用于 dsPIC®数字信号控制器的 DSP 库用户指南

简介

本文档概述了适用于 dsPIC 器件的 DSP 库，该库提供了一套预先优化的软件程序，旨在简化数字信号处理任务。该库可高效实现复杂数学函数和算法，助力开发人员提升 dsPIC DSC 器件在高级检测和控制、电源转换、音频处理、电信以及电机控制等应用中的性能与功能。DSP 库目前共支持 52 个函数。

目录

1. 适用于 dsPIC 数字信号控制器的 DSP 库

1.1. C 代码应用程序

适用于 dsPIC 器件的 DSP 库随 XC-DSC 编译器一起提供。该库及其相关文件位于编译器安装目录的以下路径中：

- *lib\libdsp-elf.a*——DSP 库/归档文件
- *src*——库函数的压缩源代码（*libdsp.zip*），其中包含用于重新编译库的批处理文件
- *support\generic\h*——DSP 库的头文件

1.2. 使用 DSP 库

在特定用户场景或传统应用中使用 DSP 库之前，请务必阅读本文档的以下章节

- 使用 DSP 库进行编译
- 存储器模型
- DSP 库函数调用约定
- 数据类型
- 数据存储器的使用
- CORCON 寄存器的使用
- 溢出和饱和处理
- 与中断和 RTOS 集成
- 重新编译 DSP 库
- DSP 库函数

1.2.1. 使用 DSP 库进行编译

使用 DSP 库编译应用程序只需两个文件：***dsp.h*** 和 ***libdsp-elf.a***。*dsp.h* 为头文件，提供该库使用的所有函数原型、*#define* 和 *typedef*。*libdsp-elf.a* 为归档库文件，其中包含所有库函数的全部目标文件（有关 ELF 专用库的更多信息，请参见 MPLAB XC-DSC Libraries Reference Manual（DS50003591））。

编译应用程序时，所有调用 DSP 库中的函数或者使用其符号或 typedef 的源文件都必须引用 ***dsp.h***（使用 #include）。链接应用程序时，必须提供 ***libdsp-elf.a*** 作为链接器的输入（使用***--library dsp*** 或***-ldsp*** 链接器开关），以便将应用程序使用的函数链接到项目中。

链接器会将 DSP 库的函数放入名为*.libdsp* 的特殊文本段中。这可以在链接器生成的映射文件中进行查看。

1.2.2. 存储器模型

DSP 库采用“小代码”和“小数据”存储器模型进行编译，以最大限度地减小库占用空间。DSP 库中的部分函数使用 C 语言编写，这些函数利用编译器的浮点库。因此，编译器链接描述文件将.libm 和.libdsp 文本段放置在相邻位置。这样可确保 DSP 库安全地调用 RCALL 指令，以从浮点库中调用必需的浮点程序。

1.2.3. DSP 库函数调用约定

DSP 库内的所有目标模块均符合适用于 dsPIC DSC 器件的 C 语言兼容性准则。该准则遵循 XC-DSC C Compiler User's Guide（DS50003589）的“Function call convention”部分提供的函数调用约定。

1.2.4. 数据类型

DSP 库提供的运算充分利用了 dsPIC30F/33F/33E/33C/33A 器件的 DSP 指令集和架构特性。因此，大多数运算均采用小数算术。

对于 dsPIC30F、dsPIC33F、dsPIC33C 和 dsPIC33E 系列器件，所有小数输入和输出参数均采用 1.15 定点数据类型格式。而对于 dsPIC33A 系列，则采用 1.31 定点数据类型格式。

DSP 库通过整数类型定义小数类型：

```
#ifndef fractional
     typedef int fractional;
#endif
```

对于 dsPIC30F/33F/33E/33C，小数数据类型用于表示含 1 个符号位和 15 个小数位的数据。采用该格式的数据通常称为 1.15 定点数据。对于使用乘法器的函数，通过 40 位累加器和“9.31”算术计算结果。该数据格式包含 9 个符号/幅值位和 31 个小数位，支持超出 1.15 格式范围（-1.00 至+1.00 左右）的扩展计算。这些函数以 1.15 格式的小数数据类型提供结果。

类似地，dsPIC33A 器件的小数数据采用 1 个符号位和 31 个小数位表示，通常称为 1.31 定点格式。对于使用乘法器的函数，通过 72 位累加器和“9.63”算术计算结果。该格式包含 9 个符号/幅值位和 63 个小数位，支持超出 1.31 格式范围（-1.00 至+1.00 左右）的扩展计算。当这些函数返回结果时，会先将其恢复为 1.31 格式的小数数据类型。

得益于可用小数位数的增加，1.31 格式相较于 1.15 格式具有更高的计算精度。

使用小数算术时，允许输入到特定函数的数值集会受到一些限制。但是，一些函数会对输入数据和/或输出结果执行隐式缩放，这可能会降低输出值的分辨率（与浮点实现相比）。

DSP 库中的部分运算需要更高的数值分辨率，其采用浮点算术。但是，这些运算的结果需转换成小数值，以便与应用程序集成。惟一的例外是 *MatrixInvert* 函数，该函数采用浮点算术对浮点矩阵求逆并以浮点格式提供结果。

1.2.5. 数据存储器的使用

DSP 库不执行 RAM 分配，而是将该任务留给用户处理。如果用户没有正确分配和对齐数据存储器，函数执行过程中就会出现意外结果。此外，为了最大限度地缩短执行时间，DSP 库不会检查所提供的函数参数（包括指向数据存储器的指针）是否有效。

大多数函数接受数据指针作为函数参数，其中包含要运算的数据，通常还包含存储结果的位置。为了方便起见，DSP 库中的大多数函数都希望将其输入参数分配到默认 RAM 存储空间（X 数据空间或 Y 数据空间），并将输出存储回默认 RAM 存储空间。但是，计算密集型函数要求一些操作数驻留在 X 数据空间和 Y 数据空间（或程序存储器和 Y 数据空间），以便运算可以充分利用 dsPIC 架构的双数据获取能力。

注：dsPIC33A 架构不强制特定操作数位于 X 数据空间或 Y 数据空间。但是，对于少数计算密集型函数，将特定操作数放在同一个数据空间（无论是 X 数据空间还是 Y 数据空间）会导致按顺序执行操作数访问，进而影响数据获取效率。

1.2.6. CORCON 寄存器的使用

DSP 库的许多函数通过修改 CORCON 寄存器将器件置于特殊工作模式。在进入这些函数时，CORCON 寄存器的内容会压入堆栈，然后经过修改以正确执行所需的运算，最后从堆栈中弹出以保留其原始值。该机制允许在不破坏 CORCON 设置的情况下执行库。

CORCON 寄存器配置为将目标器件置于以下工作模式。

- DSP 乘法器配置为利用有符号小数数据。
- 累加器 A 和累加器 B 均使能累加器饱和。
- 饱和模式设置为 9.31/9.63 饱和（超饱和）。
- 使能数据空间写饱和。
- 如果适用，禁止程序空间可视性。

- 使能收敛（无偏）舍入。

有关 CORCON 寄存器及其作用的详细说明，请参见相应的器件数据手册。

1.2.7. 溢出和饱和处理

DSP 库使用 9.31/9.63 饱和模式执行大多数计算，但必须分别以 1.15 或 1.31 格式存储函数的输出。如果使用的累加器在运算过程中饱和，则状态寄存器中的相应饱和位（SA 或 SB）将置 1。该位将保持置 1 状态，直到被清零。因此，可以在函数执行后检查 SA 或 SB 以确定是否应采取措施来缩放函数的输入数据。

同理，如果使用累加器执行的计算导致溢出，则状态寄存器中的相应溢出位（OA 或 OB）也将置 1。与 SA 和 SB 状态位不同的是，OA 和 OB 不会保持置 1 状态等待被清零。每次使用累加器执行运算时都会更新这些位。如果超出该指定范围是某个重要事件的标志，建议通过 INTCON1 寄存器（在 dsPIC30F/33F/33E/33C 中）或 INTCON4 寄存器（在 dsPIC33A 中）中的 OVATE、OVBTE 和 COVTE 位来使能累加器溢出陷阱。这样，一旦发生溢出情况，就会生成算术数学错误陷阱，然后采取所需的操作。

1.2.8. 与中断和 RTOS 集成

DSP 库可以轻松集成到利用中断或遵循特定准则的 RTOS 的应用程序中。为了最大限度地缩短执行时间，DSP 库使用了 DO 循环（若已实现）、REPEAT 循环、模寻址和位反转寻址。这些组件都是 dsPIC 器件上的有限硬件资源，后台代码若需要中断执行 DSP 库函数，必须考虑这些资源的使用情况。

与 DSP 库集成时，务必检查每个函数说明的函数配置文件，以确定使用了哪些资源。如果要中断库函数，用户需负责保存和恢复该函数使用的所有寄存器的内容，其中包括 DO（若已实现）、REPEAT 和特殊寻址硬件的状态。此外，用户还需负责保存和恢复 CORCON 寄存器与状态寄存器的内容。

1.2.9. 重新编译 DSP 库

要编译 DSP 库，需解压*(编译器安装路径)\src\libdsp.zip* 路径中的源文件；这将创建一个名为 *libdsp* 的子文件夹。执行该文件夹中的 *makedsplib.bat* 文件可重新编译库归档。这将创建两个文件夹，分别名为“*lib*”和“*obj*”。当批处理文件执行完毕后，重新编译的 *libdsp.a* 文件位于“*lib*”文件夹中。

重新编译 DSP 库需要使用 MPLAB XC-DSC 编译器，并且必须将其安装路径添加到 Windows®或 Linux®环境变量中。

1.2.10. DSP 库函数

DSP 库支持下列函数模块：

- 向量函数
- 窗函数
- 矩阵函数
- 滤波函数
- 变换函数
- 控制函数
- 转换函数
- 堆栈函数

2. 向量函数

函数	说明
VectorAdd	将源一向量中每个元素的值与源二向量中对应元素的值相加，并将结果放入目标向量中。
VectorConvolve	计算两个源向量之间的卷积并将结果存入目标向量中。
VectorCopy	将源向量的元素复制到（已存在的）目标向量的开头。
VectorCorrelate	计算两个源向量之间的相关性并将结果存入目标向量中。
VectorDotProduct	计算源一向量与源二向量中对应元素的乘积之和。
VectorMax	查找源向量中值大于或等于所有前序元素的最后一个元素，然后返回该最大值和最大元素的索引。
VectorMin	查找源向量中值小于或等于所有前序元素的最后一个元素，然后返回该最小值和最小元素的索引。
VectorMultiply	将源一向量中每个元素的值与源二向量中对应元素的值相乘，并将结果放入目标向量的相应元素中。
VectorNegate	对源向量中元素的值取反（改变符号）并将其放入目标向量中。
VectorPower	计算源向量的幂，即其元素的平方和。
VectorScale	将源向量中所有元素的值按一定比例缩放（乘以一定的缩放值）并将结果放入目标向量中。
VectorSubtract	从源一向量中每个元素的值中减去源二向量中对应元素的值，并将结果放入目标向量中。
VectorZeroPad	将源向量复制到（已存在的）目标向量的开头，然后用零填充目标向量的其余元素。

2.1. 小数向量运算

小数向量由一系列在存储器中连续分配的数值元素构成，第一个元素位于最低存储器地址处。每个元素占用一个字的存储空间，其数值必须解释为以 *1.15/1.31 数据格式*表示的小数。

指向向量第一个元素的指针作为句柄，用于授权访问向量内的每个值。第一个元素的地址称为向量的基址（Base Address，BA）。假设每个向量元素为 16 位（对于 1.15 小数数据）或 32 位（对于 1.31 小数数据），则基址的最后 1 位或 2 位*必须*为零。

向量的一维排列适配器件的存储器模型，因此 N 元素向量中第 n 个元素的地址可通过向量的基址进行偏移计算：

BA + (sizeof(fractional))*(n – 1)，其中 1 ≤ n ≤ N。

该库实现了一元和二元小数向量运算。一元运算中的操作数向量称为源向量。在二元运算中，第一个操作数向量称为源一向量，第二个操作数向量称为源二向量。每个运算都会对源向量的一个或多个元素进行特定计算。部分运算生成的结果是标量值（可解释为 1.15/1.31 小数），而其他运算生成的结果是向量。当结果为向量时，称为目标向量。

部分生成向量的运算允许就地计算。这意味着，运算结果会回写到源向量中（对于二元运算则为源一向量）。在这种情况下，目标向量将（物理上）取代源（一）向量。如果某个运算可就地计算，则会在函数说明提供的备注中作此标示。

对于某些二元运算，两个操作数可以是同一（物理）源向量，这意味着该运算将应用于源向量与其自身。如果对于给定的运算可以进行这种类型的计算，则会在函数说明提供的备注中将其标示为“可自应用”。

一些运算既可自应用，也可就地计算。

该库中的所有小数向量运算都将操作数向量的基数（元素个数）作为参数。根据该参数的值，将做出以下假设：

1. 特定运算中涉及的所有向量的大小总和在目标器件可用数据存储器的容量范围内。
2. 对于二元运算，两个操作数向量的基数*必须*遵守向量代数规则（特别是 *VectorConvolve* 和 *VectorCorrelate* 函数）。

3. 目标向量*必须*足够大以容纳运算的结果。

2.2. 补充说明

函数说明通常会限制这些运算的常规使用范围。但是，由于在计算这些函数时不执行边界检查，因此会根据特定需求解释运算及其结果。

例如，在计算 *VectorMax* 函数时，源向量的长度可能大于 *numElems*。在这种情况下，该函数将仅用于在源向量的前 *numElems* 个元素中查找最大值。

同理，如果需要将目标向量中位于 N 与 N+*numElems*-1 之间的 *numElems* 个元素替换为源向量中位于 M 与 M+*numElems*-1 之间的 *numElems* 个元素，则可以按照如下形式使用 *VectorCopy* 函数：

```
fractional* dstV[DST_ELEMS] = {...};
fractional* srcV[SRC_ELEMS] = {...};

int n = NUM_ELEMS;
int N = N_PLACE;    /* NUM_ELEMS+N ≤ DST_ELEMS */
int M = M_PLACE;    /* NUM_ELEMS+M ≤ SRC_ELEMS */

fractional* dstVector = dstV+N;
fractional* srcVector = srcV+M;

dstVector = VectorCopy (n, dstVector, srcVector);
```

另外，在该上下文中，*VectorZeroPad* 函数可就地运行，其中 *dstV = srcV*，*numElems* 是源向量开头要保持不变的元素个数，*numZeros* 是向量尾部要设置为零的元素个数。鉴于不会执行边界检查，因此可以开发利用其他可能性。

