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

为ADSP-TS201 TigerSHARC®处理器编写高效的浮点FFT

Boris Lerner提供

修订版2-2004年4月

引言

是否想为ADSP-TS201 TigerSHARC®处理器编写高效的代码？或者，也许有关该处理器的优化浮点FFT举例给你留下了深刻印象，你想了解它的工作原理以及程序员在书写代码时的所思所想。本应用手记试图通过全面细致地分析该FFT代码例子及其所有优化级来回答你的这两个问题。本例可以在开发针对ADSPTS201S处理器的其它的优化算法和代码时遵循。

一般来讲，大多数算法都有几个优化级，这些将在本手记中详加讨论。第一个和最直接的优化层次是正如处理器所允许的，是指令的并行。它简单而枯燥。第二个优化层次是循环体展开（loop unrolling）和软件流水线操作，以获得最大的并行性和避免流水线阻塞（stall）。尽管比级别1的简单并行性要复杂，但它不要求对算法的很好理解便可以依规定的步骤完成，因而它要求很少的智力劳动。第三个优化层次是对算法的数学表达进行重建，仍能产生有效的结果，而重建后的新算法更好地适于处理器架构。做到这一步需要对算法的全面理解，且不像软件流水线操作那样，它没有规定步骤引导出最佳解决方案。这也正是编写优化代码的最精妙之处。

实际应用中，不是经常要经历全部三个层次的。但在需要经过全部三个层次时，以反向的次序来做这些优化层次总是最好的。在代码完全进入流水线操作后，再来改变基础的底层算法就太迟了。因此，作为一名编程人员，需要首先考虑算法结构并据此对代码进行组织。而后，级别2和级别1（并行、展开以及流水线操作）优化层次通常同时进行。

在本手记中出现的代码由模拟器件公司以某种形式提供，它允许作为一种实数或复数FFT被调用，函数的最后调用参数定义是调用实数还是复数。实数N-点FFT由复数N/2-点FFT取得，它在终结处有一个附加的特别进程（stage）。本手记更多关注代码的优化而不是该特别进程的技术性，因此，只讨论代码的复数FFT部分的算法。实数FFT最后的特别进程在代码注释中详加讨论。

标准Radix-2 FFT算法

图1示出一种输入经过比特位反转（bitreverse）后的标准16点radix-2 FFT实现。传统来看，在这一算法中，第一级和第二级蝶形运算是按照所要求的位反转方式结合在一起成为一种单一的优化循环的（因为这两级蝶形运算不要求乘法运算，只是加和减）。其余的每一级蝶形运算都是将共享相同旋转因子（twiddle factor）的蝶形运算（butterflies）结合成组（这样对于每一组，旋转因子只能取数一次）来进行。用TigerSHARC处理器实现这一算法的未优化汇编源代码示于列表1中。它具有几个与本讨论无关的决窍之处（trick），这便是面向ADSP-TS101处理器目标时32位浮点FFT代码的书写方式。这一算法包括位反转在ADSP-TS101和ADSP-TS201上运行的基准（在内核时钟周期下）示于表1中。注意，由于ADSP-TS101在单位存储

块内存容量上比ADSPTS201要小，因此，较大点尺寸（point size）的基准不施加于ADSP-TS101。清楚可见，只要数据与ADSP-TS201高速缓存适配，那么它便是高效的。一旦数据对于高速缓存变得过大，这个FFT实现方案就会变得极为低效---循环计数由最佳增加到5倍。

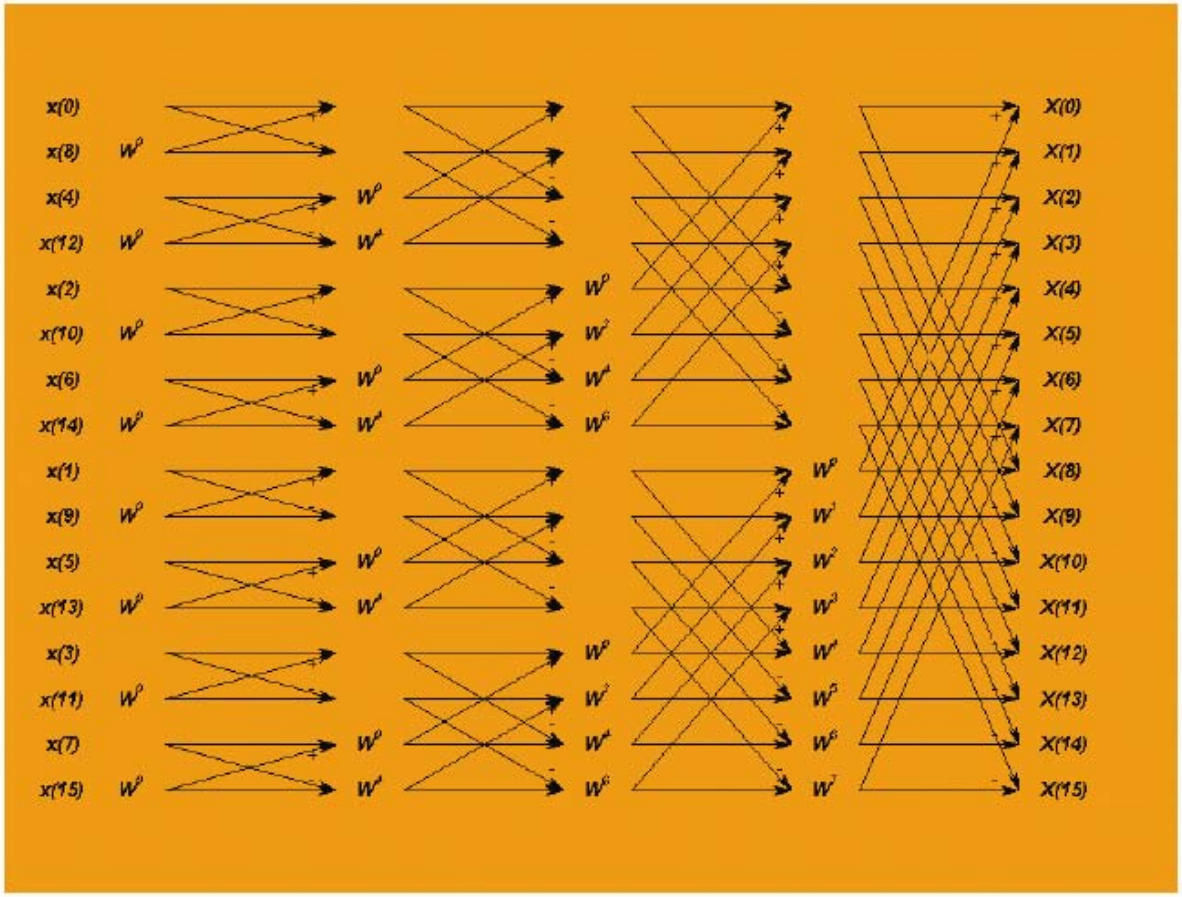

图1 16点FFT的标准结构

```
//********************************** Stages ************************************

      k10 = k31 + N;;                                   // twiddles stride
      k11 = k31 + N/2;;                                 // butterflies/group
      j12 = j31 + 1;;                                   // groups
      j10 = j31 + 2;;                                   // width of butterfly
      j11 = j31 + 4;;                                   // butterfly stride
      k20 = k31 + STAGES;;

_stages_loop:
            j0 = j31 + j29;;                            // j0 -> internal_buff
            k0 = k31 + k30;;                            // k0 -> twiddles
            j13 = j31 + 0;;

            LC1 = j12;;
_group_loop:
                  xr1:0 = L[k0 += k10];;                // xr0=cos, xr1=-sin
                  j1 = j0 + j10;;                       // j1 -> second input to butterfly

                  LC0 = k11;;

_butterfly_loop:
                        xr3:2 = L[j0 += 0];;            // xr2=Re1, xr3=Im1
                        xr5:4 = L[j1 += 0];;            // xr4=Re2, xr5=Im2
                        xfr6 = r4 * r0;;                // xr6=Re2*cos
                        xfr7 = r5 * r1;;                // xr7=Im2*sin
                        xfr8 = r6 - r7;;                // xr8=Re(z2*twid)
                        xfr9 = r4 * r1;;                // xr6=Re2*sin
                        xfr10 = r5 * r0;;               // xr7=Im2*cos
                        xfr11 = r9 + r10;;              // xr8=Im(z2*twid)
                        xfr12 = r2 + r8, fr14 = r2 - r8;;   // Re(butterfly)
                        xfr13 = r3 + r11, fr15 = r3 - r11;; // Im(butterfly)
                        L[j0 += j11] = xr13:12;;
                        L[j1 += j11] = xr15:14;;
                        if NLC0E, jump _butterfly_loop (NP);;

                  j13 = j13 + 2;;
                  j0 = j29 + j13;;                      // offset for the next group
                  if NLC1E, jump _group_loop (NP);;

            k10 = lshiftr k10;;                         // twiddles stride
            k11 = lshiftr k11;;                         // butterflies/group
            j12 = j12 + j12;;                           // groups
            j10 = j10 + j10;;                           // width of butterfly
            j11 = j11 + j11;;                           // butterfly stride

            k20 = k20 - 1;;
            if NKEQ, jump _stages_loop (NP);;
```

