恩智浦半导体
应用笔记

文档编号：AN4945
第0版，2014年4月

利用恩智浦32位DSC指令集加速CFFT

使用Processor-Expert生成的代码的修改方法

作者：John L. Winters

本文介绍了如何使用位反转索引指令来提高恩智浦32位DSC（数字信号控制器）的CFFT性能。根据恩智浦文档AN4837《将传统DSC应用移植到恩智浦32位DSC系列》中介绍的方法，CFFT的测试工具的使用步骤如下：

1. 移植到32位DSC
2. 修改为使用位反转索引指令
3. 运行测试工具

测量了性能以展示改进前后的对比。本应用笔记的目标受众是对DSC感兴趣的人员，无论是否有集成开发环境（IDE）经验。

目录

1 项目工作概述

在之前的应用笔记（AN4837，《将传统DSC应用移植到恩智浦32位DSC系列》）中，我们介绍了将Processor Expert测试工具软件示例从早期DSC(DSP56858)迁移到当前DSC(MC56F84xxx)的方法。

在应用笔记AN4837中，首次使用类似的方法，将复数快速傅里叶变换（CFFT）测试工具的基于CW8.3 for DSC Processor Expert的示例代码从56858芯片迁移到MC56F84789。由此生成的工程作为本应用笔记的起点。

以上述工程作为起点，本文将通过使用为新指令集量身定制的函数替换其中一个生成的函数，进一步开发该工程。要替换的例程是进行位反转索引的例程。位反转索引与位反转寻址不同：在位反转索引中，索引首先进行位反转，然后递增1，再次进行位反转，而不是按照下一个整数值计数。要递增“1”，首先要将“1”进行位反转，使其占据log2N位右对齐索引字段的最高有效位；然后将其与索引相加，实现反向进位。这样就得到了位反转后的下一个索引。

利用一条能执行“反向进位加法”的指令，可以节省大量的CPU时间（最多35%）。然而，实现这一操作所需的逻辑非常简单，与普通加法器并无不同，唯一的区别在于参考框架是反向的。只需一个控制位就可以打开或关闭反向进位。反向进位的基数由添加到索引中的位的值确定，无需单独调整。当然，正如示例代码一样，这种方法具有局限性，仅限于N是2的整数幂的情况。

例如，如果$\log_2 N$为4（N=16，或16点CFFT），那么为加数设置的位数将是$\log_2 N$ -1或位3。用十六进制表示就是0x8。

将0x8与0相加并反向进位，得到序列0,8。

将0x8与8相加并反向进位，得到序列中的前三个值0,8,4。

完整的序列是0,8,4,C,2,A,6,E,1,9,5,D,3,B,7,F。

这正是CFFT算法所需的序列类型。

需要注意的是，由于在函数执行过程中位反转对发生交换，因此需要注意避免重新交换已经交换过的位反转对。这一保护措施已内置在函数中。

1.1 所需材料

对于这个程序，恩智浦的CodeWarrior 8.3 for DSC (CW 8.3)是可选的，只有当您希望从原始源代码派生工程时才需要。在www.nxp.com.cn注册的用户可免费获得该集成开发环境（IDE）的特别版。

使用CW 8.3 for DSC IDE的下列步骤不需要实际硬件。这是因为CW 8.3随附的模拟器支持DP56858。不需要调试盒，也不需要板卡。

只需要一台安装CW 8.3的PC。支持的操作系统包括Windows XP和Windows 7。

当然，需要使用CodeWarrior for MCU 10.5 (CW10.5)，可从www.nxp.com.cn获取。此外，运行完成的工程还需要TWR-56F8400产品。TWR-56F8400模块是一块小型电路板，可以在不使用塔式系统的情况下独立运行，也可以在塔式系统中运行。

它将托管已完成的移植应用程序。在撰写本文时，模拟器还不能用于MC56F84789（TWR-56F8400上使用的芯片）。

1.2 启动工程 – 最终工程

建议本工程从转换后的工程开始。它将在CodeWarrior for MCU 10.5版IDE下，在MC56F84789上运行原始的位反转代码（该代码也可在56800E芯片上运行）。本文提供的压缩文件包含一个工程，即最终工程。

实际上，我只提供带有新位反转功能的最终工程。这不是问题，因为要回到起点，只需从CW10.5的用户模块中删除dfr16bitrev.asm，然后使用Processor Expert重新生成代码。旧的效率较低的模块将恢复并安装在生成的代码目录中，但最终工程中缺少此模块，正如本应用笔记的相关软件压缩文件中所提供的那样。

