                                 //|       |MPY4-- |M3-- |       |      |
cb Q[k1+=k5]=r25:24; k8=j4;                r13=mr1:0,  mr1:0+=r4**r20(C);;                                                 //|       |MPY5-- |M4-- |       |      |
                                                                                                                            // ------------------------------------

.align_code 4;
if LC1E, CJMP(ABS);                                                                                                         // ------------------------------------
cb Q[k1+=k5]=r27:26; j5=j7;               r10=mr1:0,  mr1:0+=r5**r21(C);;                                                   //|       |MPY6-- |M5-- |       |      |
                                                                                                                            // ------------------------------------

.align_code 4;
_MultXposeLoop:                                                                                                             // ------------------------------------
r1:0=   Q[j9+=16];   r17:16=Q[k3+=16];    r14=mr1:0,  mr1:0+=r6**r22(C);;                                                   //|F1-   |MPY7-- |M6-- |       |TF1-  |
r3:2=   Q[j9+=16];   r19:18=Q[k3+=16];    r11=mr1:0,  mr1:0+=r7**r23(C);;                                                   //|F2-   |MPY8-- |M7-- |       |TF2-  |
r5:4=   Q[j9+=16];   r21:20=Q[k3+=16];    r15=mr1:0,  mr1:0+=r0**r16(C);;                                                   //|F3-   |MPY1-  |M8-- |       |TF3-  |
r7:6=   Q[j9+=16];   r23:22=Q[k3+=16];    r24=mr1:0,  mr1:0+=r1**r17(C);;                                                   //|F4-   |MPY2-  |M1-  |       |TF4-  |
Q[j10+=16]=yr11:8;   j0=k8;               r28=mr1:0,  mr1:0+=r2**r18(C);;                                                   //|S1--  |MPY3-  |M2-  |       |      |
Q[j10+=16]=yr15:12;  k9=k8+128;           r25=mr1:0,  mr1:0+=r3**r19(C);;                                                   //|S2--  |MPY4-  |M3-  |       |      |
Q[j10+=16]=xr11:8;   j1=k9;               r29=mr1:0,  mr1:0+=r4**r20(C);;                                                   //|S3--  |MPY5-  |M4-  |       |      |
Q[j10+=-44]=xr15:12; k1=j6;               r26=mr1:0,  mr1:0+=r5**r21(C);;                                                   //|S4--  |MPY6-  |M5-  |       |      |
                                                                                                                            // ------------------------------------
r1:0=   Q[j9+=16];   r17:16=Q[k3+=16];    r30=mr1:0,  mr1:0+=r6**r22(C);;                                                   //|F1    |MPY7-  |M6-  |       |TF1   |
```