列表1　　fft32_unoptimized.asm

点N	*ADSPTS101*	*ADSP-TS201 输入没位于高速缓*	*ADSP-TS201 输入位于高速缓存中*
256	*2172*	*2641*	*2218*
512	*4582*	*5533*	*4649*
1024	*9872*	*12170*	*9992*
2048	*21338*	*26610*	*22173*
4096	*46244*	*197272*	*NA*
8192	*99886*	*444628*	*NA*
16384	*215224*	*987730*	*NA*
32768	*NA*	*2133220*	*NA*
65536	*NA*	*4720010*	*NA*

表1 N点复数FFT的内核时钟周期

优化ADSP-TS201处理器的FFT结构

为做到算法重建，实现其在ADSP-TS201的最佳操作，我们需要了解为什么采用常规FFT结构的大FFT，其性能会如此之差。

ADSP-TS201内存是针对顺序读取优化的。高速缓冲的设计用来为非顺序读取的算法提供帮助。在常规FFT算法中，每一级的蝶形运算步进为双倍，这样，读取就是非顺序的，而对于每一个新级的蝶形运算，高速缓存越来越不可能做到命中---读取随处进行（all over the place）。此解决方案的目的是要对一个进程的输出进行重排以确保下一个进程的读取是顺序的。算法实现结构示于图2中。

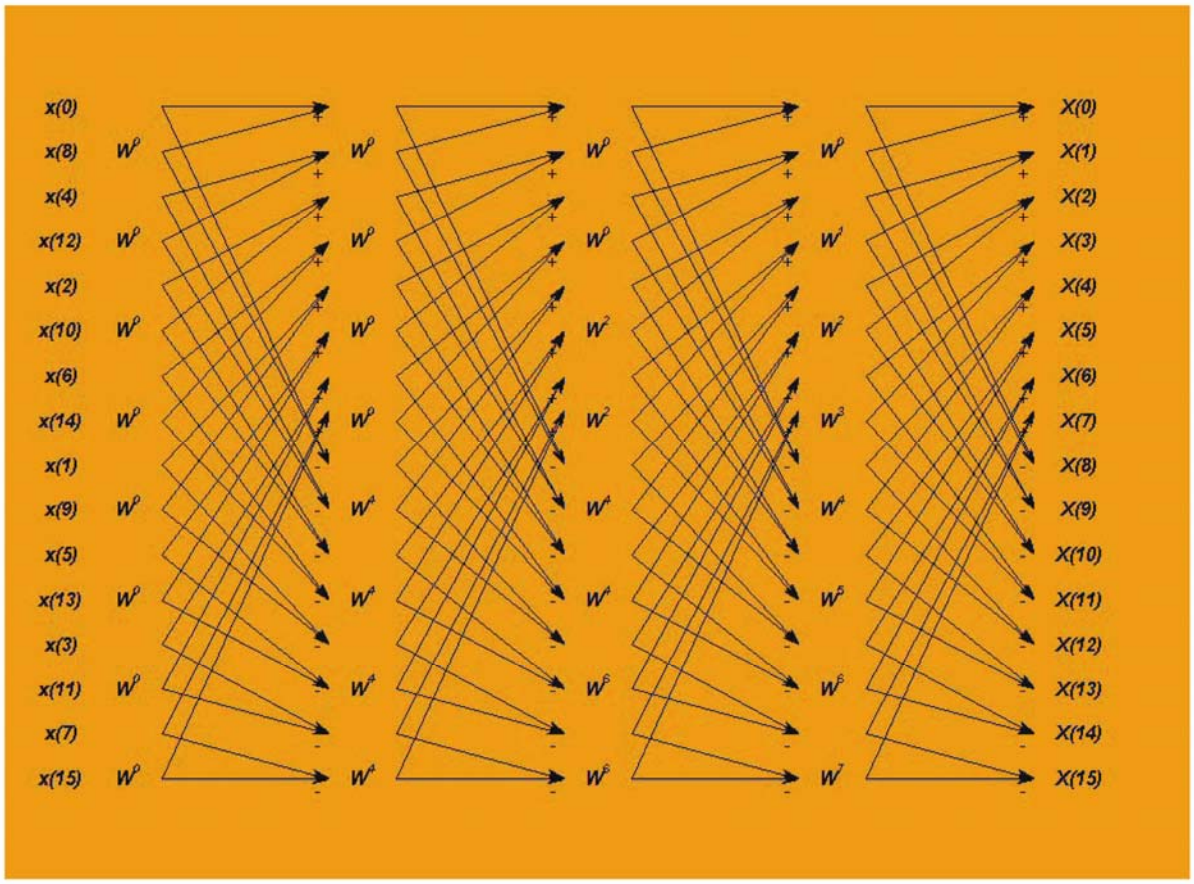

图2　重排的16点FFT结构

简单地用手工跟踪这张图表便可看出，它不过是图1中图解的重新排序。认人惊奇的是，最终的输出是以正确的次序进行的。这可以通过通常的N=*2k* =FFT中的点数量容易地得到证明。注意，重新排序由以下公式给出：

$$f(n)=\begin{cases}\dfrac{n}{2}, \text{if } n \text{ is even}\\ \dfrac{n-1}{2}+\dfrac{N}{2}, \text{if } n \text{ is odd}\end{cases}$$

这样，如果n为偶数，那么它就右移，如果N为奇数，那么它右移且最高位的bit被置为1 。这当然等效于1-bit右旋转操作，它在K=log2(*N*)步骤后返回原始n值。这样，K个进程后的输出再次回到正确的顺序。

真棒！ 我们拥有了我们的新结构。它为顺序读取，我们幸运地得到输出为正确的顺序形式。这应该会有效得多。对吧？让我们来写一写它的代码。在我们花去大量时间书写代码之前，我们应该确保我们实际将要进行的所有DSP操作都高效地适于我们的处理器架构。如果底层数据运算受制，优化数据移动应是责无旁贷。

第一个需要注意的明显之处是，*因其重新排序，这一结构不能做到原封不动*。进程将需要对其输入/输出缓冲器进行ping-pong操作。这应该不成问题。ADSP-TS201处理器拥有大量的板上内存，不过需要进行内存优化（输入不一定要保留），我们可以把输入当作两个ping-pong缓冲器之一。

接下来我们要注意，传统的FFT是将共享旋转的蝶形运算结合到同一个组以节省旋转取数周

期。惊奇的是，图2中结构的旋转呈线性排列---某一时刻有一个组。我们又一次很幸运!

现在来看一看这种新结构的一个蝶形运算都有哪些组成。表2列出了进行单一的复数蝶形运算所需要的操作。由于ADSP-TS201是一种SIMD处理器（即，它可使所有计算加倍）我们将以SIMD格式书写列表1中概括的步骤，这样相邻的两个蝶形运算便为并行计算，一个位于X-Compute单元块，另一个位于Y-Compute单元块。我们来进一步详细分析DSP操作。F1、F2、K2及F4找取总计4个32-bit字，这在ADSP-TS201上通过一个在X-Compute单元块寄存器中的单一四元取数（single quad fetch）完成。为了能够给SIMD机提供数据，我们还需要在Y-Compute单元块寄存器内进行第二次蝶形四元取数。而后M1、M2、M3、M4、A1和A2将针对两蝶形运算实施SIMD操作。

ADSP-TS201支持单一的加/减指令，因此A3和A4可以结合成一个单操作（这当然是一次在两个蝶形运算上执行SIMD），同样A5和A6也可以结合。

记忆存储器	*操作*
F1	*Fetch Real(Input1) of the Butterfly*
F2	*Fetch Imag(Input1) of the Butterfly*
K2	*Fetch Real(Input2) of the Butterfly*
F4	*Fetch Imag(Input2) of the Butterfly*
M1	*K2 * Real(twiddle)*
M2	*F4 * Imag(twiddle)*
M3	*K2 * Imag(twiddle)*
M4	*F4 * Real(twiddle)*
A1	*M1　M2 = Real(Input2*twiddle)*
A2	*M3+M4 = Imag(Input2*twiddle)*
A3	*F1 + A1 = Real(Output1)*
A4	*F1 - A1 = Real(Output2)*
A5	*F2 + A2 = Imag(Output1)*
A6	*F2　A2 = Imag(Output2)*
S1	*Store(Real(Output1))*
S2	*Store(Imag(Output1))*
S3	*Store(Real(Output2))*
S4	*Store(Imag(Output2))*

表2　呈线性的单蝶形运算---逻辑实现

记忆存储器	*操作*
F1	*Fetch Input1,2 of the Butterfly1*
F2	*Fetch Input1,2 of the Butterfly2*
M1	*Real(Input2) * Real(twiddle)*
M2	*Imag(Input2) * Imag(twiddle)*
M3	*Real(Input2) * Imag(twiddle)*
M4	*Imag(Input2) * Real(twiddle)*
A1	*M1　M2 = Real(Input2*twiddle)*
A2	*M3+M4 = Imag(Input2*twiddle)*
A3	*Real(Input1)+/-A1=al(Output1,2)*
A4	*Imag(Input1)+/-A2=ag(Output1)*
S1	*Store(Output1, both Butterflies)*
S2	*Store(Output2, both Butterflies)*

表3 呈线性的单蝶形运算---实际的ADSP-TS20x实现

现在我们进入问题：S1、S2、S3和S4不能在同一周期内进行操作，因为S3和S4因输出的重新排列被指向存储器中的另一个位置。但，我们可以在一个周期内将S1和S2存储于两个蝶形运算（再次幸运---这些是相邻的），在下个周期将S3和S4存储于两个蝶形运算。一切顺利---新