本文的目的是说明最终工程是如何从起点衍生出来的。最终工程使用新的位反转功能节省了周期，还实时测量了自身的周期计数。

借助它，读者能够使用这款32位DSC的高级指令来处理减少周期计数的工程。首先，获取所有必需的材料，并检查位反转功能，包括新旧功能。

要获得最终解决方案，只需再次解压并导入文件即可。

实际上，如果重命名其中一个导入的工程，就可以在工作区中同时拥有这两个工程！

2 修改位反转汇编代码

第2.1节展示了在使用V2内核的传统DSC芯片上执行位反转的代码，并附有一些讨论。第2.2节展示了相同的代码，经过修改后，可利用全新32位DSC内核中的反向进位加法指令，并附有一些讨论。

修改相当简单。事实上，修改是对一个相当小的例程的简化。DSP代码通常都是小例程，因为每个数据点能提供的指令周期并不多。

然而，如何修改Processor Expert生成的汇编语言代码？其实很简单。从生成的代码文件夹中删除该模块，将其放在主程序所在的文件夹中。这样，IDE就会当作你拥有此模块。请注意不要让Processor Expert重新生成代码。不过不用担心，如果它确实重新生成了代码，您只需从生成的代码目录中删除该文件即可。

在本工程中，我们将使用或查看名为dfr16bitrev.asm的文件。

2.1 原始的传统V2位反转代码

```
;*****************************************************************
;
; (c) Freescale Semiconductor
```

修改位反转汇编代码

```
; 2004 All Rights Reserved
;
;*********************************************************************
;
; File Name:  dfr16bitrev.asm
;
; Description: Assembly module for Bit Reverse
;
; Modules
;  Included: Fdfr16Cbitrev_
;
; Author(s):  Sandeep S
;        Alwin Anbu.D
;
; Date:     3 Dec 2001
;
;*********************************************************************

    SECTION rtlib

  include "portasm.h"

  GLOBAL Fdfr16Cbitrev_

;*********************************************************************
;
; Module Name: Fdfr16Cbitrev_
;
; Description: Bit Reverses the Input Array
;
; Functions
;   Called: None
;
; Calling
; Requirements: 1. r2 - Pointer to Input Buffer.
;        2. r3 - Pointer to Output Buffer.
;        3. y0 - Length of the input/output buffer
;
; C Callable:  Yes
;
; Reentrant:  Yes
;
; Globals:   None
;
; Statics:   None
;
; Registers
;   Changed: All except r1 and r5
;
; DO loops:   1
;
; REP loops:  None
;
; Environment: MetroWerks on PC
;
; Special
```

```
;   Issues:  1.r2 and r3 MUST have even boundaries
;
;*****************************Change History************************
;
;  DD/MM/YY   Code Ver   Description   Author(s)
;  ----------   -----------   ---------------   --------
;  18/01/2001  0.1     Module created  Sandeep S
;  18/01/2001  1.0     Baselined    Sandeep S
;  30/11/2001  1.1     Modified     Alwin Anbu.D
;
;*********************************************************************

Fdfr16Cbitrev_

  adda   #2,sp
  move.l  c2,x:(sp)+
  move.l  c10,x:(sp)+
  move.l  d2,x:(sp)+
  move.l  d10,x:(sp)

  clr.w   d              ; d is the normal index
  move.w  #0,x0            ; x0 is the bit reversed index
  tfra   r2,r0            ; r0 points to input
  tfra   r3,r4            ; r4 points to output
  move.w  y0,c
  asr    c              ; a1=(No.of points)/2

  dec.w   y0              ; y0=n-1

  if CODEWARRIOR_WORKAROUND==1
  do    y0,>>End_Do
  else
  do    y0,End_Do
  endif

  move.w  x:(r0)+,a0          ; Move real part to a0
  move.w  x:(r0)+,a1          ; Move imaginary part to a1
  cmp.w   x0,d             ; Check if d < x0
  blt    elsepart           ; if yes,bit reversal
                     ; already done.Jump to elsepart
  moveu.wd,n               ; Move bit reversed index to n
  asla   n
  move.w  x:(r2+n),b0      ; Move real parts at bit
                    ; reversed locations to
                    ; normal location
  adda   #1,n
  move.w  x:(r2+n),b1         ; Move imaginary parts at bit
                    ; reversed locations to
                    ; normal location
  move.w  a1,x:(r4+n)
  adda   #-1,n
  move.w  a0,x:(r4+n)

  move.w  b0,x:(r3)+         ; Move imaginary part
  move.w  b1,x:(r3)-         ; Move imaginary part at bit
                    ; reversed location to
                     ; normal location
```

```
elsepart

  move.w  c1,y0            ; y0=N/2
  cmp     y0,d            ; Check if d < N/2 .Update r3
  move.w  x:(r3)+,y1
  blt     skip_change        ; If yes,skip change

chk_again            ; this loop is quite costly and is not needed with the V3 instruction set.
  sub     y0,d            ; d=d-y0
  asr     y0             ; y0=y0/2
  cmp.w   y0,d            ; Check if d>=y0
  bge     chk_again         ; If yes,check again

skip_change

  add     y0,d            ; Update r3
  move.w  x:(r3)+,y1
  inc.w   x0             ; Increment normal index
End_Do

  move.w  x:(r0)+,a0          ; Move last pair to a
  move.w  x:(r0)-,a1
  move.w  a0,x:(r3)+          ; Move a to last output location
  move.w  a1,x:(r3)-
  move.l  x:(sp)-,d
  move.l  x:(sp)-,d2
  move.l  x:(sp)-,c
  move.l  x:(sp)-,c2

  rts

  ENDSEC

;************************* End of file ********************************
```