2.3. 函数

2.3.1. VectorAdd

说明

`VectorAdd` 用于将源一向量中每个元素的值与源二向量中对应元素的值相加，并将结果放入目标向量中。

原型

`fractional* VectorAdd (int numElems, fractional* dstV, fractional* srcV1, fractional* srcV2);`

参数

参数	说明
numElems	源向量中的元素个数
dstV	指向目标向量的指针
srcV1	指向源一向量的指针
srcV2	指向源二向量的指针

返回

指向目标向量的基址的指针。

备注

如果 *srcV1[n] + srcV2[n]*的绝对值大于 1.15/1.31 小数数据类型的最大值，则该运算将导致第 n 个元素饱和。

该函数可就地计算。

该函数可自应用。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `vadd.s`
- 对于 dsPIC33A
  `vadd_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	13	17 + 3*(*numElems*)
dsPIC33E/33C	16	25 + 3*(*numElems*)
dsPIC33A	11	36 + 4*(*numElems*)

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W4*——已使用，未恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 1 条 *DO* 指令
    - 无 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W4*——已使用，未恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——无

2.3.2. VectorConvolve

说明

`VectorConvolve` 用于计算两个源向量之间的卷积并将结果存入目标向量中。计算结果如下：

$$y[n] = \sum_{k=0}^{n} x[k] * h[n-k] \quad , for\ 0 \le n \le N + M - 1$$

其中，

x[k] = 源一向量，大小为 *N*，

h[k] = 源二向量，大小为 *M*（*M* < *N*）。

原型

```
fractional* VectorConvolve (int numElems1,int numElems2,fractional* dstV,
fractional* srcV1,fractional* srcV2);
```

参数

参数	说明
numElems1	源一向量中的元素个数
numElems2	源二向量中的元素个数
dstV	指向目标向量的指针
srcV1	指向源一向量的指针
srcV2	指向源二向量的指针

返回

指向目标向量的基址的指针。

备注

源二向量的元素个数*必须*小于或等于源一向量中的元素个数。

目标向量*必须*已存在，并且恰好包含 *numElems1* + *numElems2* - 1 个元素。

该函数可自应用。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `vcon.s`
- 对于 dsPIC33A
  `vcon_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	58	鉴于 N = *numElems1* 且 M = *numElems2*， $\mathbf{33} + 13M + 6\sum_{m=1}^{M} m + (N-M)(7+3M)$ *for* $M < N$ $\mathbf{33} + 13M + 6\sum_{m=1}^{M} m$ *for* $M = N$
dsPIC33E/33C	62	鉴于 N = *numElems1* 且 M = *numElems2*， $\mathbf{35} + 13M + 6\sum_{m=1}^{M} m + (N-M)(7+3M)$ *for* $M < N$ $\mathbf{35} + 13M + 6\sum_{m=1}^{M} m$ *for* $M = N$
dsPIC33A	37	鉴于 N = *numElems1* 且 M = *numElems2*， $\mathbf{32} + 17M + 2\sum_{m=1}^{M} m + (N-M)(7+M)$ *for* $M < N$ $\mathbf{32} + 17M + 2\sum_{m=1}^{M} m$ *for* $M = N$

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W7*——已使用，未恢复
  - *W8...W10*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 两级 *DO* 指令

    - 无 *REPEAT* 指令
- 对于 dsPIC33A
    - *W0...W7*——已使用，未恢复
    - *W8...W9*——已保存，已使用，已恢复
    - *ACCA*——已使用，未恢复
    - *CORCON*——已保存，已使用，已恢复
    - 两条 *REPEAT* 指令

2.3.3. VectorCopy

说明

`VectorCopy` 用于将源向量的元素复制到（已存在的）目标向量的开头，以使：

dstV[n] = srcV[n]，0 ≤ n < numElems

原型

```
fractional* VectorCopy (int numElems, fractional* dstV, fractional* srcV);
```

参数

参数	说明
numElems	源向量中的元素个数
dstV	指向目标向量的指针
srcV1	指向源一向量的指针

返回

指向目标向量的基址的指针。

备注

目标向量必须已存在。目标向量的大小必须大于或等于 *numElems*。

该函数可就地计算。有关该工作模式的备注，请参见本节开头的补充说明。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `vcopy.s`
- 对于 dsPIC33A
  `vcopy_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	6	12 + (*numElems*)
dsPIC33E/33C	9	20 + (*numElems*)
dsPIC33A	5	*见下表

dsPIC33A 的周期数	
源向量大小	**周期数**
32	68
64	122

VectorCopy （续）	
dsPIC33A 的周期数	
128	228
256	442
512	868
1024	1722
2048	3428

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C/33A
  - *W0...W3*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 无 *DO* 指令（对于 dsPIC33A *不适用*）
    - 一条 *REPEAT* 指令

2.3.4. VectorCorrelate

说明

`VectorCorrelate` 用于计算两个源向量之间的相关性并将结果存入目标向量中。计算结果如下：

$$y[n] = \sum_{k=0}^{n} x[k] * h[k-n] \quad , for\ 0 \le n \le N + M - 1$$

原型

```
fractional* VectorCorrelate (int numElems1, int numElems2, fractional* dstV,
fractional* srcV1, fractional* srcV2);
```

参数

参数	说明
numElems1	源一向量中的元素个数（*N*）
numElems2	源二向量中的元素个数（*M*，*M* ≤ *N*）
dstV	指向目标向量（大小为 *N+M-1*）的指针
srcV1	指向源一向量的指针
srcV2	指向源二向量的指针

返回

指向目标向量的基址的指针。

备注

源二向量的元素个数必须小于或等于源一向量中的元素个数。

目标向量必须已存在，并且恰好包含 *numElems1 + numElems2 - 1* 个元素。

该函数可自应用。

该函数使用 *VectorConvolve*。

源文件

- 对于 dsPIC30F/33F/33E/33C

vcor.s

- 对于 dsPIC33A
  vcor_aa.s

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	14	$19 + \lfloor M/2 \rfloor \times 3$
dsPIC33E/33C	17	$25 + \lfloor M/2 \rfloor \times 3$
dsPIC33A	12	$28 + \lfloor M/2 \rfloor \times 3$

注：

1. 上述程序字数和周期数仅与 *VectorCorrelate* 有关。但是，由于该函数本身利用 *VectorConvolve*，因此还必须考虑 *VectorConvolve* 的程序字数和周期数。
2. 在 *VectorConvolve* 的说明中，报告的周期数包括四个周期的 C 函数调用开销。因此，从 *VectorConvolve* 加到 *VectorCorrelate* 的实际周期数比单独 *VectorConvolve* 报告的周期数少四个。

系统资源的使用

VectorConvolve 函数不使用以下系统资源。

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W7*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 一级 *DO* 指令
    - 无 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W7*——已使用，未恢复
  - *REPEAT* 指令的使用——无

2.3.5. VectorDotProduct

说明

VectorDotProduct 用于计算源一向量与源二向量中对应元素的乘积之和。

原型

```
fractional VectorDotProduct (int numElems, fractional* srcV1,
fractional* srcV2);
```

参数

参数	说明
numElems	源向量中的元素个数
srcV1	指向源一向量的指针
srcV2	指向源二向量的指针

返回

乘积之和的值。

备注

如果 *sum-of-products* 的绝对值大于 1.15/1.31 小数数据类型的最大值，则该运算将导致饱和。

该函数可就地计算。

该函数可自应用。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `vdot.s`
- 对于 dsPIC33A
  `vdot_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	13	17 + 3*(*numElems*)
dsPIC33E/33C	16	25 + 3*(*numElems*)
dsPIC33A	11	32 + *numElems*（如果 *srcV2* 位于 y 存储区中且 *srcV1* 位于 x 存储区中） 32 + 2**numElems*（如果 *srcV2* 和 *srcV1* 均位于 x 存储区中）

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W5*——已使用，未恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 一级 *DO* 指令
    - 无 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W4*——已使用，未恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——1 条

2.3.6. VectorMax

说明

`VectorMax` 用于返回源向量中值大于或等于所有前序元素的最后一个元素的值和索引。

原型

```
fractional VectorMax (int numElems, fractional* srcV, int* maxIndex);
```

参数

参数	说明
numElems	源向量中的元素个数
srcV	指向源向量的指针
maxIndex	指向（最后一个）最大元素的索引存储位置的指针

返回

向量中的最大值。

备注

如果 *srcV*[i] = *srcV*[j] = *maxVal* 且 i < j，则*maxIndex = j。

源文件

- 对于 dsPIC30F/33F/33E/33C
  vmax.s
- 对于 dsPIC33A
  vmax_aa.s

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	13	14（如果 *numElems* = 1） 20 + 8(*numElems* - 2)（如果 srcV[n] ≤ srcV[n + 1]，0 ≤ n < *numElems* - 1） 19 + 7(*numElems* - 2)（如果 srcV[n] > srcV[n + 1]，0 ≤ n < *numElems* - 1）
dsPIC33E/33C	16	22（如果 *numElems* = 1） 28 + 10(*numElems* - 2)（如果 srcV[n] ≤ srcV[n + 1]，0 ≤ n < *numElems* - 1） 27 + 11(*numElems* - 2)（如果 srcV[n] > srcV[n + 1]，0 ≤ n < *numElems* - 1）
dsPIC33A	10	32 + 4*(*numElems* - 1)

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W5*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 无 *DO* 或 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W6*——已使用，未恢复
  - *REPEAT* 指令的使用——无

2.3.7. VectorMin

说明

VectorMin 用于返回源向量中值小于或等于所有前序元素的最后一个元素的值和索引。

原型

```
fractional VectorMin (int numElems, fractional* srcV, int* minIndex);
```

参数

参数	说明
numElems	源向量中的元素个数
srcV	指向源向量的指针
minIndex	指向（最后一个）最小元素的索引存储位置的指针

返回

向量中的最小值。

备注

如果 *srcV*[i] = *srcV*[j] = *minVal* 且 i < j，则*minIndex = j。

源文件

- 对于 dsPIC30F/33F/33E/33C
  vmin.s
- 对于 dsPIC33A
  vmin_aa.s

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	13	14（如果 *numElems* = 1） 20 + 8(*numElems* - 2)（如果 srcV[n] ≤ srcV[n + 1]，0 ≤ n < *numElems* - 1） 19 + 7(*numElems* - 2)（如果 srcV[n] > srcV[n + 1]，0 ≤ n < *numElems* - 1）
dsPIC33E/33C	16	22（如果 *numElems* = 1） 28 + 10(*numElems* - 2)（如果 srcV[n] ≤ srcV[n + 1]，0 ≤ n < *numElems* - 1） 27 + 11(*numElems* - 2)（如果 srcV[n] > srcV[n + 1]，0 ≤ n < *numElems* - 1）
dsPIC33A	10	32 + 4*(*numElems* - 1)

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W5*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 无 *DO* 或 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W6*——已使用，未恢复
  - *REPEAT* 指令的使用——无

2.3.8. VectorMultiply

说明

VectorMultiply 用于将源一向量中每个元素的值与源二向量中对应元素的值相乘，并将结果放入目标向量中。

原型

```
fractional* VectorMultiply (int numElems, fractional* dstV,
fractional* srcV1, fractional* srcV2);
```

参数

参数	说明
numElems	源向量中的元素个数
dstV	指向目标向量的指针
srcV1	指向源一向量的指针
srcV2	指向源二向量的指针

返回

指向目标向量的基址的指针。

备注

该运算也称为向量逐元素乘法。

该函数可就地计算。

该函数可自应用。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `vmul.s`
- 对于 dsPIC33A
  `vmul_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	14	17 + 4*(*numElems*)
dsPIC33E/33C	17	25 + 4*(*numElems*)
dsPIC33A	20	36 + ⌈2.5 x *numElems*⌉

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W5*——已使用，未恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 1 条 *DO* 指令
    - 无 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W4*——已使用，未恢复
  - *W13*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——无

2.3.9. VectorNegate

说明

`VectorNegate` 用于对源向量中元素的值取反（改变符号）并将其放入目标向量中。

原型

```
fractional* VectorNegate (int numElems, fractional* dstV, fractional* srcV1);
```

参数

参数	说明
numElems	源向量中的元素个数
dstV	指向目标向量的指针
srcV1	指向源向量的指针

返回

指向目标向量的基址的指针。

备注

0x80..0 的取反值设置为 0x7F..F。

该函数可就地计算。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `vneg.s`
- 对于 dsPIC33A
  `vneg_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	16	19 + 4*(*numElems*)
dsPIC33E/33C	19	27 + 4*(*numElems*)
dsPIC33A	10	32 + (*numElems*)

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W5*——已使用，未恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 一级 *DO* 指令
    - 无 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W3*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——无

2.3.10. VectorPower

说明

`VectorPower` 用于计算源向量的幂，即其元素的平方和。

原型

```
fractional VectorPower(int numElems, fractional* srcV);
```

参数

参数	说明
numElems	源向量中的元素个数
srcV	指向源向量的指针

返回

向量幂值（平方和）。

备注

如果 *sum-of-squares* 的绝对值大于 1.15/1.31 小数数据类型的最大值，则该运算将导致饱和。

该函数可自应用。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `vpow.s`
- 对于 dsPIC33A
  `vpow_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	12	16 + 2*(*numElems*)
dsPIC33E/33C	15	24 + 2*(*numElems*)
dsPIC33A	9	36 + (*numElems*)

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W4*——已使用，未恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 无 *DO* 指令
    - 一条 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W4*——已使用，未恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - 一条 *REPEAT* 指令

2.3.11. VectorScale

说明

`VectorScale` 用于将源向量中所有元素的值乘以一定的缩放值并将结果放入目标向量中。

原型

```
fractional* VectorScale (int numElems, fractional* dstV, fractional* srcV,
fractional sclVal);
```

参数

参数	说明
numElems	源向量中的元素个数
dstV	指向目标向量的指针
srcV	指向源向量的指针
sclVal	用于缩放向量元素的比例值

返回

指向目标向量的基址的指针。

备注

sclVal 必须是 1.15/1.31 格式的小数。

该函数可就地计算。

该函数可自应用。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `vscl.s`
- 对于 dsPIC33A
  `vscl_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	14	17 + 4*(*numElems*)
dsPIC33E/33C	17	25 + 4*(*numElems*)
dsPIC33A	20	40 + ⌈2.5 x *numElems*⌉

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W4*——已使用，未恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 1 级 *DO* 指令
    - 无 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W4*——已使用，未恢复
  - *W13*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——无

2.3.12. VectorSubtract

说明

VectorSubtract 用于从源一向量中每个元素的值中减去源二向量中对应元素的值，并将结果放入目标向量中。

原型

```
fractional* VectorSubtract (int numElems, fractional* dstV,
fractional* srcV1, fractional* srcV2);
```