的操作集在表3中进行了归总。

表3中的每一项操作都是ADSP-TS201上的单一周期操作。总共有2个取数、四个乘法、4个ALU和2个存储指令操作。由于ADSP-TS201对于乘法和ALU操作允许在单一个周期内并行开展找取/存储，因此，循环展开、流水线操作和并行带来这两个SIMD蝶形运算的一次4周期执行（我们在存储器的使用上更是高效的！）。此刻我们可以有理由相信上述所列将产生高效代码，我们可以开始对它进行开发。不过，这时我们的仔细观察会有助于我们对这种结构的进一步优化。注意，我们只能在单一个存储块中比如JALU指针寄存器的情形，使用总共4个取数和存储操作。以并行方式我们可以进行3个以上的取数/存储/KALU操作而不丢失任何周期（实际上我们可以做到4个操作，但我们得要保留一个位置给其中的指令之一进行环跳返回（loop jump back））。

这样，每个共享蝶形组仅有一次旋转取数的旧有法则就不再需要---旋转取数变得自由！再有，由于图2中箭头的结构在每一个进程都是相同的，因此，假如我们可以找到一种正确找取每一进程旋转的途径（旋转成为分辨图2各进程的惟一借物），我们便能够把FFT由通常的3个嵌套循环减少到2个。图2示出旋转在每一进程如何必须找取。第一个Stage---全部为*W0*。第二个Stage　半为*W0*,，另一半为*WN/4*。第三个Stage　四分之一为*W0*，另四分之一为*WN/8*，再四分之一为*W2N/8*，最后四分之一为*W3N/8*　依此类推….。如果我们保持一个有效旋转指针偏移量，以每个蝶形运算递增1到下一个顺序旋转，而在实际将它用于旋转取数前用一个屏蔽对它进行AND操作，那么我们就准确地得到该旋转取数次序。另外，这一法则对每个进程都是相同的，除非屏蔽在每个进程都必须降移一个bit（即每个进程要求两倍于原有进程旋转分辨率的精细度）。这时我们未曾用过的KALU操作唾手可得。为实现这种旋转取数方法，我们需要递增有效偏移量，屏蔽它以及每个蝶形运算做一次旋转取数….。噢，不！我们是SIMD方式（即我们在把两个蝶形运算一起进行）且我们不具有6个可用指令时段用于此！但幸运再次让我们解脱。我们能容易地注意到，全部进程除最后一个进程外都是在SIMD蝶形运算对之间共享旋转，这样，对于这些进程我们只需要做到每个SIMD蝶形运算对进行旋转取数一次即可。三个周期恰恰是我们进行这种操作所具备的。不巧的是在最后的进程中，每一个蝶形运算都其自身独特的旋转；不过最后的进程我们不必使用屏蔽---每次只将指针进阶到下一个旋转即可。它将需要分开书写，但它也将得到完全优化。表4概括了最新结构的步阶。在表3的基础上增加了3个新型KALU操作(K1, K2 and K3)。到书写代码的时候了吗？哦，还没有---我们先来解决如何对它进行流水线操作。

记忆存储器	***操作***
K1	***Virtual Pointer Offset Mask***
K2	***Twiddles Fetch***
K3	***Virtual Pointer Offset Increment***
F1	***Fetch Input1,2 of the Butterfly1***
F2	***Fetch Input1,2 of the Butterfly2***
M1	***Real(Input2) * Real(twiddle)***
M2	***Imag(Input2) * Imag(twiddle)***
M3	***Real(Input2) * Imag(twiddle)***
M4	***Imag(Input2) * Real(twiddle)***
A1	***M1　M2 = Real(Input2*twiddle)***
A2	***M3+M4 = Imag(Input2*twiddle)***
A3	***Real(Input1)+/-1=Real(Output1,2)***
A4	***Imag(Input1)+/-A2=Imag(Output1)***
S1	***Store(Output1, both Butterflies)***
S2	***Store(Output2, both Butterflies)***

表4　呈线性的单蝶形运算---改进后的ADSP-TS20x的实现方案

算法的流水线操作

图3示出了由表4演化出来的算法操作，箭头示出相关性。相关性的箭头表示箭头起始处的操作结果被箭头结束处的操作所使用，从而必须首先完成以确保正确的数据操作。有些箭头自身带有一种阻塞，特别是：

K2 -> M1, M2, M3, M4

F1, F2 -> M1, M2, M3, M4, A3, A4

M1, M2 -> A1

M3, M4 -> A2

A1, A2 -> A3, A4

这意味着，如果箭头起始处的操作后面紧跟箭头结束处的操作，那么结果将是正确的，但代码执行将产生一个阻塞。因此，完全优化此代码，箭头结束处带有阻塞的操作必须保持1条以上的指令行分离。

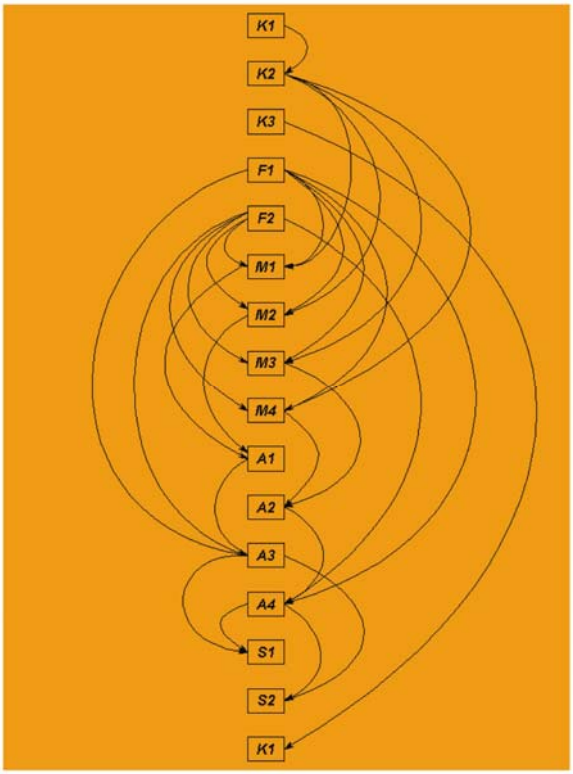

图3　重新组织的结构的相关性

对图3相关性的快速观察足以分析流水线操作的水平以及操作所需的计算单元块寄存器数量。

状态	*对状态的相关性*	*最大相关性周期*	*所需计算单元块寄存器*
K1	*K2*	*1*	*0*
K2	*M1,M2,M3,M4*	*5*	*4*([5/4]+1)=8*
K3	*K1*	*1*	*0*
F1	*M1,M2,M3,M4, A1,A2*	*10*	*4*([10/4]+1)=16*
F2	*M1,M2,M3,M4, A1,A2*	*10*	*4*([10/4]+1)=16*
M1	*A1*	*2*	*2([2/4]+1)=2*
M2	*A1*	*2*	*2([2/4]+1)=2*
M3	*A2*	*2*	*2([2/4]+1)=2*
M4	*A2*	*2*	*2([2/4]+1)=2*
A1	*A3,A4*	*2*	*2([2/4]+1)=2*
A2	*A3,A4*	*2*	*2([2/4]+1)=2*
A3	*S1,S2*	*1*	*4([1/4]+1)=4*
A4	*S1,S2*	*1*	*4([1/4]+1)=4*
S1	*none*	*0*	*0*
S2	*none*	*0*	*0*
Total	*Regs*	*6*	*0*
	总寄存器数		*60*

图5　蝶形运算流水线操作所需的计算块寄存器数量

如前所述，完全流水线操作会带来一个4周期的SIMD蝶形运算对。从而有以下

*Pipelined_CB_Registers_Per_State_Output = Unpipelined_CB_Registers_Per_State_Output * ([Maximum_Dependency_Cycles/4]+1)*

这里的*[x]*表示数*x*的整数部分。由此我们可以确定所需的计算单元块寄存器的数量，如表5所

示。注意，由A*A3* 和*A4*为加/减运算，因此，它们所要求的输出寄存器数量是*M1*，*M2*，*M3*，*M4*，*A1* 及*A2*的两倍。

实现这一代码完全流水线操作最终需要总计64个计算块寄存器中的60个，刚好满足要求。

周期操作	*JALU*	*KALU*	*MAC*	*ALU*
1	*F1*	*K1*	*M4--*	*A3---*
2	*F2*	*K2*	*M2--*	*A4----*
3	*S1----*		*M3-*	*A2--*
4	*S2---*	*K3*	*M1*	*A1-*
5	*F1+*	*K1+*	*M4-*	*A3--*
6	*F2+*	*K2+*	*M2*	*A4--*
7	*S1--*		*M3*	*A2-*
8	*S2--*	*K3+*	*M1+*	*A1*
9	*F1++*	*K1++*	*M4*	*A3-*
10	*F2++*	*K2++*	*M2+*	*A4-*
11	*S1-*		*M3+*	*A2*
12	*S2-*	*K3++*	*M1++*	*A1+*
13	*F1+++*	*K1+++*	*M4+*	*A3*
14	*F2+++*	*K2+++*	*M2++*	*A4*
15	*S1*		*M3++*	*A2+*
16	*S2*	*K3+++*	*M1+++*	*A1++*