2.2 为V3内核优化的位反转代码

上图中橙色（或浅黑色）的循环表示在新编码方案中可能被针对性削减的大部分指令。下面代码的循环中无需计算位反转，因为一条指令就能完成。参见下面蓝色的注释。

```
;*****************************************************************
;
; (c) Freescale Semiconductor
; 2013 All Rights Reserved
;
;
;*****************************************************************
;
; File Name:  dfr16bitrev.asm
;
; Description: Assembly module for Bit Reverse of Complex number vector
;        Ported from 858 PEx test projects using CW10.2 project converter
;
; Modules
;  Included: Fdfr16Cbitrev_
;
```

```
; Target Processor
;  HawkV3 family or greater (Nevis2, Anguilla Silver..)
;  Will fail to function on earlier DSC parts!
;
; IDE
;  CodeWarrior for MCU 10.4 with test harness
;
; Author(s):  John L. Winters
;
; Date:     01 Aug 2013
;
;*****************************************************************

    SECTION user

  include "portasm.h"

  GLOBAL Fdfr16Cbitrev_

;*****************************************************************
;
; Module Name: Fdfr16Cbitrev_
;
; Description: Bit Reverses the Input Array
;
; Functions
;   Called: None
;
; Calling
; Requirements: 1. r2 - Pointer to Input Buffer of complex 16 bit entries
;       2. r3 - Pointer to Output Buffer of complex 16 bit entries, may be same as input buffer
;        3. y0 - Length of the input/output buffer. Input and output buffers are same length.
;
; C Callable:  Yes
;
; Reentrant:  Yes
;
; Globals:   None
;
; Statics:   None
;
; Registers
;   Changed: All except r1 and r5
;
; DO loops:   1
;
; REP loops:  None
;
; Environment: MetroWerks on PC
;
; Special
;   Issues:  1. r2 and r3 MUST have even boundaries
;        2. core prior to V3 did not have the reverse carry add feature
;         so code must detect if V3 instructions are present prior using it.
;         if the core is not V3 or newer, the orginal code should assemble.
;         Even with the reverse carry add, it is still required to use the
;         bit reverse routine. It is just faster with V3 instructions.
```

```
;        3. This routine sets the M01 register to 0x4000 which affects how addressing with R1
functions.
;         Any ISR interrupting this routine must consider M01 treatment.
;         It is set back to all ones on return.
;
;*****************************Change History************************
;
;  DD/MM/YY    Code Ver    Description    Author(s)
;  ----------   ----------   ---------------   ---------
;  18/01/2001   0.1      Module created   Sandeep S
;  18/01/2001   1.0      Baselined     Sandeep S
;  30/11/2001   1.1      Modified      Alwin Anbu.D
;  08/01/2013   2.0      for V3 core    John L. Winters
;
;*********************************************************************
         SUBROUTINE "Fdfr16Cbitrev_",Fdfr16Cbitrev_,Fdfr16Cbitrev_END-Fdfr16Cbitrev_
Fdfr16Cbitrev_:               ; tag with colon needed for debugger information generation, as
well as above SUBROUTINE statement
  adda    #2,sp             ; step past call information in stack
  move.l  R1,x:(sp)+          ; push r1
  move.l  c2,x:(sp)+          ; puch c2
  move.l  c10,x:(sp)+          ; push c10
  move.l  d2,x:(sp)+          ; push d2
  move.l  d10,x:(sp)          ; push d10
  move.w  #$4000,x0  ; this value, when placed in M01, will activate R0 for bit reversal mode
  moveu.w  x0,m01             ; set the m01 register to reverse carry for R0 only addressing
  move.w  #0,r0    ; for caluclation of the bit reversed index, zero starting position
  clr.w   d         ; d1 is the bit reversed index, zero initially. Will be copied from r0.
  move.w  #0,y1              ; y1 is the straight index, also zero initially