参数

参数	说明
numElems	源向量中的元素个数
dstV	指向目标向量的指针
srcV1	指向源一向量的指针
srcV2	指向源二向量的指针

返回

指向目标向量的基址的指针。

备注

如果 *srcV1[n] - srcV2[n]*的绝对值大于 1.15/1.31 小数数据类型的最大值，则该运算将导致第 n 个元素饱和。

该函数可就地计算。

该函数可自应用。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `vsub.s`
- 对于 dsPIC33A
  `vsub_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	13	17 + 3*(*numElems*)
dsPIC33E/33C	16	25 + 3*(*numElems*)
dsPIC33A	11	36 + 4*(*numElems*)

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W4*——已使用，未恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 一条 *DO* 指令
    - 无 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W4*——已使用，未恢复
  - *ACCA*——已使用，未恢复

- *CORCON*——已保存，已使用，已恢复
- *REPEAT* 指令的使用——无

2.3.13. VectorZeroPad

说明

`VectorZeroPad` 用于将源向量复制到（已存在的）目标向量的开头，然后用零填充目标向量的其余 *numZeros* 个元素。

dstV[n] = srcV[n]，0 ≤ n < numElems

dstV[n] = 0，numElems ≤ n < numElems + numZeros

原型

```
fractional* VectorZeroPad (int numElems, int numZeros, fractional* dstV,
fractional* srcV1);
```

参数

参数	说明
numElems	源向量中的元素个数
numZeros	零的数量
dstV	指向目标向量（大小为 *numElems* + *numZeros*）的指针
*srcV*1	指向源向量的指针

返回

指向目标向量的基址的指针。

备注

目标向量*必须*已存在，并且恰好包含 *numElems* + *numZeros* 个元素。

该函数可就地计算。

该函数使用 *VectorCopy*

源文件

- 对于 dsPIC30F/33F/33E/33C
  `vzpad.s`
- 对于 dsPIC33A
  `vzpad_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	13	18 + (*numZeros*)
dsPIC33E/33C	16	26 + (*numZeros*)
dsPIC33A	9	12 + (*numZeros*)

注：

1. 上述程序字数和周期数仅与 *VectorZeroPad* 有关。但是，由于该函数本身利用 *VectorCopy*，因此还必须考虑 *VectorCopy* 的程序字数和周期数。
2. 在 *VectorCopy* 的说明中，报告的周期数包括四个周期的 C 函数调用开销。因此，从 *VectorCopy* 加到 *VectorZeroPad* 的实际周期数比单独 *VectorCopy* 报告的周期数少四个。

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W6*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 无 *DO* 指令
    - 一条 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W5*——已使用，未恢复
  - 一条 *REPEAT* 指令

3. 窗函数

函数	说明
BartlettInit	初始化一个长度为 *numElems* 的 Bartlett 窗。
BlackmanInit	初始化一个长度为 *numElems* 的 Blackman（三项）窗。
HammingInit	初始化一个长度为 *numElems* 的 Hamming 窗。
HanningInit	初始化一个长度为 *numElems* 的 Hanning 窗。
KaiserInit	初始化一个形状由参数 *betaVal* 确定且长度为 *numElems* 的 Kaiser 窗。
VectorWindow	对给定源向量进行加窗处理，并将加窗后的向量存储至目标向量中。

3.1. 窗运算

窗函数是一种在其域（0 ≤ n < *numElems*）内具有特定数值分布的向量。具体数值分布取决于所生成窗函数的特性。

对某一向量进行加窗处理可修改其数值分布，此时窗函数的元素个数必须与要修改的向量相同。

在对向量进行加窗处理之前，必须先创建相应的窗函数。通过窗初始化运算，可以生成窗函数各元素的值。为了提高数值精度，这些值采用浮点算术进行计算，计算结果采用 1.15/1.31 格式的小数进行存储。

为了避免加窗运算产生过多的开销，可在程序执行期间预先生成一个特定的窗函数并多次复用。因此，建议将任何初始化运算返回的窗函数存入永久（静态）向量中。

3.2. 用户注意事项

使用窗函数时，请注意以下事项：

1. 所有窗初始化函数生成的窗向量均分配到默认的 RAM 存储空间（X 数据空间或 Y 数据空间）中。
2. 窗函数旨在对分配到默认 RAM 存储空间（X 数据空间或 Y 数据空间）中的向量进行操作。
3. 建议在完成每个函数调用之后检查状态寄存器（SR）。

3.3. 函数

3.3.1. BartlettInit

说明

`BartlettInit` 用于初始化一个长度为 *numElems* 的 Bartlett 窗。

原型

```
fractional* BartlettInit(int numElems, fractional* window);
```

参数

参数	说明
numElems	源矩阵的行数。
window	源矩阵的列数。

返回

指向已初始化窗的基址的指针。

备注

窗向量必须已存在，并且恰好包含 *numElems* 个元素。

源文件

- 对于 dsPIC30F/33F/33E/33C/33A
  `initbart.c`（c 代码实现）

3.3.2. BlackmanInit

说明

`BlackmanInit` 用于初始化一个长度为 *numElems* 的 Blackman（三项）窗。

原型

```
fractional* BlackmanInit(int numElems, fractional* window);
```

参数

参数	说明
numElems	源矩阵的行数。
window	源矩阵的列数。

返回

指向已初始化窗的基址的指针。

备注

窗向量必须已存在，并且恰好包含 *numElems* 个元素。

源文件

- 对于 dsPIC30F/33F/33E/33C/33A
  `initblck.c`（c 代码实现）

3.3.3. HammingInit

说明

`HammingInit` 用于初始化一个长度为 *numElems* 的 Hamming 窗。

原型

```
fractional* HammingInit(int numElems, fractional* window);
```

参数

参数	说明
numElems	源矩阵的行数。
window	源矩阵的列数。

返回

指向已初始化窗的基址的指针。

备注

窗向量必须已存在，并且恰好包含 *numElems* 个元素。

源文件

- 对于 dsPIC30F/33F/33E/33C/33A
  `inithamm.c`（c 代码实现）

3.3.4. HanningInit

说明

HanningInit 用于初始化一个长度为 *numElems* 的 Hanning 窗。

原型

```
fractional* HanningInit(int numElems, fractional* window);
```

参数

参数	说明
numElems	源矩阵的行数。
window	源矩阵的列数。

返回

指向已初始化窗的基址的指针。

备注

窗向量必须已存在，并且恰好包含 *numElems* 个元素。

源文件

- 对于 dsPIC30F/33F/33E/33C/33A
  inithann.c（C 代码实现）

3.3.5. KaiserInit

说明

KaiserInit 用于初始化一个长度为 *numElems* 的 Kaiser 窗。

原型

```
fractional* KaiserInit(int numElems, fractional* window, float betaVal);
```

参数

参数	说明
numElems	源矩阵的行数。
window	源矩阵的列数。
betaVal	窗整形参数。

返回

指向已初始化窗的基址的指针。

备注

窗向量必须已存在，并且恰好包含 *numElems* 个元素。

源文件

- 对于 dsPIC30F/33F/33E/33C/33A
  initkais.c（c 代码实现）

3.3.6. VectorWindow

说明

VectorWindow 用于对给定源向量进行加窗处理，并将加窗后的向量存储至目标向量中。

原型

```
fractional* VectorWindow (int numElems, fractional* dstV, fractional* srcV,
fractional* window);
```

参数

参数	说明
numElems	源向量中的元素个数。
dstV	指向目标向量的指针。
srcV	指向源向量的指针。
window	指向已初始化窗的指针。

返回

指向目标向量的基址的指针。

备注

窗向量必须已完成初始化，并且恰好包含 *numElems* 个元素。

该函数可就地计算。

该函数可自应用。

该函数使用 *VectorMultiply*。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `dowindow.s`
- 对于 dsPIC33A
  `dowindow_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	3	9
dsPIC33E/33C	3	9
dsPIC33A	2	11

注：

1. 上述程序字数和周期数仅与 *VectorWindow* 有关。但是，由于该函数本身利用 *VectorMultiply*，因此还必须考虑 *VectorMultiply* 的程序字数和周期数。
2. 在 *VectorMultiply* 的说明中，报告的周期数包括四个周期的 C 函数调用开销。因此，从 *VectorMultiply* 加到 *VectorWindow* 的实际周期数比单独 *VectorMultiply* 报告的周期数少四个。

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C/33A
  - 无（*VectorWindow* 仅包含调用 *VectorMultiply* 的函数）。

4. 矩阵函数

4.1. 函数

函数	说明
MatrixAdd	将源一矩阵中每个元素的值与源二矩阵中对应元素的值相加，并将结果放入目标矩阵中。
MatrixMultiply	执行源一矩阵与源二矩阵之间的矩阵乘法，并将结果放入目标矩阵。
MatrixScale	将源矩阵中所有元素的值按一定比例缩放（乘以一定的缩放值）并将结果放入目标矩阵中。
MatrixSubtract	从源一矩阵中每个元素的值中减去源二矩阵中对应元素的值，并将结果放入目标矩阵中。
MatrixTranspose	将源矩阵中的行与列进行转置，并将结果放入目标矩阵中。
MatrixInvert	对源矩阵求逆并将结果放入目标矩阵中。

4.2. 小数矩阵运算

小数矩阵由一系列在存储器中连续分配的数值元素构成，第一个元素位于最低存储器地址处。每个元素占用一个字的存储空间，其数值必须解释为以 1.15/1.31 格式表示的小数。

指向矩阵第一个元素的指针作为句柄，用于授权访问矩阵内的每个值。第一个元素的地址称为矩阵的基址（*BA*）。由于矩阵的每个元素都是 16 位/32 位，因此基址必须按两字节/四字节对齐。

通过将矩阵元素按行主序排列，可以在存储区域中模拟矩阵的二维排列。因此，存储器中的第一个值是第一行的第一个元素，其后依次为第一行的其余元素。接着存储第二行的元素，以此类推，直到所有行均存入存储器为止。因此，对于具有 *R* 行、*C* 列的矩阵，其第 *r* 行、第 *c* 列的元素的地址可通过矩阵基址（BA）进行偏移计算：

*BA + (sizeof(fractional)) * (C(r - 1) + c - 1)，其中 1 ≤ r ≤ R，1 ≤ c ≤ C。*

该库实现了一元和二元小数矩阵运算。一元运算中的操作数矩阵称为源矩阵。在二元运算中，第一个操作数矩阵称为源一矩阵，第二个操作数矩阵称为源二矩阵。每个运算都会对源矩阵的一个或多个元素进行特定计算。运算结果为一个矩阵，称为目标矩阵。

部分生成矩阵的运算允许就地计算。这意味着，运算结果会回写到源矩阵中（对于二元运算则为源一矩阵）。在这种情况下，目标矩阵将（物理上）取代源（一）矩阵。如果某个运算可就地计算，则会在函数说明提供的备注中作此标示。

对于某些二元运算，两个操作数可以是同一（物理）源矩阵，这意味着该运算将应用于源矩阵与其自身。如果对于给定的运算可以进行这种类型的计算，则会在函数说明提供的备注中作此标示。

一些运算既可自应用，也可就地计算。

该库中的所有小数矩阵运算都将操作数矩阵的行数和列数作为参数。根据这些参数的值，将做出以下假设：

1. 特定运算中涉及的所有矩阵的大小总和在目标器件可用数据存储器的容量范围内。
2. 对于二元运算，操作数矩阵的行数和列数必须遵守向量代数的规则（即，对于矩阵加法和减法，两个矩阵的行数和列数必须相同；对于矩阵乘法，第一个操作数的列数必须与第二个操作数的行数相同）。参与求逆运算的源矩阵必须是方阵（行数与列数相同）且满足非奇异条件（其行列式不等于零）。

目标矩阵必须足够大以容纳运算的结果。

4.3. 用户注意事项

使用矩阵函数时，请注意以下事项：

1. 这些函数不执行边界检查。当使用的源矩阵尺寸超限（包括零行和/或零列矩阵），以及在二元运算中使用的源矩阵尺寸不匹配时，可能会产生意外的结果。

2. 在矩阵加减运算中，如果源矩阵中对应元素之和大于 $1-2^{-15}$（对于 1.15 小数格式）或 $1-2^{-31}$（对于 1.31 小数格式）或者小于-1，则可能导致饱和。
3. 在矩阵乘法运算中，如果对应行集与列集的乘积之和大于 $1-2^{-15}$（对于 1.15 小数格式）或 $1-2^{-31}$（对于 1.31 小数格式）或者小于-1，则可能导致饱和。
4. 建议在完成每个函数调用之后检查状态寄存器（SR）。特别是，用户可在函数返回后检查 SA、SB 和 SAB 标志以判断是否发生饱和。
5. 返回目标矩阵的运算可进行嵌套。例如，如果：
   a = Op1 (b, c)，其中 b = Op2 (d)且 c = Op3 (e, f)，则
   a = Op1 (Op2 (d), Op3 (e, f))
6. dsPIC33A 的所有周期数值均在 PBU 高速缓存使能的情况下测得，实际值可能因 PBU 高速缓存状态或者向量与代码的存放位置而异。

4.4. 补充说明

函数说明通常会限制这些运算的常规使用范围。但是，由于在计算这些函数时不执行边界检查，因此会根据特定需求解释运算及其结果。

例如，在执行 *MatrixMultiply* 函数时，源一矩阵的尺寸并非必须满足{*numRows1, numCos1Rows2*}，源二矩阵的尺寸并非必须满足{*numCols1Rows2, numCols2*}，目标矩阵的尺寸并非必须满足{*numRows1, numCols2*}。实际上，只要这些矩阵的尺寸足够大，能够保证在计算过程中指针不会超出存储器范围即可。

又如，当对尺寸为{*numRows, numCols*}的源矩阵进行转置时，目标矩阵的尺寸为{*numCols, numRows*}。因此，仅当源矩阵为方阵时，才能就地计算该运算。但是，该运算可以成功地应用于非方阵；切记其尺寸将发生隐式变化。

鉴于不会执行边界检查，因此可以开发利用其他可能性。

4.5. 函数

4.5.1. MatrixAdd

说明

`MatrixAdd` 用于将源一矩阵中每个元素的值与源二矩阵中对应元素的值相加，并将结果放入目标矩阵中。

原型

```
fractional* MatrixAdd (int numRows, int numCols, fractional* dstM,
fractional* srcM1, fractional* srcM2);
```

参数

参数	说明
numRows	源矩阵的行数。
numCols	源矩阵的列数。
dstM	指向目标矩阵的指针
srcM1	指向源一矩阵的指针
srcM2	指向源二矩阵的指针

返回

指向目标矩阵的基址的指针。

备注

如果 *srcM1[r][c]* + *srcM2[r][c]*的绝对值大于 1.15/1.31 小数数据类型的最大值，则该运算将导致第 n 个元素饱和。

该函数可就地计算。

该函数可自应用。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `madd.s`
- 对于 dsPIC33A
  `madd_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	14	20 + 3*(*numRows* * *numCols*)
dsPIC33E/33C	17	28 + 3*(*numRows* * *numCols*)
dsPIC33A	11	38 + 4*(*numRows* * *numCols*)