表6 流水线式蝶形运算

我们使用表4和图3的表记方法（mnemonics）对此流水线操作完全表征化。流水线操作示于表6中，其中操作中的“+”号表示该操作与下一组的蝶形运算对应，“-”号表示与上一组的蝶形运算操作对应。

所有指令均是并行的，没有任何阻塞，且存在一个位置一步跳转到循环体的顶部（实际是4个位置，但这只是因为流水线操作深度为4对蝶形运算，表6中循环体的每一次迭代将实际做出4对蝶形运行计算）。

代码

现在，书写代码轻而易举。ADSPTS201非常灵巧，它把所有难处完全排除在外。只需遵循表6中的流水线操作，代码便即刻完成。各进程的最终代码（最后进程除外）见列表2所示。

这一内层循环体的外部是一个进程循环，它对输入/输出缓冲器进行ping-pong操作并将改变旋转修改量屏蔽。极为简单！

将最前面的两个进程从代码主体中分离出来，使它们独立操作，做到代码的额外优化----它们并不要求复杂的乘法运算，可做到运行更快。另外，该两个进程还纳入了位反转操作。现在，最低来看，循环计数有多大的改进呢？在表7中我们重复表1的分栏内容，对新算法额外增添了基准列。对于大于缓存（larger-than cache）的FFT，循环计数改进3倍以上。对于适配缓存的FFT，循环计数性能好于早期ADSP-TS101处理器的表现，此处理器没有任何的缓存或内存等待时间问题。其原因在于新架构允许代码在两个而非三个嵌套循环体中书写，从而大大减少了内消。这一代码还可以移植到ADSP-TS101，改进其基准，见表7所示。

```
.align_code 4;
_BflyLoop:
            q[j2+=4]=r27:26;  k5=k5+k9;        fr6=r30*r12;  fr16=r6-r7;;                    // S2----,K3-,   M1-,   A1--

            yr3:0=q[j0+=4];   k3=k5 and k4;    fr15=r23*r4;  fr24=r8+r18,  fr26=r8-r18;;     // F1,    K1,    M4--,  A3---
            xr3:0=q[j0+=4];   r5:4=l[k7+k3];   fr7=r31*r13;  fr25=r9+r19,  fr27=r9-r19;;     // F2,    K2,    M2-,   A4---
            q[j1+=4]=r25:24;                   fr14=r30*r13; fr17=r14+r15;;                  // S1---,         M3-,   A2--
            q[j2+=4]=r27:26;  k5=k5+k9;        fr6=r2*r4;    fr18=r6-r7;;                    // S2---, K3,     M1,    A1-

            yr11:8=q[j0+=4];  k3=k5 and k4;    fr15=r31*r12; fr24=r20+r16, fr26=r20-r16;;    // F1+,   K1+,   M4-,   A3--
            xr11:8=q[j0+=4];  r13:12=l[k7+k3]; fr7=r3*r5;    fr25=r21+r17, fr27=r21-r17;;    // F2+,   K2+,   M2,    A4--
            q[j1+=4]=r25:24;                    fr14=r2*r5;   fr19=r14+r15;;                  // S1--,          M3,    A2-
            q[j2+=4]=r27:26;  k5=k5+k9;        fr6=r10*r12;  fr16=r6-r7;;                    // S2--,  K3+,   M1+,   A1

            yr23:20=q[j0+=4]; k3=k5 and k4;    fr15=r3*r4;   fr24=r28+r18, fr26=r28-r18;;    // F1++,  K1++,  M4,    A3-
            xr23:20=q[j0+=4]; r5:4=l[k7+k3];   fr7=r11*r13;  fr25=r29+r19, fr27=r29-r19;;    // F2++,  K2++,  M2+,   A4-
            q[j1+=4]=r25:24;                   fr14=r10*r13; fr17=r14+r15;;                  // S1-,           M3+,   A2
            q[j2+=4]=r27:26;  k5=k5+k9;        fr6=r22*r4;   fr18=r6-r7;;                    // S2-,   K3++,  M1++,  A1+

            yr31:28=q[j0+=4]; k3=k5 and k4;    fr15=r11*r12; fr24=r0+r16,  fr26=r0-r16;;     // F1+++, K1+++, M4+,   A3
            xr31:28=q[j0+=4]; r13:12=l[k7+k3]; fr7=r23*r5;   fr25=r1+r17,  fr27=r1-r17;;     // F2+++, K2+++, M2++,  A4
.align_code 4;
            if NLC0E, jump _BflyLoop;
            q[j1+=4]=r25:24; fr14=r22*r5;  fr19=r14+r15;;                                    // S1,            M3++,  A2+
```

列表2　fft32.asm---分片

N	ADSP-TS101 *老结构*	*ADSP-TS101 新结构*	*ADSP-TS201 老结构*	*输入不在缓存 ADSP-TS201 新结构*	*输入不在缓存 ADSP-TS201 老结构*	*输入在缓存 ADSP-TS201 新结构 输入在缓存*
256	*2172*	*1958*	*2641*	*2402*	*2218*	*1963*
512	*4582*	*4276*	*5533*	*5192*	*4649*	*4283*
1024	*9872*	*9410*	*12170*	*11662*	*9992*	*9419*
2048	*21338*	*20688*	*26610*	*25316*	*22173*	*20699*
4096	*46244*	*45278*	*197272*	*69924*	*NA*	*NA*
8192	*99886*	*98540*	*444628*	*147628*	*NA*	*NA*
16384	*215224*	*213243*	*987730*	*313292*	*NA*	*NA*
32768	*NA*	*NA*	*2133220*	*662614*	*NA*	*NA*
65536	*NA*	*NA*	*4720010*	*1397544*	*NA*	*NA*

表7　N点复数FFT内核时钟周期　新结构对老结构

使用准则

C可调用复数FFT例行程序称为

FFT32(&(input), &(ping_pong_buffer1), &(ping_pong_buffer2), &(output), N, F);

此外

input -> FFT 输入缓冲器，

output -> FFT 输出缓冲器，

*ping_pong_bufferx*为ping pong缓冲器，

*N=*复数点的数量

F=0 若FFT为实数；F=*1* 若FFT为复数。

如前所述，由于数据的重新排序，进程不能原封不动，需进行pingpong操作。这样，*ping_pong_buffer1*和*ping_pong_buffer2*就需要是两个截然不同的缓冲器。不过，取决于例行程序的用户要求，某些内存优化是可以办到的。如果*input*不需要保留，那么*Ping_pong_buffer1*可以作为*input*缓冲器同效使用。此外，如果*Log2(N)*为偶数，那么*output*缓冲器可以与*ping_pong_buffer2*同效使用；如果*Log2(N)*为奇数，那么*output*与*ping_pong_buffer1*同效。下面是例行程序用法的两个例子，具有最小的使用内存：

```
FFT32( &(input), &( input),
&( output), &(output), 1024, 1);
FFT32( &(input), &(input),
&( ping_pong_buffer2), &( input), 2048, 1);
```

为消除内存块访问冲突，*input*必须驻留在一个有别于*ping_pong_buffer2*的内存块中，旋转因子必须驻留在一个有别于pingpong缓冲器的内存块中。当然，全部代码也必须驻留在一个与所有数据缓冲器都不同的单元块中。但Ping-pong缓冲器可能共享一个内存块---同一周期内没有指令同时访问两个ping-pong缓冲器。