  tfra    r2,r1             ; r2 and r1 point to input
  tfra    r3,r4             ; r3 and r4 point to output
  move.w  y0,c              ; C1 number of complex input numbers
  asr     c               ; C1= number of complex numbers/2
; which is the addend into the reverse carry function to generate the bit reversed counter
  dec.w   y0               ; y0=n-1
  if CODEWARRIOR_WORKAROUND==1      ; see definition of this tag for explanation
  do     y0,>>End_Do          ;This loop is executed once for each of the complex numbers in the
input
;(or in the output).
  else
  do     y0,End_Do
  endif
  move.w  x:(r1)+,a0       ; Move real part of the input complex number to a0
  move.w  x:(r1)+,a1          ; Move imaginary part of the input complex number to a1
  cmp.w   y1,d             ; Check if d < y1, (check if the index is less than its bit reverse
that is to say)
  blt     elsepart           ; if yes,bit reversal already done, so jump to elsepart.
  moveu.wd,n               ; Move bit reversed index to n
  asla    n               ; n is multiplied by two since complex numbers have two word entries
  move.w  x:(r2+n),b0          ; Move real part at bit reversed locations to: b0
  adda    #1,n              ; address the imaginary part at the bit reversed location
  move.w  x:(r2+n),b1          ; Move imaginary part at bit reversed location to b1
  move.w  a1,x:(r4+n)          ; store imaginary part of the input complex number in bit reversed
location (n always positive)
  adda    #-1,n             ; toggle down to the real part of the bit reversed loaction
  move.w  a0,x:(r4+n)          ; store the real part of the input complex number in the bit
reversed location (n always positive)
```

```
  move.w  b0,x:(r3)+          ; Move real part and
  move.w  b1,x:(r3)-          ; imaginary part at bit reversed location to normal location

; Above section moves the numbers to effect the bit reversal
elsepart   ; bit reversal already done (used bit reverse indices are always greater than the
index to avoid re-reversing!)
; this is where the next index and next bit reversed index are generated
; this code below adds one to the bit reversed index using the reverse carry feature of the V3
Hawk core.
; increment the bit reversed counter and the straight counter
  moveu.w  c,n              ; this is the number of complex numbers divided by two,
; or the msb of the complex number index needed to bit-reverse count-up
  move.w  x:(r3)+,x0          ; update r3; x0 is don't care.
  move.w  x:(r0)+n,x0 ; increment reverse carry index
  move.w  x:(r3)+,x0           ; update r3; x0 is don't care (can be done anywhere)
  inc.w   y1                ; Increment normal index
  move.w  r0,d1              ; copy reverse carry to d
End_Do                     ; end of zero overhead loop. Last item is moved below
  move.w  #-1,x0             ; all ones is reset configuration for M01
  moveu.w  x0,m01            ; set the m register to linear addressing
  move.w  x:(r1)+,a0          ; Move last pair to a0,a1. (all ones index)
  move.w  x:(r1)-,a1          ; Im part
  move.w  a0,x:(r3)+          ; Move a to last output location (all ones index)
  move.w  a1,x:(r3)-          ; Im part
  move.l  x:(sp)-,d          ; pop d10
  move.l  x:(sp)-,d2         ; pop d2
  move.l  x:(sp)-,c          ; pop c10
  move.l  x:(sp)-,c2          ; pop c2
  move.l  x:(sp)-,r1         ; pop r1
  rts                   ; return from subroutine, stack restored
        Fdfr16Cbitrev_END:
  ENDSEC

;************************ End of file ********************************a
```