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W4*——已使用，未恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 一级 *DO* 指令
    - 无 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W5*——已使用，未恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——无

4.5.2. MatrixMultiply

说明

`MatrixMultiply` 用于执行源一矩阵与源二矩阵之间的矩阵乘法，并将结果放入目标矩阵。

其符号表示如下：

$$dstM[i][j] = \sum_{k=0}^{numCols1Rows2-1} src1M[i][k] * src2M[k][j]$$

其中：

$0 \leqslant i < numRows1$

$0 \leqslant j < numCols2$

$0 \leqslant k < numCols1Rows2$

原型

```
fractional* MatrixMultiply (int numRows1, int numCols1Rows2, int numCols2,
fractional* dstM, fractional* srcM1, fractional* srcM2);
```

参数

参数	说明
numRows1	源一矩阵的行数。
numCols1Rows2	源一矩阵的列数，与源二矩阵的行数相同。
numCols2	源二矩阵的列数。
dstM	指向目标矩阵的指针
srcM1	指向源一矩阵的指针
srcM2	指向源二矩阵的指针

返回

指向目标矩阵的基址的指针。

备注

如果 *dstM[r][c]*的结果绝对值大于 1.15/1.31 小数数据类型的最大值，则该运算将导致第(r,c)个元素饱和。

如果源一矩阵为方阵，则该函数可就地计算并自应用。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `mmul.s`
- 对于 dsPIC33A
  `mmul_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	35	36+*numRows1**(8+*numCols2**(7+4*(*numCols1Rows2*)))
dsPIC33E/33C	38	44+*numRows1**(8+*numCols2**(7+4*(*numCols1Rows2*)))
dsPIC33A	27	45+*numRows1**(6+*numCols2**(9+4*(*numCols1Rows2* - 1)))

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W7*——已使用，未恢复
  - *W8...W13*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 两级 *DO* 指令
    - 无 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W7*——已使用，未恢复
  - *W8...W12*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复

- *CORCON*——已保存，已使用，已恢复
- *REPEAT* 指令的使用——无

4.5.3. MatrixScale

说明

`MatrixScale` 用于将源矩阵中所有元素的值按一定比例缩放（乘以一定的缩放值）并将结果放入目标矩阵中。

原型

```
fractional* MatrixScale (int numRows, int numCols, fractional* dstM,
fractional* srcM, fractional sclVal);
```

参数

参数	说明
numRows	源矩阵的行数。
numCols	源矩阵的列数。
dstM	指向目标矩阵的指针
srcM1	指向源一矩阵的指针
sclVal	用于缩放矩阵元素的比例值

返回

指向目标矩阵的基址的指针。

备注

该函数可就地计算。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `mscl.s`
- 对于 dsPIC33A
  `mscl_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	14	20 + 3*(*numRows***numCols*)
dsPIC33E/33C	17	28 + 3*(*numRows***numCols*)
dsPIC33A	20	44 + ⌈2.5 x *numElems*⌉

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W4*——已使用，未恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 一级 *DO* 指令
    - 无 *REPEAT* 指令

- 对于 dsPIC33A
  - *W0...W5*——已使用，未恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——无

4.5.4. MatrixSubtract

说明

`MatrixSubtract` 用于从源一矩阵中每个元素的值中减去源二矩阵中对应元素的值，并将结果放入目标矩阵中。

原型

```
fractional* MatrixAdd (int numRows, int numCols, fractional* dstM,
fractional* srcM1, fractional* srcM2);
```

参数

参数	说明
numRows	源矩阵的行数。
numCols	源矩阵的列数。
dstM	指向目标矩阵的指针
srcM1	指向源一矩阵的指针
srcM2	指向源二矩阵的指针

返回

指向目标矩阵的基址的指针。

备注

如果 *srcM1[r][c] - srcM2[r][c]*的绝对值大于 1.15/1.31 小数数据类型的最大值，则该运算将导致第 n 个元素饱和。

该函数可就地计算。

该函数可自应用。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `msub.s`
- 对于 dsPIC33A
  `msub_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	14	20 + 3*(*numRows* * *numCols*)
dsPIC33E/33C	17	28 + 3*(*numRows* * *numCols*)
dsPIC33A	11	38 + 4*(*numRows* * *numCols*)

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C

  - *W0...W4*——已使用，未恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 一级 *DO* 指令
    - 无 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W5*——已使用，未恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——无

4.5.5. MatrixTranspose

说明

`MatrixTranspose` 用于将源矩阵中的行与列进行转置，并将结果放入目标矩阵中。

实际上：

dstM[i][j] = srcM[j][i]，

0 ≤ i < numRows，*0 ≤ j < numCols*。

原型

```
fractional* MatrixTranspose (int numRows, int numCols, fractional* dstM,
fractional* srcM);
```

参数

参数	说明
numRows	源矩阵的行数。
numCols	源矩阵的列数。
dstM	指向目标矩阵的指针
srcM	指向源矩阵的指针

返回

指向目标矩阵的基址的指针。

备注

如果源矩阵是方阵，则可就地计算该函数。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `mtrp.s`
- 对于 dsPIC33A
  `mtrp_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	14	16 + *numCols* *(6 + (*numRows-1)*3*)

MatrixTranspose （续）		
器件	程序字数	周期数
dsPIC33E/33C	17	24 + *numCols* *(6 + (*numRows-1*)*2*)
dsPIC33A	9	23 + *numCols* *(7 + (*numRows-1*)*4*)

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W4*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 两级 *DO* 指令
    - 无 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W5*——已使用，未恢复
  - *REPEAT* 指令的使用——无

4.5.6. MatrixInvert

对非奇异方阵小数矩阵求逆后的结果是另一个方阵（尺寸与原矩阵相同），其元素值可能超出离散小数集范围{-1, ..., ~1}。因此，未提供针对小数矩阵的求逆运算。

但是，由于矩阵求逆是一种非常有用的运算，因此 DSP 库中提供了基于浮点数表示和算术的实现。

说明

`MatrixInverse` 用于对源矩阵求逆并将结果放入目标矩阵中。

原型

```
float* MatrixInvert (int numRowsCols, float* dstM, float* srcM,
float* pivotFlag, int* swappedRows, int* swappedCols);
```

参数

参数	说明
numRowsCols	源矩阵（方阵）的行数/列数。
dstM	指向目标矩阵的指针
srcM	指向源矩阵的指针
pivotFlag	指向长度为 *numRowsCols* 的向量的指针
swappedRows	指向长度为 *numRowsCols* 的向量的指针
swappedCols	指向长度为 *numRowsCols* 的向量的指针

返回

指向目标矩阵的基址的指针，如果源矩阵是奇异矩阵则为 NULL。

备注

尽管向量 *pivotFlag*、*swappedRows* 和 *swappedCols* 仅供内部使用，但调用该函数之前仍必须为其分配存储空间。

如果源矩阵是奇异矩阵（行列式等于零），则该矩阵没有逆矩阵。在这种情况下，该函数返回 NULL。

该函数可就地计算。

源文件

- 对于 dsPIC30F/33F/33E/33C/33A
  `minv.c`（由 C 代码汇编而成）

5. 滤波函数

函数	说明
FIRStruct	用于描述任何 FIR 滤波器结构的结构体。
FIR	将 FIR 滤波器应用于源采样序列，将结果放入目标采样序列中并更新延时值。
FIRDecimate	以 R:1 比率对源采样序列进行抽取，等效于对信号进行 R 倍降采样。
FIRDelayInit	将 *FIRStruct* 滤波器结构中的延时值初始化为零。
FIRInterpolate	以 1:R 比率对源采样序列进行插值，等效于对信号进行 R 倍升采样。
FIRInterpDelayInit	将 *FIRStruct* 滤波器结构中的延时值初始化为零，并针对与 FIR 插值滤波器配合使用进行了优化。
FIRLattice	使用栅格结构实现将 FIR 滤波器应用于源采样序列。
FIRLMS	将自适应 FIR 滤波器应用于源采样序列，将结果存入目标采样序列中并更新延时值。
FIRLMSNorm	将自适应 FIR 滤波器应用于源采样序列，将结果存入目标采样序列中并更新延时值。此外，还会根据参考采样的值应用归一化最小均方算法逐采样更新滤波器系数。
FIRStructInit	初始化 *FIRStruct* FIR 滤波器结构中的参数值。
IIRCanonic	通过级联规范（直接 II 型）双二次节将 IIR 滤波器应用于源采样序列。
IIRCanonicInit	将 *IIRCanonicStruct* 滤波器结构中的延时值初始化为零。
IIRLattice	使用栅格结构实现将 IIR 滤波器应用于源采样序列。
IIRLatticeInit	将 *IIRLatticeStruct* 滤波器结构中的延时值初始化为零。
IIRTransposed	通过级联转置（直接 II 型）双二次节将 IIR 滤波器应用于源采样序列。
IIRTransposedInit	将 *IIRTransposedStruct* 滤波器结构中的延时值初始化为零。

5.1. 小数滤波器运算

对小数向量 *x[n]*（*0* ≤ *n* < *N*）表示的数据序列进行滤波相当于每 n 个采样求解一次下面的差分方程，从而得到滤波后的数据序列 *y[n]*。

$$y[n] + \sum_{p=1}^{P-1} -a[p] \times y[n-p] = \sum_{m=1}^{M-1} b[m] \times x[n-m]$$

从这个意义上讲，小数滤波器通过小数向量 *a[p]*（*0* ≤ *p* < *P*）和 *b[m]*（*0* ≤ *m* < *M*）表征，两者称为滤波器系数集，旨在对由输入数据序列表示的信号引起一些预先指定的变化。

滤波时，务必了解并管理输入和输出数据序列（*x[n]*（*-M + 1* ≤ *n* < *0*）和 *y[n]*（*-P + 1* ≤ *n* < *0*））的历史记录（表示滤波运算的初始条件）。此外，当重复将滤波器应用于输入数据序列的连续部分时，还需切记最后一次滤波运算的最终状态（*x[n]*（*N* – *M + 1* ≤ *n* < *N* – *1*）和 *y[n]*（*N* – *P + 1* ≤ *n* < *N* – *1*））。该最终状态随后纳入下一滤波阶段的计算中。为确保执行正确的滤波运算，需考虑历史记录和当前状态。

滤波运算的历史状态和当前状态通常通过附加序列（也称为小数向量）进行管理，这些序列称为滤波器延时线。在应用滤波运算之前，延时描述的是滤波器的历史状态。在执行滤波运算之后，延时包含一组最近滤波后的数据采样和最新的输出采样。

注：为确保特定滤波器实现正常工作，建议通过调用相应的初始化函数将延时值初始化为零。

在 DSP 库提供的滤波器实现中，输入数据序列称为源采样序列，而经过滤波后的序列称为目标采样。通常，滤波器系数（*a,b*）与延时共同构成滤波器结构。在所有滤波器实现中，可以将输入和输出数据采样分配到默认 RAM 存储空间（X 数据空间或 Y 数据空间）中。滤波器系数可位于 X 数据存储区或程序存储器中，而滤波器延时值只能从 Y 数据存储区访问。

注：dsPIC33A 架构不强制要求滤波器延时值位于 Y 数据空间中。但是，不将其放入 Y 数据空间中会导致按顺序执行操作数访问，进而影响数据获取效率。

5.2. FIR 和 IIR 滤波器实现

滤波器的属性取决于其系数的数值分布。其中两类滤波器需特别关注：第一类是有限冲激响应（Finite Impulse Response，FIR）滤波器，当 $1 \leqslant m < M$ 时，$a[m] = 0$；第二类是无限冲激响应（Infinite Impulse Response，IIR）滤波器，$a[0] \neq 0$ 且{1, ..., M}中的某个值 m 对应的 $a[m] \neq 0$。FIR 和 IIR 滤波器系列的其他分类基于其运算对输入数据序列的影响。

再者，尽管滤波涉及求解上述差分方程，但有几种实现方式比直接计算差分方程更为高效。此外，还有一些其他实现专门用于在小数算术施加的约束下执行滤波运算。

因此，滤波运算的数量激增，DSP 库中只提供了其中的一部分。

5.3. 单采样滤波

DSP 库中提供的滤波函数采用块处理设计。每个滤波函数接受一个名为 *numSamps* 的参数，其表示将参与运算的输入数据字数（块大小）。如果需要单采样滤波，可将 *numSamps* 设置为 1。这样即可对一个输入采样进行滤波，并且滤波函数将计算出一个输出采样。

5.4. 用户注意事项

该库中的所有小数滤波运算都依赖于输入参数或数据结构元素的值来指定要处理的采样数以及系数和延时向量的大小。根据这些值，将做出以下假设：

1. 特定运算中涉及的所有向量（采样序列）的大小总和在目标器件可用数据存储器的容量范围内。
2. 目标向量必须足够大以容纳运算的结果。
3. 这些函数不执行边界检查。当使用的源向量尺寸超限（包括长度为零的向量），以及使用的源向量和系数集不匹配时，可能会产生意外的结果。
4. 建议在完成每个函数调用之后检查状态寄存器（SR）。特别是，用户可在函数返回后检查 SA、SB 和 SAB 标志以判断是否发生饱和。
5. 返回目标向量的运算可进行嵌套。例如，如果：
   a = Op1 (b, c)，其中 b = Op2 (d)且 c = Op3 (e, f)，则
   a = Op1 (Op2 (d), Op3 (e, f))
6. dsPIC33A 的所有周期数值均在 PBU 高速缓存使能的情况下测得，实际值可能因 PBU 高速缓存状态或者向量与代码的存放位置而异。

5.5. 函数

本节介绍用于实现滤波运算的各个函数。有关数字滤波器的更多信息，请参见 Alan Oppenheim 和 Ronald Schafer 的“Discrete-Time Signal Processing”（Prentice Hall，1989 年）。有关最小均方 FIR 滤波器的实现细节，请参见 T. Hsia 的“Convergence Analysis of LMS and NLMS Adaptive Algorithms”（Proc.ICASSP，第 667-670 页，1983 年）以及 Sangil Park 和 Garth Hillman 的“On Acoustic-Echo Cancellation Implementation with Multiple Cascadable Adaptive FIR Filter Chips”（Proc.ICASSP，1989 年）。

5.5.1. FIRStruct

说明

`FIRStruct` 用于描述任何 FIR 滤波器的结构。

声明

```
typedef struct {
  int numCoeffs;
```

```
  fractional* coeffsBase;
  fractional* coeffsEnd;
  int coeffsPage;
  fractional* delayBase;
  fractional* delayEnd;
  fractional* delay;
} FIRStruct;
```