附录

优化FFT的完整源代码

```
/*  fft32.asm

      Prelim rev.     October 19, 2003 - BL
      Rev. 1.0 - added real inputs case - PM

      This is assembly routine for the Complex radix-2 C-callable FFT on TigerSHARC
      family of DSPs.

      I. Description of Calling.

            1. Inputs:
                  j4 -> input (ping-pong buffer 1)
                  j5 -> ping-pong buffer 1
                  j6 -> ping-pong buffer 2
                  j7 -> output
                  j27+0x18 -> N = Number of points
                  j27+0x19 -> REAL or COMPLEX

            2. C-Calling Example:
                  fft32(&(input), &(ping_pong_buffer1), &(ping_pong_buffer2), &(output), N, COMPLEX);

            3. Limitations:
                  a. All buffers must be aligned on memory boundary which is a multiple of 4.
                  b. N must be between 32 and MAX_FFT_SIZE.
                  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.
                  d. If memory space savings are required, output can be the same buffer
                     as ping_pong_buffer2 if the number of FFT stages is even (i.e.
                     Log2(N) is even) or the same as ping_pong_buffer1 if the number of
                     FFT stages is odd (i.e. Log2(N) is odd).

            4. MAX_FFT_SIZE can be selected via #define. Larger values allow for more choices
               of N, but its twiddles will occupy more memory.

            5. This C - callable function can process up to 64K blocks of data on TS201
               (16K blocks on TS101) because C environment itself necessitates memory.
               Therefore, if more input points are necessary, assembly language development
               may become a must. On TS201, a block of memory is 128K words long, so
               maximum N is 128K real points or 64K complex points. TS101 contains
               only 2 blocks of data memory of 64K words and 4 buffers must be
               accommodated. Therefore, maximum N is 32K real words or 16K complex words.

      II. Description of the FFT algorithm.

            1. The input data is treated as complex interleaved N-point.

            2. Due to re-ordering, no stage can be done in-place.

            3. The bit reversal and the first two stages are combined into
               a single loop. This loop takes data from input and stores it
               in the ping-pong buffer1.

            4. Each subsequent stage ping-pongs the data between the two ping-pong
               buffers. The last stage uses FFT output buffer for its output.

            5. Although the FFT is designed to be called with any point size
               N <= MAX_FFT_SIZE by subsampling the twiddle factors, for ADSP-TS20x
               processors, the best cycle optimization is achieved when MAX_FFT_SIZE=N.
               For ADSP-TS101 all choices of MAX_FFT_SIZE are equally optimal.

      III. Description of the REAL FFT algorithm.

            1. The input data is treated as complex interleaved N/2-point. The N/2 point complex
               FFT will be computed first. Thus, N is halved, now number of points = N/2.

            2. Details and source code of the N/2 point complex FFT are in II above.

            3. Real re-combine:
                  Here the complex N/2-point FFT computed in the previous steps is recombined to
                  produce the N-point real FFT. If G is the complex FFT and F is the real FFT,
                  the formula for F is given by:

                  F(n) = 0.5*(G(n)+conj(G(N/2-n))-0.5*i*exp(-2*pi*i*n/N)*(G(n)-conj(G(N/2-n)).

                  From this the following can be derived:

                  conj(F(N/2-n)) = 0.5*(G(n)+conj(G(N/2-n))+0.5*i*exp(-2*pi*i*n/N)*(G(n)-conj(G(N/2-n)).

                  Thus, this can be computed in (n,N/2-n) pairs, as follows (dropping factor of 2):

                  G(n) ------------------------------->------------------------->--------> F(n)
                                                \ +/                        \ +/
                                                 \/                          \/
                                                 /\                          /\
                  conj                          / -\  exp(-2*pi*i*n)*i      / -\   conj
                  G(N/2-n) -----> conj(G(N/2-n))------>------------------------->--------> F(N/2-n)

                  This is very efficient on the TigerSHARC architecture due to the add/subtract
                  instruction.
      IV. For all additional details regarding this algorithm and code, see EE-218
```

```
         application note, available from the ADI web site.
*/
//************************************ Includes *********************************

#include "FFTDef.h"
#include "defts201.h"

//************************* Externs ************************************

.extern _twiddles;

//******************************** FFT Routine ******************************
.section program;
.global _FFT32;

_FFT32:

//********************************** Prologue *********************************

      mENTER
      mPUSHQ(xR31:28)
      mPUSHQ(xR27:24)
      mPUSHQ(yR31:28)
      mPUSHQ(yR27:24)

//************************************ Setup **********************************
      j17 = [j27 + 0x18];;                                      //j17 = N
      j11 = [j27 + 0x19];;                                      // j11=COMPLEX or REAL, off the stack

      comp(j11,COMPLEX);;                                       // Complex or Real?
      if jeq, jump _FFTStages1and2;;
      j17=ashiftr j17;;                                         // if Real, half N

//*****************************************************************************
_FFTStages1and2:

      j11 = j31 + j17;;                                         // j11=N
      xr3=j11; k7=k31+_twiddles;;
      k1=j11; j8=lshiftr j11;;                                  // k1=N, j8=N/2
      j9=lshiftr j8; xr0=MAX_FFT_SIZE; xr3=LD0 r3;;             // j9=N/4, compute the twiddle stride
      k8=lshiftr k1; xr0=LD0 r0; xr1=j11;;
      k8=lshiftr k8; xr1=LD0 r1; xr2=(31-3);;                   // k8=N/4, Compute Stages-3
      k0=j4; k10=lshiftr k8; xr1=r1-r0; xr0=lshift r0 by -32;;  // k0->input, xr1=bit difference between MAX and N
      k10=lshiftr k10; xr0=bset r0 by r1; xr30=r2-r3;;           // k10=N/16, xr30=Stages-3
      k10=k10-1; xr0=lshift r0 by 2; LC1=xr30;;                 // k10=N/16-1, LC1=Stages-3
      k9=xr0; k4=k31+(MAX_FFT_SIZE/4-1);;
      k4=not k4; j10=lshiftr j9;;                               // initial twiddles pointer mask, j10=N/8

//****************** Bit Reverse and Stages 1 & 2 *****************************

      k5=lshiftr k1;;                                           // k5=N/2
      j0=j31+j6; k6=k6-k6;;                                     // j0->ping_pong_buffer2
      j1=j0+j9; LC0=k10;;                                       // j1->ping_pong_buffer2+N/4, LC0=N/16-1
      j2=j1+j9; k1=k0+k5;;                                      // j2->ping_pong_buffer2+N/2, k1->input+N/2
      j3=j2+j9; k2=k1+k5;;                                      // j3->ping_pong_buffer2+3N/4, k2->input+N
      j12=j3+j9; k3=k2+k5;;                                     // j12->ping_pong_buffer2+N, k3->input+3N/2
      j13=j12+j9; k5=lshiftr k5;;                               // j13->ping_pong_buffer2+5N/4, k5=N/4
      j14=j13+j9; r1:0=q[k0+k6];;                               // j14->ping_pong_buffer2+3N/2
      j15=j14+j9; r3:2=q[k2+k6];;                               // j15->ping_pong_buffer2+7N/4

      r5:4=q[k1+k6];;
      r7:6=q[k3+k6];;

      k6=k6+k5 (br); fr0=r0+r2, fr20=r0-r2;;
      r9:8=q[k0+k6]; fr2=r1+r3, fr29=r1-r3;;
      r11:10=q[k2+k6]; fr4=r4+r6, fr21=r4-r6;;
      r13:12=q[k1+k6]; fr5=r5+r7, fr28=r5-r7;;

      r15:14=q[k3+k6]; fr18=r8+r10, fr22=r8-r10;;
      k6=k6+k5 (br); fr19=r9+r11, fr31=r9-r11;;
      fr26=r12+r14, fr23=r12-r14;;
      fr27=r13+r15, fr30=r13-r15;;

      fr20=r20+r28, fr28=r20-r28;;
      fr29=r29+r21, fr21=r29-r21;;
      fr22=r22+r30, fr30=r22-r30;;
      fr31=r31+r23, fr23=r31-r23;;

.align_code 4;
_Stages1and2Loop:
              r1:0=q[k0+k6]; q[j2+=4]=yr23:20; fr16=r0+r4, fr24=r0-r4;;
              r3:2=q[k2+k6]; q[j3+=4]=xr23:20; fr17=r2+r5, fr25=r2-r5;;
              r5:4=q[k1+k6]; q[j14+=4]=yr31:28; fr18=r18+r26, fr26=r18-r26;;
              r7:6=q[k3+k6]; q[j15+=4]=xr31:28; fr19=r19+r27, fr27=r19-r27;;

              k6=k6+k5 (br); q[j0+=4]=yr19:16; fr0=r0+r2, fr20=r0-r2;;
              r9:8=q[k0+k6]; q[j1+=4]=xr19:16; fr2=r1+r3, fr29=r1-r3;;
              r11:10=q[k2+k6]; q[j12+=4]=yr27:24; fr4=r4+r6, fr21=r4-r6;;
              r13:12=q[k1+k6]; q[j13+=4]=xr27:24; fr5=r5+r7, fr28=r5-r7;;

              r15:14=q[k3+k6]; fr18=r8+r10, fr22=r8-r10;;
              k6=k6+k5 (br); fr19=r9+r11, fr31=r9-r11;;
              fr26=r12+r14, fr23=r12-r14;;
              fr27=r13+r15, fr30=r13-r15;;

              fr20=r20+r28, fr28=r20-r28;;
              fr29=r29+r21, fr21=r29-r21;;
              fr22=r22+r30, fr30=r22-r30;;
```

```
.align_code 4;
              if NLC0E, jump _Stages1and2Loop;