3 结论

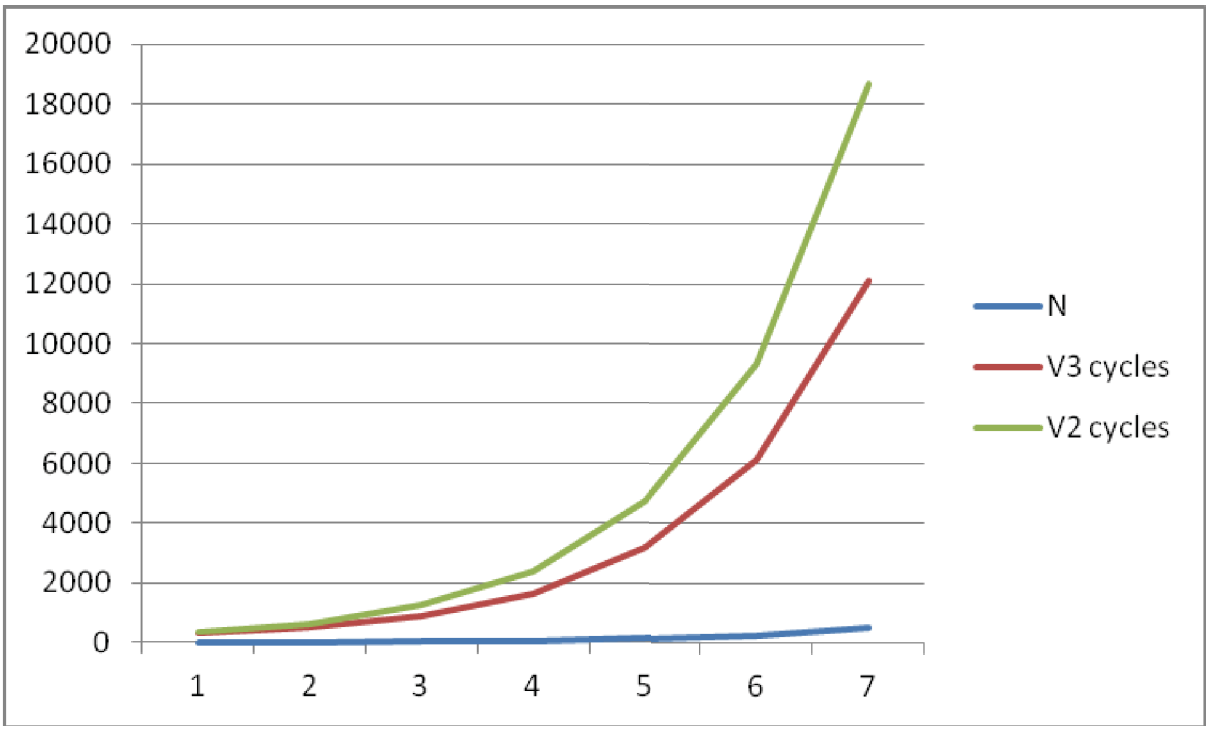
使用本应用笔记中介绍的方法，可以从早期DSC产品（从第一个V2 DSC DSP56858开始）直接移植包括内核/内存例程（不使用外设）的DSC工程。这包括整个基于V2内核的全系列DSC产品。

在Processor Expert自动生成的源代码中，可直接使用MC56F84789的新内核功能。这可以提高性能：除了利用较新芯片的更快时钟速度外，所需的机器周期也更少。

DSP56858用户可用的大量示例代码现在可供所有DSC用户随时进行移植与改进。这些代码包括源代码生成和所含的测试工具。

本文公开了修改生成的汇编语言例程的方法。在执行所有测试过的CFFT用例时，新的位反转功能可以节省大量周期。

使用这一新指令应用后，性能提高了多少?



如上所示，节省的周期数随CFFT的大小而增加。以表格形式显示如下。

表1. 周期比较

N	V3周期	V2周期	节省的周期百分比
8	315	371	15.09433962
16	499	651	23.34869432
32	899	1251	28.13749001
64	1635	2395	31.73277662
128	3171	4755	33.31230284
256	6115	9355	34.63388562
512	12131	18691	35.09710556

在所测试的用例中，通过测量可得采用此技术可以节省高达35%的位反转所需周期。

4 测试与验证

本应用笔记中考虑的Processor Expert示例工程包括在DSP56858上运行的测试工具。一旦完成移植，运行该测试工具会自动验证向新目标的移植。周期时间测量利用板载定时器进行精确周期测量，包括任何总线或内核停滞，以获得真实数字。

如果在位反转功能中发现任何缺陷，测试就会失败。在调试代码的过程中，我曾多次遇到这种情况。

本应用笔记提供的最终工程包括一些软件组件，它们利用片上系统MC56F84789上的定时器测量周期计数。读者可以基于最终工程，CodeWarrior 10.5和包含MC56F84789片上系统的TWR-MC56F8400模块重复所得的结果。

How to Reach Us:

Home Page:
nxp.com.cn

Web Support:
nxp.com.cn/support

Document Number: AN4945
Rev. 0
4/2014