参数

参数	说明
numCoeffs	滤波器系数的数量（亦即 M）
coeffsBase	滤波器系数的基址（亦即 h）
coeffsEnd	滤波器系数的结束地址
coeffsPage	系数缓冲区页编号
delayBase	延时缓冲区的基址
delayEnd	延时缓冲区的结束地址
delay	延时指针的当前值（亦即 d）

备注

滤波器系数的数量为 M。

系数 h[m]在 0 ≤ m < M 范围内定义，其位于 X 数据空间或程序存储器内。

延时缓冲区 d[m]在 0 ≤ m < M 范围内定义，其仅位于 Y 数据空间中。

对于 dsPIC30F/33F/33E/33C，如果系数存储在 X 数据空间中，*coeffsBase* 指向系数实际分配到的地址。如果系数存储在程序存储器中，*coeffsBase* 是从包含系数的程序页边界到系数分配到的页地址的偏移量。该偏移量可以使用嵌入汇编运算符 *psvoffset()*进行计算。

coeffsEnd 是滤波器系数缓冲区的最后一个字节在 X 数据空间中的地址（如果在程序存储器中则为偏移量）。

如果系数存储在 X 数据空间中，*coeffsPage* 必须设置为 0xFF00（*COEFFS_IN_DATA* 的定义值）。如果系数存储在程序存储器中，则为包含系数的程序页编号。该程序页编号可以使用嵌入汇编运算符 *psvpage()*进行计算。

由于 dsPIC33A 系列器件实现了非分页线性 RAM/程序存储空间，无论系数存储在程序存储器还是 RAM 中都指向实际位置，因此将忽略结构成员 *coeffsPage*。

delayBase 指向在延时缓冲区中实际分配的地址。

delayEnd 是滤波器延时缓冲区的最后一个字节的地址。

当系数缓冲区和延时缓冲区实现为循环递增模缓冲区时，*coeffsBase* 和 *delayBase* 这两个地址必须按 2 的“零”次幂对齐（*coeffsEnd* 和 *delayEnd* 为奇数地址）。关于这两个缓冲区是否实现为循环递增模缓冲区，会在每个 FIR 滤波器函数说明的备注部分进行标示。由于不能跨页边界进行模寻址（进入和退出默认页 0 时除外），因此必须正确分配使用模寻址的运算中的 *coeffsBase* 和 *delayBase* 向量。

当系数缓冲区和延时缓冲区未实现为循环（递增）模缓冲区时，*coeffsBase* 和 *delayBase* 这两个地址无需按 2 的“零”次幂对齐，并且在特定 FIR 滤波器函数实现内将忽略 *coeffsEnd* 和 *delayEnd* 的值。

5.5.2. FIR

说明

`FIR` 用于将 FIR 滤波器应用于源采样序列，将结果放入目标采样序列中并更新延时值。

原型

```
fractional* FIR (int numSamps, fractional* dstSamps, fractional* srcSamps,
FIRStruct* filter);
```

参数

参数	说明
numSamps	待滤波的输入采样数（亦即 *N*）
dstSamps	指向目标采样（亦即 *y*）的指针
srcSamps	指向源采样（亦即 *x*）的指针
filter	指向 *FIRStruct* 滤波器结构的指针

返回

指向目标采样的基址的指针。

备注

滤波器系数的数量为 M。

系数 h[m]在 0 ≤ m < M 范围内定义，其实现为循环递增模缓冲区。

延时 d[m]在 0 ≤ m < M 范围内定义，其实现为循环递增模缓冲区。延时向量必须位于 Y 数据空间中。

源采样 x[n]在 0 ≤ n < N 范围内定义。

目标采样 y[n]在 0 ≤ n < N 范围内定义。

（另见 *FIRStruct*、*FIRStructInit* 和 *FIRDelayInit*。）

对于 dsPIC33E/33C：

当 *coeffsPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + *TABLE_SIZE* < *__YDATA_BASE*

其中，

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 *STACK_GUARD* 时必须小心。

- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 *STACK_GUARD* 的值。
- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 *STACK_GUARD* 的值。

对于 dsPIC33A 器件，将忽略 *coeffsPage*。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `fir.s`
- 对于 dsPIC33A
  `fir_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	61	61 + N*(4 + M)（如果系数位于数据存储器中） 68 + N*(9 + M)（如果系数位于程序存储器中）
dsPIC33E/33C	97	73 + N*(4 + M)（如果系数位于数据存储器中） (101 + M) + N*(4 + M)（如果系数位于程序存储器中但被复制到数据存储器中） 83 + N*(24 + M)（如果系数位于程序存储器中且未复制到数据存储器中）
dsPIC33A	40	*见下表

dsPIC33A 的周期数：

源向量大小	系数位于 X 存储区中时的周期数	系数位于 P 存储区中时的周期数
32	1222	3112
64	2376	6152
128	4676	12232
256	9288	24392
512	18500	48712
1024	36936	97352
2048	73796	194632

*（*所有值的测量条件为 `numCoeffs = 32`）*

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W7*——已使用，未恢复
  - *W8...W14*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *MODCON*——已保存，已使用，已恢复
  - *XMODSTRT*——已保存，已使用，已恢复
  - *XMODEND*——已保存，已使用，已恢复
  - *YMODSTRT*——已保存，已使用，已恢复
  - *YMODEND*——已保存，已使用，已恢复
  - *PSVPAG*——已保存，已使用，已恢复
  - *DSRPAG*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用

  - 一级 *DO* 指令
  - 一条 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W7*——已使用，未恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *MODCON*——已保存，已使用，已恢复
  - *XMODSTRT*——已保存，已使用，已恢复
  - *XMODEND*——已保存，已使用，已恢复
  - *YMODSTRT*——已保存，已使用，已恢复
  - *YMODEND*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——1 条

5.5.3. FIRDecimate

说明

`FIRDecimate` 用于以 R:1 比率对源采样序列进行抽取，等效于对信号进行 R 倍降采样。

实际上，

y[n] = x[Rn]。

为了减少混叠的影响，先对源采样进行滤波再进行降采样。抽取结果存入目标采样序列中，同时更新延时值。

原型

```
fractional* FIRDecimate (int numSamps, fractional* dstSamps,
fractional* srcSamps, FIRStruct* filter, int rate);
```

参数

参数	说明
numSamps	待滤波的输出采样数（亦即 *N*；*N 是 R 的倍数*）
dstSamps	指向目标采样（亦即 *y*）的指针
srcSamps	指向源采样（亦即 *x*）的指针
filter	指向 *FIRStruct* 滤波器结构的指针
rate	抽取速率（降采样因子，亦即 R）

返回

指向目标采样的基址的指针。

备注

滤波器系数的数量为 M，M 为 R 的整数倍。

系数 h[m]在 0 ≤ m < M 范围内定义，其未实现为循环模缓冲区。

延时 d[m]在 0 ≤ m < M 范围内定义，其未实现为循环模缓冲区。

源采样 x[n]在 0 ≤ n < NR 范围内定义。

目标采样 y[n]在 0 ≤ n < N 范围内定义。

（另见 *FIRStruct*、*FIRStructInit* 和 *FIRDelayInit*。）

对于 dsPIC33E/33C：

当 *coeffsPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + *TABLE_SIZE* < *__YDATA_BASE*

其中，

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 *STACK_GUARD* 时必须小心。

- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 *STACK_GUARD* 的值。
- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 *STACK_GUARD* 的值。

对于 dsPIC33A 器件，将忽略 `coeffsPage` 的值。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `firdecim.s`
- 对于 dsPIC33A
  `firdecim_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	53	52 + N*(9+ 2M)（如果系数位于数据存储器中） 55 + N*(13 + 2M)（如果系数位于程序存储器中）
dsPIC33E/33C	96	61 + N*(10 + 2M)（如果系数位于数据存储器中） (101 + M) + N*(10 + 2M)（如果系数位于程序存储器中但被复制到数据存储器中） 71 + N*(25 + 2M)（如果系数位于程序存储器中且未复制到数据存储器中）
dsPIC33A	30	*见下表

dsPIC33A 的周期数：

源向量大小	系数位于 X 存储区中时的周期数	系数位于 P 存储区中时的周期数
32	2352	4240

FIRDecimate （续）		
源向量大小	系数位于 X 存储区中时的周期数	系数位于 P 存储区中时的周期数
64	4660	8432
128	9258	16816
256	18484	33584
512	36916	67120
1024	73780	134192
2048	147508	268336

*（*所有值的测量条件为 numCoeffs = 32; rate = 2）*

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W7*——已使用，未恢复
  - *W8...W14*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *PSVPAG/DSRPAG*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 一级 *DO* 指令
    - 一条 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W7*——已使用，未恢复
  - *W8...W11*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——2 条

5.5.4. FIRDelayInit

说明

FIRDelayInit 用于将 *FIRStruct* 滤波器结构中的延时值初始化为零。

原型

```
void FIRDelayInit (FIRStruct* filter);
```

参数

参数	说明
filter	指向 *FIRStruct* 滤波器结构的指针

返回

无。

备注

请参见上文 *FIRStruct* 结构的说明。

注：通过 *FIRInterpDelayInit* 函数初始化 FIR 插值器的延时。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `firdelay.s`
- 对于 dsPIC33A
  `firdelay_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	7	11 + M
dsPIC33E/33C	10	20 + M
dsPIC33A	6	22 + M

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W2*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 无 *DO* 指令
    - 一条 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W2*——已使用，未恢复
  - *REPEAT* 指令的使用——1 条

5.5.5. FIRInterpolate

说明

`FIRInterpolate` 用于以 1:R 比率对源采样序列进行插值，等效于对信号进行 R 倍升采样。

实际上，

y[n] = x[n/R]。

为了减少混叠的影响，先对源采样进行升采样再进行滤波。插值结果存入目标采样序列中，同时更新延时值。

原型

```
fractional* FIRInterpolate (int numSamps, fractional* dstSamps,
fractional* srcSamps, FIRStruct* filter, int rate);
```

参数

参数	说明
numSamps	待滤波的输入采样数（亦即 *N*；*N 是 R 的倍数*）
dstSamps	指向目标采样（亦即 *y*）的指针
srcSamps	指向源采样（亦即 *x*）的指针
filter	指向 *FIRStruct* 滤波器结构的指针
rate	插值速率（升采样因子，亦即 R）

返回

指向目标采样的基址的指针。

备注

滤波器系数的数量为 M，M 为 R 的整数倍。

系数 h[m]在 0 ≤ m < M 范围内定义，其未实现为循环模缓冲区。

延时 d[m]在 0 ≤ m < M/R 范围内定义，其未实现为循环模缓冲区。

源采样 x[n]在 0 ≤ n < N 范围内定义。

目标采样 y[n]在 0 ≤ n < NR 范围内定义。

（另见 *FIRStruct*、*FIRStructInit* 和 *FIRInterpDelayInit*。）

对于 dsPIC33E/33C 器件：

当 *coeffsPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + *TABLE_SIZE* < *__YDATA_BASE*

其中，

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 *STACK_GUARD* 时必须小心。

- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 *STACK_GUARD* 的值。
- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 *STACK_GUARD* 的值。

对于 dsPIC33A 器件，将忽略 `coeffsPage` 的值。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `firinter.s`
- 对于 dsPIC33A
  `firinter_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	68	`50 + 6*(M/R) + N*(14 + M/R + 3M + 4R)`（如果系数位于数据存储器中） `54 + 6*(M/R) + N*(14 + M/R + 4M + 4R)`（如果系数位于程序存储器中）

FIRInterpolate （续）		
器件	**程序字数**	**周期数**
dsPIC33E/33C	127	61 + 8*(M/R) + N*(16 + M/R + 3M + 5R)（如果系数位于数据存储器中） (86 + M) + 8*(M/R) + N*(16 + M/R + 3M + 5R)（如果系数位于程序存储器中但被复制到数据存储器中） 76 + 8*(M/R) + N*(19 + M/R + 7M + 6R)（如果系数位于程序存储器中且未复制到数据存储器中）
dsPIC33A	36	*见下表

dsPIC33A 的周期数：

源向量大小	系数位于 X 存储区中时的周期数	系数位于 P 存储区中时的周期数
32	5344	9002
64	10624	17930
128	21184	35786
256	42304	71498
512	84544	142922
1024	169024	285770
2048	337984	571466

*（*所有值的测量条件为 `numCoeffs = 32; rate = 2`）*

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W7*——已使用，未恢复
  - *W8...W14*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *PSVPAG/DSRPAG*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 两级 *DO* 指令
    - 一条 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W7*——已使用，未恢复
  - *W8...W12*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——2 条

5.5.6. FIRInterpDelayInit

说明

`FIRInterpDelayInit` 用于将 *FIRStruct* 滤波器结构中的延时值初始化为零，并针对与 FIR 插值滤波器配合使用进行了优化。

原型

```
void FIRInterpDelayInit (FIRStruct* filter, int rate);
```

参数

参数	说明
filter	指向 *FIRStruct* 滤波器结构的指针。
rate	插值速率（升采样因子，亦即 R）

返回

无。

备注

延时 d[m]在 0 ≤ m < M/R 范围内定义，其中 M 是插值器中的滤波器系数的数量。请参见上文 *FIRStruct* 结构的说明。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `firinterpdelay.s`
- 对于 dsPIC33A
  `firinterpdelay_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	13	10 + 7*(M/R)
dsPIC33E/33C	16	22 + 8*(M/R)
dsPIC33A	7	32 + M/R

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W4*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 无 *DO* 指令
    - 一条 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W3*——已使用，未恢复
  - *REPEAT* 指令的使用——2 条

5.5.7. FIRLattice

说明

`FIRLattice` 使用栅格结构实现将 FIR 滤波器应用于源采样序列。然后，将结果放入目标采样序列中并更新延时值。

原型

```
fractional* FIRLattice (int numSamps, fractional* dstSamps,
fractional* srcSamps, FIRStruct* filter);
```

参数

参数	说明
numSamps	待滤波的输入采样数（亦即 *N*；*N 是 R 的倍数*）
dstSamps	指向目标采样（亦即 *y*）的指针
srcSamps	指向源采样（亦即 *x*）的指针
filter	指向 *FIRStruct* 滤波器结构的指针

返回

指向目标采样的基址的指针。

备注

滤波器系数的数量为 M。

系数 h[m]在 0 ≤ m < M 范围内定义，其未实现为循环模缓冲区。

延时 d[m]在 0 ≤ m < M 范围内定义，其未实现为循环模缓冲区。

源采样 x[n]在 0 ≤ n < N 范围内定义。

目标采样 y[n]在 0 ≤ n < N 范围内定义。

（另见 *FIRStruct*、*FIRStructInit* 和 *FIRDelayInit*。）

对于 dsPIC33E/33C 器件：

当 *coeffsPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + *TABLE_SIZE* < *__YDATA_BASE*