              fr31=r31+r23, fr23=r31-r23;;

      q[j2+=4]=yr23:20; fr16=r0+r4, fr24=r0-r4;;
      q[j3+=4]=xr23:20; fr17=r2+r5, fr25=r2-r5;;
      q[j14+=4]=yr31:28; fr18=r18+r26, fr26=r18-r26;;
      q[j15+=4]=xr31:28; fr19=r19+r27, fr27=r19-r27;;

      q[j0+=4]=yr19:16;;
      q[j1+=4]=xr19:16;;
      q[j12+=4]=yr27:24;;
      q[j13+=4]=xr27:24;;

//********************** Stages 3 to Log2(N)-1  ******************************

      j0=j31+j6; k5=k31+0;;

.align_code 4;
_StageLoop:
              yr3:0=q[j0+=4];   k3=k5 and k4;;                                                    // F1,    K1
              xr3:0=q[j0+=4];   r5:4=l[k7+k3];;                                                   // F2,    K2
              LC0=k10;          k5=k5+k9;;                                                        //        K3,    M1

              yr11:8=q[j0+=4];  k3=k5 and k4;    fr6=r2*r4;;                                      // F1+,   K1+
              xr11:8=q[j0+=4];  r13:12=l[k7+k3]; fr7=r3*r5;;                                      // F2+,   K2+,   M2
                                                          fr14=r2*r5;;                            //                      M3
              j1=j31+j5;        k5=k5+k9;        fr6=r10*r12;  fr16=r6-r7;;                       //        K3+,   M1+,   A1

              yr23:20=q[j0+=4]; k3=k5 and k4;    fr15=r3*r4;;                                     // F1++,  K1++,  M4
              xr23:20=q[j0+=4]; r5:4=l[k7+k3];   fr7=r11*r13;;                                    // F2++,  K2++,  M2+
                                                 fr14=r10*r13; fr17=r14+r15;;                     //                M3+,   A2
              j2=j1+j11;         k5=k5+k9;        fr6=r22*r4;   fr18=r6-r7;;                     //        K3++,  M1++,  A1+

              yr31:28=q[j0+=4]; k3=k5 and k4;    fr15=r11*r12; fr24=r0+r16, fr26=r0-r16;;         // F1+++, K1+++, M4+,   A3
              xr31:28=q[j0+=4]; r13:12=l[k7+k3]; fr7=r23*r5;   fr25=r1+r17, fr27=r1-r17;;         // F2+++, K2+++, M2++,  A4
              q[j1+=4]=r25:24;                   fr14=r22*r5;  fr19=r14+r15;;                     // S1,            M3++,  A2+

.align_code 4;
_BflyLoop:
                    q[j2+=4]=r27:26;  k5=k5+k9;        fr6=r30*r12;  fr16=r6-r7;;                // S2----,K3-,   M1-,   A1--

                    yr3:0=q[j0+=4];   k3=k5 and k4;    fr15=r23*r4;  fr24=r8+r18,  fr26=r8-r18;; // F1,    K1,    M4--,  A3---
                    xr3:0=q[j0+=4];   r5:4=l[k7+k3];   fr7=r31*r13;  fr25=r9+r19,  fr27=r9-r19;; // F2,    K2,    M2-,   A4---
                    q[j1+=4]=r25:24;                   fr14=r30*r13; fr17=r14+r15;;               // S1---,         M3-,   A2--
                    q[j2+=4]=r27:26;  k5=k5+k9;        fr6=r2*r4;    fr18=r6-r7;;                // S2---, K3,     M1,    A1-

                    yr11:8=q[j0+=4];  k3=k5 and k4;    fr15=r31*r12; fr24=r20+r16, fr26=r20-r16;; // F1+,   K1+,   M4-,   A3--
                    xr11:8=q[j0+=4];  r13:12=l[k7+k3]; fr7=r3*r5;    fr25=r21+r17, fr27=r21-r17;;  // F2+,   K2+,   M2,    A4--
                    q[j1+=4]=r25:24;                     fr14=r2*r5;   fr19=r14+r15;;              // S1--,          M3,    A2-
                    q[j2+=4]=r27:26;  k5=k5+k9;        fr6=r10*r12;  fr16=r6-r7;;                // S2--,  K3+,   M1+,   A1

                    yr23:20=q[j0+=4]; k3=k5 and k4;    fr15=r3*r4;   fr24=r28+r18, fr26=r28-r18;; // F1++,  K1++,  M4,    A3-
                    xr23:20=q[j0+=4]; r5:4=l[k7+k3];   fr7=r11*r13;  fr25=r29+r19, fr27=r29-r19;; // F2++,  K2++,  M2+,   A4-
                    q[j1+=4]=r25:24;                   fr14=r10*r13; fr17=r14+r15;;               // S1-,           M3+,   A2
                    q[j2+=4]=r27:26;  k5=k5+k9;        fr6=r22*r4;   fr18=r6-r7;;                // S2-,   K3++,  M1++,  A1+

                    yr31:28=q[j0+=4]; k3=k5 and k4;    fr15=r11*r12; fr24=r0+r16,  fr26=r0-r16;; // F1+++, K1+++, M4+,   A3
                    xr31:28=q[j0+=4]; r13:12=l[k7+k3]; fr7=r23*r5;   fr25=r1+r17,  fr27=r1-r17;; // F2+++, K2+++, M2++,  A4
.align_code 4;
              if NLC0E, jump _BflyLoop;
              q[j1+=4]=r25:24; fr14=r22*r5;  fr19=r14+r15;;                                       // S1,            M3++,  A2+

              q[j2+=4]=r27:26;                   fr6=r30*r12;  fr16=r6-r7;;                      // S2----,        M1-, A1--

              j0=j31+j5;                         fr15=r23*r4;  fr24=r8+r18,  fr26=r8-r18;;       //                M4--,  A3---
                                                                                                 // swap ping-pong pointers
              j5=j31+j6;                         fr7=r31*r13;  fr25=r9+r19,  fr27=r9-r19;;       //                M2-,   A4---
              q[j1+=4]=r25:24;                   fr14=r30*r13; fr17=r14+r15;;                    // S1---,         M3-,   A2--
              q[j2+=4]=r27:26;                                 fr18=r6-r7;;                      // S2---,                A1-

              j6=j31+j0;                         fr15=r31*r12; fr24=r20+r16, fr26=r20-r16;;      //                M4-,   A3--
                                                               fr25=r21+r17, fr27=r21-r17;;      //                       A4--
              q[j1+=4]=r25:24;                                             fr19=r14+r15;;       // S1--,                 A2-
              q[j2+=4]=r27:26;                   fr24=r28+r18, fr22=r28-r18;;                    // S2--                  A3-

              j0=j31+j6;                         fr25=r29+r19, fr23=r29-r19;;                    //                       A4-
              q[j1+=4]=r25:24; k5=k31+0;;                                                        // S1-
.align_code 4;
              if NLC1E, jump _StageLoop;
              q[j2+=4]=r23:22; k4=ashiftr k4;;                                                   // S2-, shift the mask

//****************************** Last stage *********************************
  k9 = ashiftr k9;;//in this manner any MAX_FFT_SIZE can be used

      yr3:0=q[j0+=4]; yr5:4 = l[k7+=k9];;                                                        // F1,
      xr3:0=q[j0+=4];   xr5:4=l[k7+=k9];;                                                        // F2,    K2
      j1=j31+j7;        fr6=r2*r4; LC0=k10;;                                                     //        M1

      yr11:8=q[j0+=4];  yr13:12=l[k7+=k9];;                                                      // F1+
      xr11:8=q[j0+=4];  xr13:12=l[k7+=k9]; fr7=r3*r5;;                                           // F2+,   K2+,   M2
      j2=j1+j11;                           fr14=r2*r5;;                                          //        M3
                                           fr6=r10*r12;  fr16=r6-r7;;                            //        M1+,   A1

      yr23:20=q[j0+=4]; yr5:4=l[k7+=k9];  fr15=r3*r4;;                                           // F1++,  M4
      xr23:20=q[j0+=4]; xr5:4=l[k7+=k9];   fr7=r11*r13;;                                         // F2++,  K2++,  M2+
                                          fr14=r10*r13; fr17=r14+r15;;                           //               M3+,   A2
                                          fr6=r22*r4;   fr18=r6-r7;;                             //               M1++,  A1+
```

```
      yr31:28=q[j0+=4]; yr13:12=l[k7+=k9];fr15=r11*r12; fr24=r0+r16, fr26=r0-r16;;                    // F1+++,        M4+,   A3
      xr31:28=q[j0+=4]; xr13:12=l[k7+=k9]; fr7=r23*r5;   fr25=r1+r17, fr27=r1-r17;;                  // F2+++, K2+++, M2++,  A4
      q[j1+=4]=r25:24;                     fr14=r22*r5;  fr19=r14+r15;;                              // S1,           M3++,  A2+

.align_code 4;
_BflyLastLoop:
              q[j2+=4]=r27:26;                   fr6=r30*r12;  fr16=r6-r7;;                        // S2----,       M1-,   A1--

              yr3:0=q[j0+=4];   yr5:4=l[k7+=k9];  fr15=r23*r4;  fr24=r8+r18,  fr26=r8-r18;;        // F1,           M4--,  A3---
              xr3:0=q[j0+=4];   xr5:4=l[k7+=k9];  fr7=r31*r13;  fr25=r9+r19,  fr27=r9-r19;;        // F2,    K2,    M2-,   A4---
              q[j1+=4]=r25:24;                    fr14=r30*r13; fr17=r14+r15;;                      // S1---,        M3-,   A2--
              q[j2+=4]=r27:26;                    fr6=r2*r4;    fr18=r6-r7;;                        // S2---,        M1,    A1-

              yr11:8=q[j0+=4];  yr13:12=l[k7+=k9];fr15=r31*r12; fr24=r20+r16, fr26=r20-r16;;        // F1+,          M4-,   A3--
              xr11:8=q[j0+=4];  xr13:12=l[k7+=k9];fr7=r3*r5;    fr25=r21+r17, fr27=r21-r17;;        // F2+,   K2+,   M2,    A4--
              q[j1+=4]=r25:24;                    fr14=r2*r5;   fr19=r14+r15;;                      // S1--,         M3,    A2-
              q[j2+=4]=r27:26;                    fr6=r10*r12;  fr16=r6-r7;;                        // S2--,         M1+,   A1

              yr23:20=q[j0+=4]; yr5:4=l[k7+=k9];  fr15=r3*r4;   fr24=r28+r18, fr26=r28-r18;;        // F1++,         M4,    A3-
              xr23:20=q[j0+=4]; xr5:4=l[k7+=k9];  fr7=r11*r13;  fr25=r29+r19, fr27=r29-r19;;        // F2++,  K2++,  M2+,   A4-