```
r3:2=   Q[j9+=16];   r19:18=Q[k3+=16];       r27=mr1:0,     mr1:0+=r7**r23(C);;                    //|F2   |MPY8-  |M7-  |       |TF2  |
r5:4=   Q[j9+=16];   r21:20=Q[k3+=16];       r31=mr1:0,     mr1:0+=r0**r16(C);;                    //|F3   |MPY1   |M8-  |       |TF3  |
r7:6=   Q[j9+=16];   r23:22=Q[k3+=16];       r8=mr1:0,      mr1:0+=r1**r17(C);;                    //|F4   |MPY2   |M1   |       |TF4  |
Q[j10+=16]=yr27:24;  k9=k8+64;               r12=mr1:0,     mr1:0+=r2**r18(C);;                    //|S1-  |MPY3   |M2   |       |     |
Q[j10+=16]=yr31:28;  j2=k9;                  r9=mr1:0,      mr1:0+=r3**r19(C);;                    //|S2-  |MPY4   |M3   |       |     |
Q[j10+=16]=xr27:24;  k9=k8+(128+64);         r13=mr1:0,     mr1:0+=r4**r20(C);;                    //|S3-  |MPY5   |M4   |       |     |
Q[j10+=-44]=xr31:28; j3=k9;                  r10=mr1:0,     mr1:0+=r5**r21(C);;                    //|S4-  |MPY6   |M5   |       |     |
                                                                                                   // ------------------------------------
r1:0=   Q[j9+=16];   r17:16=Q[k3+=16];       r14=mr1:0,     mr1:0+=r6**r22(C);;                    //|F1+  |MPY7   |M6   |       |TF1+ |
r3:2=   Q[j9+=16];   r19:18=Q[k3+=16];       r11=mr1:0,     mr1:0+=r7**r23(C);;                    //|F2+  |MPY8   |M7   |       |TF2+ |
r5:4=   Q[j9+=16];   r21:20=Q[k3+=16];       r15=mr1:0,     mr1:0+=r0**r16(C);;                    //|F3+  |MPY1+  |M8   |       |TF3+ |
r7:6=   Q[j9+=-236]; r23:22=Q[k3+=k2];       r24=mr1:0,     mr1:0+=r1**r17(C);;                    //|F4+  |MPY2+  |M1+  |       |TF4+ |
Q[j10+=16]=yr11:8;                           r28=mr1:0,     mr1:0+=r2**r18(C);;                    //|S1   |MPY3+  |M2+  |       |     |
Q[j10+=16]=yr15:12;                          r25=mr1:0,     mr1:0+=r3**r19(C);;                    //|S2   |MPY4+  |M3+  |       |     |
Q[j10+=16]=xr11:8;                           r29=mr1:0,     mr1:0+=r4**r20(C);;                    //|S3   |MPY5+  |M4+  |       |     |
Q[j10+=-44]=xr15:12;                         r26=mr1:0,     mr1:0+=r5**r21(C);;                    //|S4   |MPY6+  |M5+  |       |     |
                                                                                                   // ------------------------------------
r1:0=   Q[j9+=16];   r17:16=Q[k3+=16];       r30=mr1:0,     mr1:0+=r6**r22(C);;                    //|F1++ |MPY7+  |M6+  |       |TF1++|
r3:2=   Q[j9+=16];   r19:18=Q[k3+=16];       r27=mr1:0,     mr1:0+=r7**r23(C);;                    //|F2++ |MPY8+  |M7+  |       |TF2++|
r5:4=   Q[j9+=16];   r21:20=Q[k3+=16];       r31=mr1:0,     mr1:0+=r0**r16(C);;                    //|F3++ |MPY1++ |M8+  |       |TF3++|
r7:6=   Q[j9+=16];   r23:22=Q[k3+=16];       r8=mr1:0,      mr1:0+=r1**r17(C);;                    //|F4++ |MPY2++ |M1++ |       |TF4++|
Q[j10+=16]=yr27:24;                          r12=mr1:0,     mr1:0+=r2**r18(C);;                    //|S1+  |MPY3++ |M2++ |       |     |
Q[j10+=16]=yr31:28;                          r9=mr1:0,      mr1:0+=r3**r19(C);;                    //|S2+  |MPY4++ |M3++ |       |     |
Q[j10+=16]=xr27:24;                          r13=mr1:0,     mr1:0+=r4**r20(C);;                    //|S3+  |MPY5++ |M4++ |       |     |
.align_code 4;                                                                                     // ------------------------------------
if NLC0E, jump _MultXposeLoop;                                                                     // ------------------------------------
Q[j10+=4]=xr31:28;                           r10=mr1:0,     mr1:0+=r5**r21(C);;                    //|S4+  |MPY6++ |M5++ |       |     |
                                                                                                   // ------------------------------------
.align_code 4;
jump _VerticalLoop;                                                                                // Repeat the vertical loop
r5:4=br Q[j2+=32];   k3=k31+(_twiddles16+2);;                                                      // with swapped pointers

//****************************************** Done ***************************************************************************
_fft256pt.end:
```

表1　fft256pt.asm

参考文献

[1] ADSP-TS201 TigerSHARC Processor Programming Reference. Revision 1.0, August 2004. Analog Devices, Inc.

文件历史

版本	描述
第一版 2005年2月3日 *Boris Lerner提供*	第一版