其中，

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 *STACK_GUARD* 时必须小心。

- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 *STACK_GUARD* 的值。
- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 *STACK_GUARD* 的值。

对于 dsPIC33A 器件，将忽略 *coeffsPage* 的值。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `firlatt.s`
- 对于 dsPIC33A
  `firlatt_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	54	47 + N*(4 + 7M)（如果系数位于数据存储器中） 50 + N*(4 + 8M)（如果系数位于程序存储器中）
dsPIC33E/33C	102	56 + N*(4 + 7M)（如果系数位于数据存储器中） (81 + M) + N*(4 + 7M)（如果系数位于程序存储器中但被复制到数据存储器中） 66 + N*(3 + 15M)（如果系数位于程序存储器中且未复制到数据存储器中）
dsPIC33A	32	*见下表

dsPIC33A 的周期数：

源向量大小	系数位于 X 存储区中时的周期数	系数位于 P 存储区中时的周期数
32	6410	10154
64	12782	20266
128	25518	40490
256	50990	80938
512	101934	161834
1024	203822	323626
2048	407597	647210

*（*所有值的测量条件为 `numCoeffs = 32`）*

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W7*——已使用，未恢复
  - *W8...W14*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *PSVPAG/DSRPAG*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 两级 *DO* 指令
    - 一条 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W7*——已使用，未恢复
  - *W8...W10*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——无

5.5.8. FIRLMS

说明

FIRLMS 用于将自适应 FIR 滤波器应用于源采样序列，将结果存入目标采样序列中并更新延时值。

此外，还会根据参考采样的值应用最小均方算法逐采样更新滤波器系数。

原型

```
fractional* FIRLMS (int numSamps, fractional* dstSamps, fractional* srcSamps,
FIRStruct* filter, fractional* refSamps, fractional muVal);
```

参数

参数	说明
numSamps	待滤波的输入采样数（亦即 *N*；*N 是 R 的倍数*）
dstSamps	指向目标采样（亦即 *y*）的指针
srcSamps	指向源采样（亦即 *x*）的指针
filter	指向 *FIRStruct* 滤波器结构的指针
refSamps	指向参考采样（亦即 r）的指针
muVal	适应因子（亦即 mu）

返回

指向目标采样的基址的指针。

备注

滤波器系数的数量为 M。

系数 h[m]在 0 ≤ m < M 范围内定义，其实现为循环模缓冲区。

延时 d[m]在 0 ≤ m < M 范围内定义，其实现为循环模缓冲区，并位于 Y 数据空间中。

源采样 x[n]在 0 ≤ n < N 范围内定义。

参考采样 r[n]在 0 ≤ n < N 范围内定义。

目标采样 y[n]在 0 ≤ n < N 范围内定义。

适应：

$h_m[n] = h_m[n - 1] + mu * (r[n] - y[n]) * x[n - m]$，*其中 0 ≤ n < N，0 ≤ m < M*。

如果(r[n] - y[n])的绝对值大于或等于 1，则该运算可能导致饱和。

不得将滤波器系数分配到程序存储器中，否则可能无法调整其值。如果检测到滤波器系数已分配到程序存储器中，则函数返回 NULL。

（另见 *FIRStruct*、*FIRStructInit* 和 *FIRInterpDelayInit*。）

对于 dsPIC33A 器件，将忽略 *fractPage*。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `firlms.s`
- 对于 dsPIC33A
  `firlms_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	76	67 + N*(13 + 5M)
dsPIC33E/33C	76	67 + N*(13 + 5M)
dsPIC33A	56	*见下表

dsPIC33A 的周期数：

源向量大小	系数位于 X 存储区中时的周期数
32	5586
64	11098
128	22102
256	44122
512	88150
1024	176218
2048	352342

*（*所有值的测量条件为 `numCoeffs = 32`）*

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W7*——已使用，未恢复
  - *W8...W12*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *MODCON*——已保存，已使用，已恢复
  - *XMODSTRT*——已保存，已使用，已恢复
  - *XMODEND*——已保存，已使用，已恢复
  - *YMODSTRT*——已保存，已使用，已恢复
  - *YMODEND*——已保存，已使用，已恢复
  - *PSVPAG/DSRPAG*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 两级 *DO* 指令
    - 一条 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W7*——已使用，未恢复
  - *W8...W12*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *MODCON*——已保存，已使用，已恢复
  - *XMODSTRT*——已保存，已使用，已恢复
  - *XMODEND*——已保存，已使用，已恢复
  - *YMODSTRT*——已保存，已使用，已恢复

- *YMODEND*——已保存，已使用，已恢复
- *REPEAT* 指令的使用——1 条

5.5.9. FIRLMSNorm

说明

`FIRLMSNorm` 用于将自适应 FIR 滤波器应用于源采样序列，将结果存入目标采样序列中并更新延时值。

此外，还会根据参考采样的值应用归一化最小均方算法逐采样更新滤波器系数。

原型

```
fractional* FIRLMSNorm (int numSamps, fractional* dstSamps,
fractional* srcSamps, FIRStruct* filter, fractional* refSamps,
fractional muVal, fractional* energyEstimate);
```

参数

参数	说明
numSamps	待滤波的输入采样数（亦即 *N*；*N 是 R 的倍数*）
dstSamps	指向目标采样（亦即 *y*）的指针
srcSamps	指向源采样（亦即 *x*）的指针
filter	指向 *FIRStruct* 滤波器结构的指针
refSamps	指向参考采样（亦即 r）的指针
muVal	适应因子（亦即 mu）
energyEstimate	指向最后 *M* 个输入采样的估计能量（*e[N-1]*）值的指针。

返回

指向目标采样的基址的指针。

备注

滤波器系数的数量为 M。

系数 h[m]在 0 ≤ m < M 范围内定义，其实现为循环模缓冲区。

延时 d[m]在 0 ≤ m < M 范围内定义，其实现为循环模缓冲区。

源采样 x[n]在 0 ≤ n < N 范围内定义。

参考采样 r[n]在 0 ≤ n < N 范围内定义。

目标采样 y[n]在 0 ≤ n < N 范围内定义。

适应：

$h_m[n] = h_m[n - 1] + nu * (r[n] - y[n]) * x[n - m]$，其中 0 ≤ n < N，0 ≤ m < M。

其中，

$$nu = \frac{mu}{mu + E[n]}$$

其中，

$E[n] = E[n - 1] + (x[n])^2 - (x[n - M - 1])^2$，为输入信号能量估计。

启动时，*energyEstimate* 应初始化为 E[-1]的值（首次调用滤波器时为零）。返回后，*energyEstimate* 更新为值 E[N – 1]（如果需要继续对输入信号进行滤波，该值可用作后续函数调用的启动值）。

如果*(r[n] – y[n])*的绝对值大于或等于 1，则该运算可能导致饱和。

注：能量估计的另一种表达式为： $E[n] = (x[n])^2 + (x[n - 1])^2 + \cdots + (x[n - M + 2])^2$

因此，为避免计算估计值时发生饱和，输入采样值应满足以下条件：

$$\sum_{m=0}^{-M+2} (x[m+n])^2 < 1$$

不得将滤波器系数分配到程序存储器中，否则可能无法调整其值。如果检测到滤波器系数已分配到程序存储器中，则函数返回 NULL。

（另见 *FIRStruct*、*FIRStructInit* 和 *FIRInterpDelayInit*。）

对于 dsPIC33A 器件，将忽略 `coeffsPage` 的值。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `firlms.s`
- 对于 dsPIC33A
  `firlms_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	91	66 + N(49 + 5M)
dsPIC33E/33C	94	72 + N*(49 + 5M)
dsPIC33A	74	*见下表

dsPIC33A 的周期数：

源向量大小	系数位于 X 存储区中时的周期数
32	6360
64	12636
128	25176
256	50268
512	100440
1024	200796
2048	401496

*（*所有值的测量条件为 `numCoeffs = 32`）*

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W7*——已使用，未恢复
  - *W8...W13*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *MODCON*——已保存，已使用，已恢复
  - *XMODSTRT*——已保存，已使用，已恢复

  - *XMODEND*——已保存，已使用，已恢复
  - *YMODSTRT*——已保存，已使用，已恢复
  - *YMODEND*——已保存，已使用，已恢复
  - *PSVPAG/DSRPAG*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 两级 *DO* 指令
    - 一条 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W7*——已使用，未恢复
  - *W8...W13*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *MODCON*——已保存，已使用，已恢复
  - *XMODSTRT*——已保存，已使用，已恢复
  - *XMODEND*——已保存，已使用，已恢复
  - *YMODSTRT*——已保存，已使用，已恢复
  - *YMODEND*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——2 条

5.5.10. FIRStructInit

说明

`FIRStructInit` 用于初始化 *FIRStruct* FIR 滤波器结构中的参数值。

原型

```
void FIRStructInit (FIRStruct* filter, int numCoeffs, fractional* coeffsBase,
int coeffsPage, fractional* delayBase);
```

参数

参数	说明
filter	指向 *FIRStruct* 滤波器结构的指针。
numCoeffs	滤波器系数的数量（亦即 M）
coeffsBase	滤波器系数的基址（亦即 h）
coeffsPage	系数缓冲区页编号
delayBase	延时缓冲区的基址

返回

无。

备注

请参见上文 *FIRStruct* 结构的说明。

完成后，*FIRStructInit* 将相应地初始化 *coeffsEnd* 和 *delayEnd* 指针。此外，还会将延时设置为与 *delayBase* 相等。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `firinit.s`
- 对于 dsPIC33A
  `firinit_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	10	19
dsPIC33E/33C	16	28
dsPIC33A	8	26

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W5*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 无 *DO* 指令
    - 无 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W5*——已使用，未恢复
  - *REPEAT* 指令的使用——无

5.5.11. IIRCanonic

说明

`IIRCanonic` 用于通过级联规范（直接 II 型）双二次节将 IIR 滤波器应用于源采样序列。然后，将结果放入目标采样序列中并更新延时值。

原型

```
typedef struct {
 int numSectionsLess1;
 fractional* coeffsBase;
 int coeffsPage;
 fractional* delayBase;
 int initialGain;
 int finalShift;
} IIRCanonicStruct;
```

```
fractional* IIRCanonic (int numSamps, fractional* dstSamps,
fractional* srcSamps, IIRCanonicStruct* filter);
```

参数

滤波器结构：

参数	说明
numSectionsLess1	比级联二阶（双二次）节的数量少 1（亦即 S-1）
coeffsBase	指向滤波器系数（亦即{a, b}）的指针，位于 X 数据空间或程序存储器中
coeffsPage	系数缓冲区页编号，如果系数存储在数据空间中则为定义值 0xFF00（`COEFFS_IN_DATA` 的定义值）。
delayBase	指向滤波器延时（亦即 d）的指针，仅位于 Y 数据空间中
initialGain	初始增益值

IIRCanonic （续）	
参数	**说明**
finalShift	输出缩放（左移）

滤波器说明：

参数	说明
numSamps	待滤波的输入采样数（亦即 *N*；*N 是 R 的倍数*）
dstSamps	指向目标采样（亦即 *y*）的指针
srcSamps	指向源采样（亦即 *x*）的指针
filter	指向 IIRCanonicStruct 滤波器结构的指针

返回

指向目标采样的基址的指针。

备注

每个二阶（双二次）节有五个系数（由外部生成），按有序集排列——**{$a_2[s]$, $a_1[s]$, $a_0[s]$, $b_1[s]$, $b_0[s]$}，0 ≤ s < S。**

延时由每个节的两个滤波器状态字组成——{$d_1[s]$, $d_2[s]$}，0 ≤ s < S。

源采样 x[n]在 0 ≤ n < N 范围内定义。

目标采样 y[n]在 0 ≤ n < N 范围内定义。

每个输入采样先应用初始增益值再进入滤波器结构。

输出缩放以移位方式应用于滤波器结构的输出，随后将结果存入输出序列中。这用于将滤波器增益恢复到 0 dB。移位计数可能为零；如果不为零，则表示要移位的位数：负数表示左移，正数表示右移。

对于 dsPIC33E/33C：

当 *coeffsPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + *TABLE_SIZE* < *__YDATA_BASE*

其中，

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 *STACK_GUARD* 时必须小心。

- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 *STACK_GUARD* 的值。
- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 *STACK_GUARD* 的值。

对于 dsPIC33A 器件，将忽略 *coeffsPage* 的值。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `iircan.s`
- 对于 dsPIC33A
  `iircan_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	48	38 + N(8 + 7S)（如果系数位于数据存储器中） 41 + N(9 + 12S)（如果系数位于程序存储器中）
dsPIC33E/33C	101	44 + N*(9 + 7S)（如果系数位于数据存储器中） (91 + 2*S) + N*(9 + 7S)（如果系数位于程序存储器中但被复制到数据存储器中） 62 + N*(18 + 28S)（如果系数位于程序存储器中且未复制到数据存储器中）
dsPIC33A	30	*见下表

dsPIC33A 的周期数：

源向量大小	系数位于 X 存储区中时的周期数	系数位于 P 存储区中时的周期数
32	1920	3538
64	3818	7026
128	7594	14002
256	15146	27954
512	30250	55858
1024	60458	111666
2048	120874	223282

*（*所有值的测量条件为 S = 5）*

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W7*——已使用，未恢复
  - *W8...W14*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *PSVPAG/DSRPAG*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 两级 *DO* 指令
    - 一条 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W7*——已使用，未恢复
  - *W8...W9*——已保存，已使用，已恢复

- *ACCA*——已使用，未恢复
- *CORCON*——已保存，已使用，已恢复
- *REPEAT* 指令的使用——无

5.5.12. IIRCanonicInit

说明

`IIRCanonicInit` 用于将 *IIRCanonicStruct* 滤波器结构中的延时值初始化为零。

原型

```
void IIRCanonicInit(IIRCanonicStruct* filter);
```

参数

参数	说明
filter	指向 *IIRCanonicStruct* 滤波器结构的指针

返回

无。

备注

请参见上文 *IIRCanonic 函数*的说明。

每个二阶节具有两个滤波器状态字{$d_1[s]$, $d_2[s]$}，0 ≤ s < S。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `iircan.s`
- 对于 dsPIC33A
  `iircan_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	7	10 + 2*S
dsPIC33E/33C	10	22 + 2*S
dsPIC33A	6	28+S

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W1*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 一级 *DO* 指令
    - 无 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W2*——已使用，未恢复
  - *REPEAT* 指令的使用——1 条

5.5.13. IIRLattice

说明

IIRLattice 使用栅格结构实现将 IIR 滤波器应用于源采样序列。然后，将结果放入目标采样序列中并更新延时值。

原型

```
typedef struct {
 int order;
  fractional* kappaVals;
  fractional* gammaVals;
  int coeffsPage;
  fractional* delay;
} IIRLatticeStruct;
```

```
fractional* IIRLattice (int numSamps, fractional* dstSamps,
fractional* srcSamps, IIRLatticeStruct* filter);
```