              q[j1+=4]=r25:24;                    fr14=r10*r13; fr17=r14+r15;;                      // S1-,          M3+,   A2
              q[j2+=4]=r27:26;                    fr6=r22*r4;   fr18=r6-r7;;                        // S2-,          M1++,  A1+

              yr31:28=q[j0+=4];yr13:12=l[k7+=k9]; fr15=r11*r12; fr24=r0+r16,  fr26=r0-r16;;         // F1+++,        M4+,   A3
              xr31:28=q[j0+=4];xr13:12=l[k7+=k9]; fr7=r23*r5;   fr25=r1+r17,  fr27=r1-r17;;         // F2+++, K2+++, M2++,  A4
.align_code 4;
      if NLC0E, jump _BflyLastLoop;
      q[j1+=4]=r25:24;  fr14=r22*r5;  fr19=r14+r15;;                                                // S1,           M3++,  A2+

      q[j2+=4]=r27:26;                  fr6=r30*r12;  fr16=r6-r7;;                                  // S2----,       M1-, A1--
                                        fr15=r23*r4;  fr24=r8+r18,  fr26=r8-r18;;                   //               M4--,  A3---
                                        fr7=r31*r13;  fr25=r9+r19,  fr27=r9-r19;;                   //               M2-,   A4---
      q[j1+=4]=r25:24;                  fr14=r30*r13; fr17=r14+r15;;                                // S1---,        M3-,   A2--
      q[j2+=4]=r27:26;                                fr18=r6-r7;;                                  // S2---,                A1-
                                        fr15=r31*r12; fr24=r20+r16, fr26=r20-r16;;                  //               M4-,   A3--
                                                      fr25=r21+r17, fr27=r21-r17;;                  //                      A4--
      q[j1+=4]=r25:24;                                            fr19=r14+r15;;                  // S1--,                 A2-
      q[j2+=4]=r27:26;;                                                                           // S2--
                                                      fr24=r28+r18, fr26=r28-r18;;                  //                      A3-
                                                      fr25=r29+r19, fr27=r29-r19;;                  //                      A4-
      q[j1+=4]=r25:24;;                                                                           // S1-
      q[j2+=4]=r27:26;;                                                                           // S2-
      j11=[j27+0x19];;                                                                  // j11=COMPLEX or REAL, off the stack
      comp(j11,COMPLEX);;                                                               // Complex or Real?
.align_code 4;
      if jeq, jump _FFTEpilogue;;                                                       // If Complex, done

//****************************** Real re-combine *******************************

                                                                                //j17=N/2, j7=output
      k8=k31+_twiddles; j0=j31+j7;;                                             // k8->twiddles, j0->internal buffer
      k9=ashiftr k9; j10=j31+j7;;                                               // k9=twiddle stride, j10->internal buffer
      j14=j17+j17;;                                                             // j14=N (N/2 complex values)
      j14=j14-4;;                                                               // j14=N-4 real=N/2-2 complex
      j1=j0+j14;;                                                               // j1->internal buffer+(N/2-2)
      j14=j10+j14;;                                                             // j14->internal buffer+(N/2-2)
      j29 = ashiftr j17;;                                                       // j29=N/4
      k15=k31+MAX_FFT_SIZE/4; j30=ashiftr j29;;                                 // k15=N/4*twiddle_stride, j30=N/8
      j30 = ashiftr j30;;                                                       // N/16
      k8=k8+k9; r0=l[j7+j17];;                                                  // k8->twiddles+1, get G(N/4)
      j0=j0+2; k12=k8+k15;;                                                     // j0->internal buffer+1, k12->twiddles+N/8+1
      LC0=j30; fr0=r0+r0; j2=j0+j29;;                                           // LC0=N/16, compute F(N/4)=2*conj(G(N/4)),
                                                                                // j2->internal buffer+1+N/8
      j3=j1-j29;;                                                               // j3->internal buffer+3N/8-2
      xfr0=-r0; j10=j10+2; k10=yr0;;                                            // j10->internal buffer+1, k10=Im(F(N/4))
      j12=j10+j29;;                                                             // j12->internal buffer+N/8+1
      if LC0E; j13=j14-j29; k11=xr0;;                                           // LC0=N/16-1, j13->internal buffer+3N/8-2, k11=Re(F(N/4))
      yr3:0=DAB q[j0+=4];;                                                      // Prime the DAB
      xr3:0=DAB q[j2+=4];;                                                      // Prime the DAB
      yr3:0=DAB q[j0+=4];;                                                      // yr0=Re(G(n)), yr1=Im(G(n)), yr2=Re(G(n+1)), yr3=Im(G(n+1))
      xr3:0=DAB q[j2+=4];;                                                      // xr0=Re(G(n+N/8)), xr1=Im(G(n+N/8))
                                                                                // xr2=Re(G(n+1+N/8)), xr3=Im(G(n+1+N/8))
      yr7:4=q[j1+=-4]; xr9:8=l[k12+=k9];;                                       // yr4=Re(G(N/2-(n+1))), yr5=Im(G(N/2-(n+1)))
                                                                                // yr6=Re(G(N/2-n)), yr7=Im(G(N/2-n))
                                                                                // twiddles(n+N/8) - want to mult by sin(x)-icos(x)
      xr7:4=q[j3+=-4]; xr11:10=l[k12+=k9];;                                     // xr4=Re(G(N/2-(n+1+N/8))), xr5=Im(G(N/2-(n+1+N/8)))
                                                                                // xr6=Re(G(N/2-(n+N/8))), xr7=Im(G(N/2-(n+N/8)))
                                                                                // twiddles(n+1+N/8)
      if LC0E; fr16=r0+r6, fr20=r0-r6; yr9:8=l[k8+=k9];;                        // LC0=N/16-2, r16=Re(G(n)+conj(G(N/2-n))),
                                                                                // r20=Re(G(n)-conj(G(N/2-n)))
                                                                                // twiddles(n)
      fr18=r2+r4, fr22=r2-r4; yr11:10=l[k8+=k9];;                               // r18=Re(G(n+1)+conj(G(N/2-(n+1)))),
                                                                                // r22=Re(G(n+1)-conj(G(N/2-(n+1))))
                                                                                // twiddles(n+1)
      fr24=r20*r9; fr21=r1+r7, fr17=r1-r7;;                                     // r24=s(n)*Re(G(n)-conj(G(N/2-n)))
                                                                                // r17=Im(G(n)+conj(G(N/2-n))), r21=Im(G(n)-conj(G(N/2-n)))
      fr26=r22*r11; fr23=r3+r5, fr19=r3-r5; xr3:0=DAB q[j2+=4];;                // r26=s(n+1)*Re(G(n+1)-conj(G(N/2-(n+1))))
                                                                                // r19=Im(G(n+1)+conj(G(N/2-(n+1)))),
                                                                                // r23=Im(G(n+1)-conj(G(N/2-(n+1))))
                                                                                // xr3:0=next G(n+2+N/8), G(n+3+N/8)
      fr25=r21*r8;  yr3:0=DAB q[j0+=4];;                                        // r25=c(n)*Im(G(n)-conj(G(N/2-n))),
                                                                                // yr3:0=next G(n+2), G(n+3)
      fr27=r23*r10;;                                                            // r27=c(n+1)*Im(G(n+1)-conj(G(N/2-(n+1))))
      fr24=r24+r25; fr25=r21*r9; yr7:4=q[j1+=-4];;                              // r24=Re(-i*exp(2*pi*i*n)(G(n)-conj(G(N/2-n))))
                                                                                // r13=s(n)*Im(G(n)-conj(G(N/2-n)))
                                                                                // yr7:4=next G(N/2-(n+2)), G(N/2-(n+3))
      fr26=r26+r27; fr27=r23*r11; xr7:4=q[j3+=-4];;                             // r26=Re(-i*exp(2*pi*i*(n+1))(G(n+1)-conj(G(N/2-(n+1)))))
```

```
                                                                        // r27=s(n+1)*Im(G(n+1)-conj(G(N/2-(n+1))))
                                                                        // xr7:4=next G(N/2-(n+2+N/8)), G(N/2-(n+3+N/8))
      fr13=r20*r8; fr12=r16+r24, fr30=r16-r24;;                         // r13=c(n)*Re(G(n)-conj(G(N/2-n))),
                                                                        // r12=Re(F(n)), r30=Re(F(N/2-n))
      fr15=r22*r10; fr14=r18+r26, fr28=r18-r26;;                        // r15=c(n+1)*Re(G(n+1)-conj(G(N/2-(n+1))))
                                                                        // r14=Re(F(n+1)), r28=Re(F(N/2-(n+1)))
      fr13=r25-r13; xr9:8=l[k12+=k9];;                                  // r13=Im(-i*exp(2*pi*i*x)(G(n)-conj(G(N/2-n)))),
                                                                        // next twiddles(n+2+N/8)
      fr15=r27-r15; xr11:10=l[k12+=k9];;                                // r15=Im(-i*exp(2*pi*i*x)(G(n+1)-conj(G(N/2-(n+1)))))
                                                                        // next twiddles(n+3+N/8)
.align_code 4;
_combine_stage:
              fr16=r0+r6, fr20=r0-r6; yr9:8=l[k8+=k9];;                 // r16=Re(G(n+2)+conj(G(N/2-(n+2)))),
                                                                        // r20=Re(G(n+2)-conj(G(N/2-(n+2))))
                                                                        // next twiddles(n+2)
              fr18=r2+r4, fr22=r2-r4; yr11:10=l[k8+=k9];;               // r18=Re(G(n+3)+conj(G(N/2-(n+3)))),
                                                                        // r22=Re(G(n+3)-conj(G(N/2-(n+3))))
                                                                        // next twiddles(n+3)
              fr13=r13+r17, fr31=r13-r17;;                              // r13=Im(F(n)), r31=Im(F(N/2-n))
              fr15=r15+r19, fr29=r15-r19; l[j12+=2]=xr13:12;;           // r15=Im(F(n+1)), r29=Im(F(N/2-(n+1))), store F(n+N/8)
              fr24=r20*r9; fr21=r1+r7, fr17=r1-r7; q[j14+=-4]=yr31:28;;  // r24=s(n+2)*Re(G(n+2)-conj(G(N/2-(n+2))))
                                                                        // r21=Im(G(n+2)+conj(G(N/2-(n+2)))),
                                                                        // r17=Im(G(n+2)-conj(G(N/2-(n+2))))
                                                                        // store F(N/2-n), F(N/2-(n+1))
              fr26=r22*r11; fr23=r3+r5, fr19=r3-r5; xr3:0=DAB q[j2+=4];;  // r26=s(n+3)*Re(G(n+3)-conj(G(N/2-(n+3))))
                                                                        // r23=Im(G(n+3)+conj(G(N/2-(n+3)))),
                                                                        // r19=Im(G(n+3)-conj(G(N/2-(n+3))))
                                                                        // xr3:0=next G(n+4+N/8), G(n+5+N/8)
              fr25=r21*r8; yr3:0=DAB q[j0+=4];;                         // r25=c(n+2)*Im(G(n+2)-conj(G(N/2-(n+2))))
                                                                        // yr3:0=next G(n+4), G(n+5)
              fr27=r23*r10; q[j13+=-4]=xr31:28;;                        // r27=c(n+3)*Im(G(n+3)-conj(G(N/2-(n+3))))
                                                                        // store F(N/2-(n+N/8)), F(N/2-(n+1+N/8))
              fr24=r24+r25; fr25=r21*r9; l[j10+=2]=yr13:12;;            // r24=Re(-i*exp(2*pi*i*x)(G(n+2)-conj(G(N/2-(n+2)))))
              fr26=r26+r27; fr27=r23*r11; l[j10+=2]=yr15:14;;           // r25=s(n+2)*Im(G(n+2)-conj(G(N/2-(n+2)))), store F(n)
                                                                        // r26=Re(-i*exp(2*pi*i*x)(G(n+3)-conj(G(N/2-(n+3)))))
                                                                        // r27=s(n+3)*Im(G(n+3)-conj(G(N/2-(n+3)))), store F(n+1)
              fr13=r20*r8; fr12=r16+r24, fr30=r16-r24; l[j12+=2]=xr15:14;;  // r13=cos(n+2)*Re(G(n+2)-conj(G(N/2-(n+2))))
                                                                        // r12=Re(F(n+2)), r30=Re(F(N/2-(n+2))), store F(n+1+N/8)
              fr15=r22*r10; fr14=r18+r26, fr28=r18-r26; xr7:4=q[j3+=-4];;  // r15=cos(n+3)*Re(G(n+3)-conj(G(N/2-(n+3))))
                                                                        // r14=Re(F(n+3)), r28=Re(F(N/2-(n+3)))
                                                                        // xr7:4=next G(N/2-(n+4+N/8)), G(N/2-(n+5+N/8))
              fr13=r25-r13; xr9:8=l[k12+=k9]; yr7:4=q[j1+=-4];;         // r13=Im(-i*exp(2*pi*i*x)(G(n+2)-conj(G(N/2-(n+2)))))
                                                                        // next twiddles(n+4+N/8)
                                                                        // yr7:4=next G(N/2-(n+4)), G(N/2-(n+5))
.align_code 4;
              if NLC0E, jump _combine_stage(P); fr15=r27-r15; xr11:10=l[k12+=k9];;// r15=Im(-i*exp(2*pi*i*x)(G(n+3)-conj(G(N/2-(n+3)))))
                                                                        // next twiddles(n+5+N/8)
      fr16=r0+r6, fr20=r0-r6; yr9:8=l[k8+=k9];;                         // r16=Re(G(n+4)+conj(G(N/2-(n+4)))),
                                                                        // r20=Re(G(n+4)-conj(G(N/2-(n+4))))
                                                                        // next twiddles(n+4)
      fr18=r2+r4, fr22=r2-r4; yr11:10=l[k8+=k9];;                       // r18=Re(G(n+5)+conj(G(N/2-(n+5)))),
                                                                        // r22=Re(G(n+5)-conj(G(N/2-(n+5))))
                                                                        // next twiddles(n+5)
      fr13=r13+r17, fr31=r13-r17;;                                      // r13=Im(F(n+2)), r31=Im(F(N/2-(n+2)))
      fr15=r15+r19, fr29=r15-r19;;                                      // r15=Im(F(n+3)), r29=Im(F(N/2-(n+3)))
      fr24=r20*r9; fr21=r1+r7, fr17=r1-r7; yr1:0=l[j31+j7];;            // r24=s(n+4)*Re(G(n+4)-conj(G(N/2-(n+4))))
                                                                        // r21=Im(G(n+4)+conj(G(N/2-(n+4)))),
                                                                        // r17=Im(G(n+4)-conj(G(N/2-(n+4))))
                                                                        // yr0=Re(G(0)), yr1=Im(G(0))
      fr26=r22*r11; fr23=r3+r5, fr19=r3-r5;;                            // r26=s(n+5)*Re(G(n+5)-conj(G(N/2-(n+5))))
                                                                        // r23=Im(G(n+5)+conj(G(N/2-(n+5)))),
                                                                        // r19=Im(G(n+5)-conj(G(N/2-(n+5))))
      fr25=r21*r8;;                                                     // r25=cos(x)*Im(G(n)-conj(G(N/2-n)))
      fr27=r23*r10; l[j12+=2]=xr13:12;;                                 // r27=cos(x)*Im(G(n)-conj(G(N/2-n)))
                                                                        // store F(n:2:N/8)
      yfr0=r1+r0; yr1=lshift r1 by -32; q[j14+=-4]=yr31:28;;            // yr0=Re(G(0))+Im(G(0)), yr1=0=Im(F(0))
                                                                        // store F(N/2-(n+2)), F(N/2-(n+3))
      yfr0=r0+r0; q[j13+=-4]=xr31:28;;                                  // yr0=Re(F(0))
                                                                        // store F(N/2-(n+2+N/8)), F(N/2-(n+3+N/8))
      fr24=r24+r25; fr25=r21*r9; l[j10+=2]=yr13:12;;                    // r24=Re(-i*exp(2*pi*i*x)(G(n+4)-conj(G(N/2-(n+4)))))
                                                                        // r25=s(n+4)*Im(G(n+4)-conj(G(N/2-(n+4)))), store F(n+2)
      fr26=r26+r27; fr27=r23*r11; l[j10+=2]=yr15:14;;                   // r26=Re(-i*exp(2*pi*i*x)(G(n+5)-conj(G(N/2-(n+5)))))
                                                                        // r27=s(n+5)*Im(G(n+5)-conj(G(N/2-(n+5)))), store F(n+3)
      fr13=r20*r8; fr12=r16+r24, fr30=r16-r24; l[j12+=2]=xr15:14;;      // r13=c(n+4)*Re(G(n+4)-conj(G(N/2-(n+4))))
                                                                        // r12=Re(F(n+4)), r30=Re(F(N/2-(n+4))), store F(n+3+N/8)
      fr15=r22*r10; fr14=r18+r26, fr28=r18-r26; l[j31+j7]=yr1:0;;       // r15=c(n+5)*Re(G(n+5)-conj(G(N/2-(n+5))))
                                                                        // r14=Re(F(n+5)), r28=Re(F(N/2-(n+5)))
                                                                        // store F(0)
      fr13=r25-r13; l[j7+j17]=k11:10;;                                  // r13=Im(-i*exp(2*pi*i*x)(G(n+4)-conj(G(N/2-(n+4)))))
                                                                        // store F(N/4)
      fr15=r27-r15;;                                                    // r15=Im(-i*exp(2*pi*i*x)(G(n+5)-conj(G(N/2-(n+5)))))
      fr13=r13+r17, fr31=r13-r17;;                                      // r13=Im(F(n+4)), r31=Im(F(N/2-(n+4)))
      fr15=r15+r19, fr29=r15-r19; l[j12+=2]=xr13:12;;                   // r15=Im(F(n+5)), r29=Im(F(N/2-(n+5))), store F(n+4+N/8)
      q[j14+=-4]=yr31:28;;                                              // store F(N/2-(n+4)), F(N/2-(n+5))
      q[j13+=-4]=xr31:28;;                                              // store F(N/2-(n+4+N/8)), F(N/2-(n+5+N/8))
      l[j10+=2]=yr13:12;;                                               // store F(n+4)
      l[j10+=2]=yr15:14;;                                               // store F(n+5)
      l[j12+=2]=xr15:14;;                                               // store F(n+5+N/8)

//****************************** Epilogue *********************************
_FFTEpilogue:

      mPOPQ(yR27:24)
      mPOPQ(yR31:28)
      mPOPQ(xR27:24)
      mPOPQ(xR31:28)
      mRETURN
```

列表 3. fft32.asm

参考文献

[1] *ADSP-TS201 TigerSHARC Processor Programming Reference.* Revision 0.1, June 2003. Analog Devices, Inc.

文件历史

修订	描述
修订版2，2004年3月4日　Boris Lerner提供	增加了实进程（real stage）的提法并适度更新了调用例子
修订版1，2003年12月18日 Boris Lerner提供	第一版