参数

滤波器结构：

参数	说明
order	滤波器阶数（亦即 M，M ≤ N；关于 N，请参见 FIRLattice）
kappaVals	栅格系数（亦即 k）的基址，位于 X 数据空间或程序存储器中
gammaVals	梯形系数（亦即 g）的基址，位于 X 数据空间或程序存储器中。如果为 NULL，该函数将实现全极点滤波器。
CoeffsPage	系数缓冲区页编号，如果系数位于数据空间中则为 0xFF00（*COEFFS_IN_DATA* 的定义值）
delay	延时（亦即 d）的基址，仅位于 Y 数据空间中

滤波器说明：

参数	说明
numSamps	待滤波的输入采样数（亦即 *N；N 是 R 的倍数*）
dstSamps	指向目标采样（亦即 *y*）的指针
srcSamps	指向源采样（亦即 *x*）的指针
filter	指向 IIRLatticeStruct 滤波器结构的指针

返回

指向目标采样的基址的指针。

备注

栅格系数 k[m]在 0 ≤ m ≤ M 范围内定义。

梯形系数 g[m]在 0 ≤ m ≤ M 范围内定义（实现全极点滤波器时除外）。

延时 d[m]在 0 ≤ m ≤ M 范围内定义。

源采样 x[n]在 0 ≤ n < N 范围内定义。

目标采样 y[n]在 0 ≤ n < N 范围内定义。

注：该库提供的小数实现容易发生饱和。

如果在应用滤波器之前适当缩放输入信号，应该可以防止小数实现发生饱和。

对于 dsPIC33E/33C 器件：

当 *coeffsPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + *TABLE_SIZE* < *__YDATA_BASE*

其中，

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 *STACK_GUARD* 时必须小心。

- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 *STACK_GUARD* 的值。
- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 *STACK_GUARD* 的值。

对于 dsPIC33A 器件，将忽略 `coeffsPage` 的值。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `iirlatt.s`
- 对于 dsPIC33A
  `iirlatt_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	78	48 + N(16 + 7M)（如果系数位于数据存储器中） 51 + N(16 + 8M)（如果系数位于程序存储器中） M 为滤波器阶数。
dsPIC33E/33C	170	54 + N(18 + 7M)（如果系数位于数据存储器中） (101 + 2*M) + N*(18 + 7M)（如果系数位于程序存储器中但被复制到数据存储器中） 67 + N(30 + 11M)（如果系数位于程序存储器中且未复制到数据存储器中） M 为滤波器阶数。
dsPIC33A	31	*见下表

dsPIC33A 的周期数：

源向量大小	系数位于 X 存储区中时的周期数	系数位于 P 存储区中时的周期数
32	9586	15290
64	19122	30522
128	38194	60986
256	76338	121914

IIRLattice （续）		
源向量大小	系数位于 X 存储区中时的周期数	系数位于 P 存储区中时的周期数
512	152626	243770
1024	305202	487482
2048	610254	774906

*（*所有值的测量条件为 N = 32）*

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W7*——已使用，未恢复
  - *W8...W14*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *PSVPAG/DSRPAG*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 两级 *DO* 指令
    - 一条 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W7*——已使用，未恢复
  - *W8...W14*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——无

5.5.14. IIRLatticeInit

说明

`IIRLatticeInit` 用于将 *IIRLatticeStruct* 滤波器结构中的延时值初始化为零。

原型

```
void IIRLatticeInit(IIRLatticeStruct* filter);
```

参数

参数	说明
filter	指向 *IIRLatticeStruct* 滤波器结构的指针。

返回

无。

备注

请参见上文 *IIRLattice 函数*的说明。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `iirlatt.s`

- 对于 dsPIC33A
  `iirlatt_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	6	10 + M
dsPIC33E/33C	9	20 + M
dsPIC33A	5	24 + M

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W2*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 无 *DO* 指令
    - 一条 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W2*——已使用，未恢复
  - *REPEAT* 指令的使用——1 条

5.5.15. IIRTransposed

说明

`IIRTransposed` 用于通过级联转置（直接 II 型）双二次节将 IIR 滤波器应用于源采样序列。然后，将结果放入目标采样序列中并更新延时值。

原型

```
typedef struct {
 int numSectionsLess1;
 fractional* coeffsBase;
 int coeffsPage;
 fractional* delayBase1;
 fractional* delayBase2;
 int finalShift;
} IIRTransposedStruct;
```

```
fractional* IIRTransposed (int numSamps, fractional* dstSamps,
fractional* srcSamps, IIRTransposedStruct* filter);
```

参数

滤波器结构：

参数	说明
numSectionsLess1	比级联二阶（双二次）节的数量少 1（亦即 S-1）
coeffsBase	指向滤波器系数（亦即{a, b}）的指针，位于 X 数据空间或程序存储器中
coeffsPage	系数缓冲区页编号，如果系数位于数据空间中则为 0xFF00（*COEFFS_IN_DATA* 的定义值）
delayBase1	指向滤波器状态 1 的指针，每个二阶节对应 1 字延时（亦即 d_1），仅位于 Y 数据空间中
delayBase2	指向滤波器状态 2 的指针，每个二阶节对应 1 字延时（亦即 d_2），仅位于 Y 数据空间中
finalShift	输出缩放（左移）

滤波器说明：

参数	说明
numSamps	待滤波的输入采样数（亦即 *N*；*N 是 R 的倍数*）
dstSamps	指向目标采样（亦即 *y*）的指针
srcSamps	指向源采样（亦即 *x*）的指针
filter	指向 IIRTransposedStruct 滤波器结构的指针

返回

指向目标采样的基址的指针。

备注

每个二阶（双二次）节有五个系数（由外部生成），按有序集排列——**{$b_0[s]$, $b_1[s]$, $a_1[s]$, $b_2[s]$, $a_2[s]$}，0 ≤ s < S。**

延时由每个节的两个滤波器状态字组成——{$d_1[s]$, $d_2[s]$}，0 ≤ s < S。

源采样 x[n]在 0 ≤ n < N 范围内定义。

目标采样 y[n]在 0 ≤ n < N 范围内定义。

输出缩放以移位方式应用于滤波器结构的输出，随后将结果存入输出序列中。这用于将滤波器增益恢复到 0 dB。移位计数可能为零；如果不为零，则表示要移位的位数：负数表示左移，正数表示右移。

对于 dsPIC33E/33C 器件：

当 *coeffsPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + *TABLE_SIZE* < *__YDATA_BASE*

其中，

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 *STACK_GUARD* 时必须小心。

- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 *STACK_GUARD* 的值。
- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 *STACK_GUARD* 的值。

对于 dsPIC33A 器件，将忽略 `coeffsPage` 的值。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `iirtrans.s`
- 对于 dsPIC33A
  `iirtrans_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	54	47 + N*(4 + 7M)（如果系数位于数据存储器中） 50 + N*(4 + 8M)（如果系数位于程序存储器中）
dsPIC33E/33C	102	56 + N*(4 + 7M)（如果系数位于数据存储器中） (81 + M) + N*(4 + 7M)（如果系数位于程序存储器中但被复制到数据存储器中） 66 + N*(3 + 15M)（如果系数位于程序存储器中且未复制到数据存储器中）
dsPIC33A	32	*见下表

dsPIC33A 的周期数：

源向量大小	系数位于 X 存储区中时的周期数	系数位于 P 存储区中时的周期数
32	2508	4112
64	4972	8176
128	9900	16304
256	19756	32560
512	39468	65072
1024	78892	130096
2048	157740	260144

*（*所有值的测量条件为 S = 5）*

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W7*——已使用，未恢复
  - *W8...W14*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *PSVPAG/DSRPAG*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 两级 *DO* 指令
    - 一条 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W7*——已使用，未恢复
  - *W8...W10*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——无

5.5.16. IIRTransposedInit

说明

`IIRTransposedInit` 用于将 *IIRTransposedStruct* 滤波器结构中的延时值初始化为零。

原型

```
void IIRTransposedInit (IIRTransposedStruct* filter);
```

参数

参数	说明
filter	指向 *IIRTransposedStruct* 滤波器结构的指针。

返回

无。

备注

延时由两个独立缓冲区组成，每个缓冲区包含每个节的一个滤波器状态字——{$d_2[s]$, $d_1[s]$}，$0 \leqslant s < S$。

请参见上文 *IIRTransposed 函数*的说明。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `iirtrans.s`
- 对于 dsPIC33A
  `iirtrans_aa.s`

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	8	11 + 2*S
dsPIC33E/33C	11	21 + 2*S
dsPIC33A	8	18 + 2*S

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W2*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 一条 *DO* 指令
    - 无 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W3*——已使用，未恢复
  - *REPEAT* 指令的使用——2 条

6. 变换函数

函数	说明	目标器件
BitReverseComplex	就地将复数向量的元素按位反转顺序重新排列。	dsPIC30F/33F/33E/33C/33A
BitReverseReal32bIP	就地将 32 位实数向量的元素按位反转顺序重新排列。	dsPIC30F/33F/33E/33C
CosFactorInit	生成 II 型离散余弦变换所需的余弦因子集的前半部分并将结果放入目标复数向量中。	dsPIC30F/33F/33E/33C/33A
DCT	计算源向量的离散余弦变换并将结果存入目标向量中。	dsPIC30F/33F/33E/33C/33A
DCTIP	就地计算源向量的离散余弦变换。	dsPIC30F/33F/33E/33C/33A
FFTComplex	计算源复数向量的快速傅立叶变换并将结果存入目标复数向量中。	dsPIC30F/33F/33E/33C/33A
FFTComplexIP	就地计算源复数向量的快速傅立叶变换。	dsPIC30F/33F/33E/33C/33A
IFFTComplex	计算源复数向量的逆快速傅立叶变换并将结果存入目标复数向量中。	dsPIC30F/33F/33E/33C/33A
IFFTComplexIP	就地计算源复数向量的逆快速傅立叶变换。	dsPIC30F/33F/33E/33C/33A
FFTReal32b	计算 32 位源实数向量的 32 位快速傅立叶变换并将结果存入 32 位目标实数向量中。	dsPIC30F/33F/33E/33C
FFTReal32bIP	就地计算 32 位源实数向量的 32 位快速傅立叶变换。	dsPIC30F/33F/33E/33C
IFFTReal32b	计算 32 位源实数向量的 32 位逆快速傅立叶变换并将结果存入 32 位目标实数向量中。	dsPIC30F/33F/33E/33C
IFFTReal32bIP	就地计算 32 位源实数向量的 32 位逆快速傅立叶变换。	dsPIC30F/33F/33E/33C
FFTComplex32bIP	就地计算源复数向量的 32 位快速傅立叶变换。	dsPIC30F/33F/33E/33C
IFFTComplex32bIP	就地计算源复数向量的 32 位逆快速傅立叶变换。	dsPIC30F/33F/33E/33C
FFTRealIP	就地计算 32 位源实数向量的 32 位快速傅立叶变换。	dsPIC33A
FFTReal	计算 32 位源实数向量的快速傅立叶变换并将结果存入 32 位目标实数向量中。	dsPIC33A
IFFTRealIP	就地计算源复数向量（通过对实数向量使用 `FFTReal` 函数计算得出）的逆快速傅立叶变换。	dsPIC33A
IFFTReal	计算源复数向量（通过对实数向量使用 `FFTReal` 函数计算得出）的逆快速傅立叶变换。	dsPIC33A
SquareMagnitudeComplex	计算源复数向量中每个元素的平方幅值。	dsPIC30F/33F/33E/33C/33A
SquareMagnitudeComplex32bIP	计算 32 位源复数向量中每个元素的 32 位平方幅值。	dsPIC30F/33F/33E/33C
TwidFactorInit	生成离散傅立叶变换或离散余弦变换所需的旋转因子集的前半部分，并将结果放入目标复数向量中。	dsPIC30F/33F/33E/33C/33A

6.1. 小数变换运算

小数变换是一种线性、时不变、离散运算，当应用于小数时域采样序列时，会在频域中生成小数频率。反之，当将逆小数变换运算应用于频域数据时，会得到其时域表示。

DSP 库提供了一组变换（以及一组逆变换）。第一组将离散傅立叶变换（或其逆变换）应用于复数数据集（见小数复数向量）。第二组将 II 型离散余弦变换（Discrete Cosine Transform，DCT）应用于实数序列。这些变换支持非就地和就地两种运算类型。非就地类型使用变换的结果填充输出序列。就地类型将输入序列（物理上）替换为变换后的序列。对于非就地运算，必须提供足够的存储空间来容纳计算结果。

变换利用的因子（或常数）必须在初始化期间提供给变换函数。这些变换因子是复数数据集，以浮点算术进行计算，然后转换为小数以供运算使用。为了避免在应用变换时产生过多的计算开销，可以只生成一次

特定的变换因子集，并在程序执行期间多次使用。因此，建议将任何初始化运算返回的因子存入永久（静态）复数向量中。此外，还可以“离线”生成变换因子并将其放入程序存储器中以供稍后执行程序时使用。这样，在设计涉及变换的应用时，不仅可以减少周期数，还可以节省 RAM 存储空间。

6.2. 小数复数向量

复数数据向量由一个数据集表示，其中每对值代表向量的一个元素。每对值中的第一个值是元素的实部，第二个值是元素的虚部。实部和虚部均存储在存储器中，两者各占用一个字并且必须解释为 1.15/1.31 小数。与小数向量一样，小数复数向量将其元素连续存储在存储器中。

可通过以下数据结构实现小数复数向量中的数据组织：

```
#ifdef fractional
   #ifndef fractcomplex
       typedef struct {
          fractional real;
          fractional imag;
       } fractcomplex;
   #endif
#endif
```

6.3. 用户注意事项

使用变换函数时，请注意以下事项：

1. 这些函数不执行边界检查。当使用的源复数向量尺寸超限（包括长度为零的向量），以及使用的源复数向量和因子集不匹配时，可能会产生意外的结果。
2. 建议在完成每个函数调用之后检查状态寄存器（SR）。特别是，用户可在函数返回后检查 SA、SB 和 SAB 标志以判断是否发生饱和。
3. 变换系列涉及的输入和输出复数向量必须分配到 Y 数据存储区中。变换因子可分配到 X 数据空间或程序存储器中。
4. 由于位反转寻址要求向量集按模对齐，因此在显式或隐式使用 *BitReverseComplex* 函数的运算中，必须正确分配输入和输出复数向量。
5. 返回目标复数向量的运算可进行嵌套。例如，如果：
   a = Op1 (b, c)，其中 b = Op2 (d)且 c = Op3 (e, f)，则
   a = Op1 (Op2 (d), Op3 (e, f))。
6. dsPIC33A 的所有周期数值均在 PBU 高速缓存使能的情况下测得，实际值可能因 PBU 高速缓存状态或者向量与代码的存放位置而异。

6.4. 函数

6.4.1. BitReverseComplex

说明

BitReverseComplex 用于就地将复数向量的元素按位反转顺序重新排列。

原型

`fractcomplex* BitReverseComplex (int log2N, fractcomplex* srcCV);`

参数

参数	说明
log2N	以 2 为底 N 的对数（N = 源向量中复数元素的个数）
srcCV	指向源复数向量的指针

返回

指向源复数向量的基址的指针。

备注

N 必须是 2 的整数次幂。

必须按照模 N 对齐的方式分配 srcCV 向量。

该函数就地运算。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `bitrev.s`
- 对于 dsPIC33A
  `bitrev_aa.s`

函数配置文件

程序字数	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
	27	33	18

周期数	变换长度	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
	32	245	294	306
	64	485	566	594
	128	945	1098	1154
	256	1905	2186	2306
	512	3785	4338	4578
	1024	7625	8690	9186
	2048	15225	17346	18338

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0..W7*——已使用，未恢复
  - *XBREV*——已保存，已使用，已恢复
  - *MODCON*——已保存，已使用，已恢复
  - *DSRPAG*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 一级 *DO* 指令
- 对于 dsPIC33A
  - *W0..W6*——已使用，未恢复
  - *XBREV*——已保存，已使用，已恢复
  - *MODCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——无

6.4.2. BitReverseReal32bIP

说明

`BitReverseReal32bIP` 用于就地将 32 位实数向量的元素按位反转顺序重新排列。

原型

```
long* BitReverseReal32bIP (int log2N, long* srcV);
```

参数

参数	说明
log2N	以 2 为底 N 的对数（N = 32 位源实数向量的数量）
srcV	指向 32 位源实数向量的指针

返回

指向源实数向量的基址的指针。

备注

N 必须是 2 的整数次幂。

必须按照模 N 对齐的方式分配 srcV 向量。

该函数就地运算。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `rbrev32b.s`

函数配置文件

程序字数	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
	33	39	N/A

周期数	变换长度	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
	32	325	372	N/A
	64	677	756	
	128	1333	1484	
	256	2741	3020	
	512	5461	6012	
	1024	11093	12156	

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0..W7*——已使用，未恢复
  - *XBREV*——已保存，已使用，已恢复
  - *MODCON*——已保存，已使用，已恢复
  - *DSRPAG*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 一级 *DO* 指令

6.4.3. CosFactorInit

说明

CosFactorInit 用于生成 II 型离散余弦变换所需的余弦因子集的前半部分并将结果放入目标复数向量中。

实际上，旋转因子集包含以下值：

$$CN[k] = e^{\frac{k\pi i}{2N}},\ for\ 0 \le k < \frac{N}{2}$$

原型

```
fractcomplex* CosFactorInit (int log2N, fractcomplex* cosFactors);
```

参数

参数	说明
log2N	以 2 为底 N 的对数（N = DCT 所需的复数因子的数量）
cosFactors	指向复数余弦因子的指针

返回

指向余弦因子的基址的指针。

备注

N 必须是 2 的整数次幂。

仅生成前 N/2 个余弦因子。

在调用该函数之前，必须已分配一个大小为 N/2 的复数向量并将其指定给 cosFactors。复数向量应位于 X 数据存储区中。

因子以浮点算术进行计算并转换为 1.15/1.31 复数小数。

源文件

- 对于 dsPIC30F/33F/33C/33E——`initcosf.c`
- 对于 dsPIC33A——`initcosf_aa.c`

6.4.4. DCT

说明

`DCT` 用于计算源向量的离散余弦变换。

原型

```
fractcomplex* DCT (int log2N, fractional* dstCV, fractcomplex* srcCV,
fractcomplex* cosFactors, fractcomplex* twidFactors, int factPage);
```

参数

参数	说明
log2N	以 2 为底 N（源向量中复数元素的个数）的对数
dstCV	指向目标向量的指针
srcCV	指向源向量的指针
cosFactors	指向余弦因子的指针
twidFactors	指向旋转因子的指针
factPage	用于存储变换因子的存储器页

返回

指向目标采样的基址的指针。

备注

该函数在内部使用 *DCTIP* 和 *VectorZeroPad* 函数。

N 必须是 2 的整数次幂。

必须已分配一个大小为 2N 个元素的向量并将其指定给 *dstCV*。

必须将 *dstCV* 向量分配到 Y 数据空间中，并且地址满足模 N 对齐。

计算结果存入目标向量的前 N 个元素中。

为避免计算过程中发生饱和（溢出），源向量的值应在[-0.5, 0.5]范围内。

仅需要前 N/2 个余弦因子。

仅需要前 N/2 个旋转因子。

对于 dsPIC30F/33F/33E/33C：

如果变换因子存储在 X 数据空间中，*cosFactors* 和 *twidFactors* 指向因子实际分配到的地址。如果变换因子存储在程序存储器中，*cosFactors* 和 *twidFactors* 是因子分配到的地址相对于程序页边界的偏移量。该偏移量可以使用嵌入汇编运算符 *psvoffset()*进行计算。

如果变换因子存储在 X 数据空间中，*factPage* 必须设置为 0xFF00（COEFFS_IN_DATA 的定义值）。如果旋转因子存储在程序存储器中，*factPage* 为包含旋转因子的程序页编号。该程序页编号可以使用嵌入汇编运算符 *psvpage()*进行计算。

对于 dsPIC33E/33C，当 *factPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + *TABLE_SIZE* < *__YDATA_BASE*

其中：

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 STACK_GUARD 时必须小心。

- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 STACK_GUARD 的值。
- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 STACK_GUARD 的值。

由于 dsPIC33A 系列器件实现了非分页线性 RAM/程序存储空间，无论因子存储在程序存储器还是 RAM 中，*cosFactors* 和 *twidFactors* 参数都指向实际位置，因此将忽略 *fractpage* 参数。

必须在 *conjFlag* 设置为非零值的情况下对旋转因子进行初始化。

按因子 N 对输出进行缩放。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `dctoop.s`
- 对于 dsPIC33A
  `dctoop_aa.s`

函数配置文件

	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
程序字数	172	342	64
周期数	22	29	33

注：

1. 上述程序字数和周期数仅与 *DCT* 有关。但是，由于该函数本身利用 *DCTIP* 和 *VectorZeroPad*，因此还必须考虑 *DCTIP* 和 *VectorZeroPad* 的程序字数和周期数。
2. 在 *DCTIP* 和 *VectorZeroPad* 的说明中，报告的周期数包括四个周期的 C 函数调用开销。因此，从 *DCTIP* 和 *VectorZeroPad* 加到 *DCT* 的实际周期数比单独 *DCTIP/VectorZeroPad* 报告的周期数少 2x4 个。

系统资源的使用

以下系统资源的使用不包括 *DCTIP* 和 *VectorZeroPad*。

- 对于 dsPIC30F/33F/33E/33C/33A
  - *W0...W5*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用——无

6.4.5. DCTIP

说明

DCTIP 用于就地计算源向量的离散余弦变换。

原型

```
fractcomplex * DCTIP (int log2N, fractcomplex* srcCV, fractcomplex*
cosFactors, fractcomplex* twidFactors, int factPage);
```

参数

参数	说明
log2N	以 2 为底 N（源向量中复数元素的个数）的对数
srcCV	指向源向量的指针
cosFactors	指向余弦因子的指针
twidFactors	指向旋转因子的指针
factPage	用于存储变换因子的存储器页

返回

指向目标采样的基址的指针。

备注

N 必须是 2 的整数次幂。

该函数要求源向量已用零填充至 2N 长度。

必须将 *srcCV* 向量分配到 Y 数据空间中，并且地址满足模 N 对齐。

计算结果存入源向量的前 N 个元素中。

为避免计算过程中发生饱和（溢出），源向量的值应在[-0.5, 0.5]范围内。

仅需要前 N/2 个余弦因子。

仅需要前 N/2 个旋转因子。

对于 dsPIC30F/33F/33E/33C：

如果变换因子存储在 X 数据空间中，*cosFactors* 和 *twidFactors* 指向因子实际分配到的地址。如果变换因子存储在程序存储器中，*cosFactors* 和 *twidFactors* 是因子分配到的地址相对于程序页边界的偏移量。该偏移量可以使用嵌入汇编运算符 *psvoffset()*进行计算。

如果变换因子存储在 X 数据空间中，*factPage* 必须设置为 0xFF00（COEFFS_IN_DATA 的定义值）。如果旋转因子存储在程序存储器中，*factPage* 为包含旋转因子的程序页编号。该程序页编号可以使用嵌入汇编运算符 *psvpage()*进行计算。

对于 dsPIC33E/33C，当 *factPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + *TABLE_SIZE* < *__YDATA_BASE*

其中：

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 STACK_GUARD 时必须小心。

- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 STACK_GUARD 的值。
- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 STACK_GUARD 的值。

由于 dsPIC33A 系列器件实现了非分页线性 RAM/程序存储空间，无论因子存储在程序存储器还是 RAM 中，*cosFactors* 和 *twidFactors* 参数都指向实际位置，因此将忽略 *fractpage* 参数。

必须在 *conjFlag* 设置为非零值的情况下对旋转因子进行初始化。

按因子 N 对输出进行缩放。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `dct.s`
- 对于 dsPIC33A
  `dct_aa.s`

函数配置文件

程序字数	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
	172	342	64

	变换长度	dsPIC30F/33F	
		旋转因子位于 X 存储区中时的周期数	旋转因子位于 P 存储区中时的周期数
周期数	32	2,266	2,467
	64	4,904	5,361
	128	10,704	11,737
	256	23,402	25,715
	512	50,944	56,073
	1024	110,424	121,697
	2048	238,038	262,637

	变换长度	dsPIC33E/33C		
		旋转因子位于 X 存储区中时的周期数	旋转因子从 P 存储区复制到堆栈时的周期数	旋转因子位于 P 存储区中时的周期数
周期数	32	2,395	2,517	3,897
	64	5,123	5,309	8,609
	128	11,105	11,419	19,071
	256	24,157	24,727	42,107
	512	52,413	53,495	92,404
	1024	113,314	115,420	201,456
	2048	243,784	247,938	436,394

	变换长度	dsPIC33A	
		旋转因子位于 X 存储区中时的周期数	旋转因子位于 P 存储区中时的周期数
周期数	32	2,468	3,556
	64	5,350	7,926
	128	11,668	17,636
	256	25,478	39,062
	512	55,364	85,844
	1024	119,798	187,398
	2048	257,812	406,308

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W7*——已使用，未恢复
  - *W8...W14*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *PSVPAG/DSRPAG*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 一级 *DO* 指令
    - 一条 *REPEAT* 指令
- 对于 dsPIC33A

- *W0...W7*——已使用，未恢复
- *W8...W11*——已保存，已使用，已恢复
- *ACCA*——已使用，未恢复
- *CORCON*——已保存，已使用，已恢复
- *REPEAT* 指令的使用——1 条

6.4.6. FFTComplex

说明

`FFTComplex` 用于计算源复数向量的快速傅立叶变换。

原型

```
fractcomplex* FFTComplex(int log2N, fractcomplex* dstCV, fractcomplex* srcCV,
fractcomplex* twidFactors, int factPage);
```

参数

参数	说明
log2N	以 2 为底 N（源向量中复数元素的个数）的对数
dstCV	指向目标向量的指针
srcCV	指向源向量的指针
twidFactors	指向旋转因子的指针
factPage	用于存储旋转因子的存储器页

返回

指向目标采样的基址的指针。

备注

N 必须是 2 的整数次幂。

该函数在内部调用 *VectorCopy*、*FFTComplexIP* 和 *BitReversal* 函数。

该函数并非就地运算。必须已分配一个大小足够接收运算结果的复数向量并将其指定给 *dstCV*。

源复数向量中的元素应按自然顺序排列，并且目标向量中的最终变换结果也按自然顺序存储。

必须将 *dstCV* 向量分配到 Y 数据空间中，并且地址满足模 N 对齐。

为避免计算过程中发生饱和（溢出），源向量的值应在[-0.5, 0.5]范围内。

仅需要前 N/2 个旋转因子。

对于 dsPIC30F/33F/33E/33C：

如果变换因子存储在 X 数据空间中，*twidFactors* 指向因子实际分配到的地址。如果变换因子存储在程序存储器中，*twidFactors* 是因子分配到的地址相对于程序页边界的偏移量。该偏移量可以使用嵌入汇编运算符 *psvoffset()*进行计算。

如果变换因子存储在 X 数据空间中，*factPage* 必须设置为 0xFF00（COEFFS_IN_DATA 的定义值）。如果旋转因子存储在程序存储器中，*factPage* 为包含旋转因子的程序页编号。该程序页编号可以使用嵌入汇编运算符 *psvpage()*进行计算。

对于 dsPIC33E/33C，当 factPage 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + TABLE_SIZE < __YDATA_BASE

其中：

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 STACK_GUARD 时必须小心。

- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 STACK_GUARD 的值。
- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 STACK_GUARD 的值。

由于 dsPIC33A 系列器件实现了非分页线性 RAM/程序存储空间，无论因子存储在程序存储器还是 RAM 中，*twidFactors* 参数都指向实际位置，因此将忽略 *fractpage* 参数。

必须在 *conjFlag* 设置为零的情况下对旋转因子进行初始化。

按因子 N 对输出进行缩放。

有关 dsPIC DSC 的详细 FFT 实现指南，请参见技术简介 TB3141。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `fftoop.s`
- 对于 dsPIC33A
  `fftoop_aa.s`

函数配置文件

	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
程序字数	17	17	9
周期数	23	23	36

注：

1. 上述程序字数和周期数仅与 `FFTComplex` 有关。但是，由于该函数本身利用 `VectorCopy`、`FFTComplexIP` 和 `BitReversalComplex`，因此还必须考虑这三个函数的程序字数和周期数。
2. 在 `VectorCopy`、`FFTComplexIP` 和 `BitReversalComplex` 的说明中，报告的周期数包括四个周期的 C 函数调用开销。因此，从这三个函数加到 `FFTComplex` 的实际周期数比单独 `FFTComplex` 报告的周期数少 3x4 个。

系统资源的使用

以下系统资源的使用不包括 *VectorCopy*、*FFTComplexIP* 和 *BitReversalComplex*。

- 对于 dsPIC30F/33F/33E/33C/33A
  - *W0...W4*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用——无

6.4.7. FFTComplexIP

说明

`FFTComplexIP` 用于就地计算源复数向量的快速傅立叶变换。

原型

```
fractcomplex* FFTComplexIP (int log2N, fractcomplex* srcCV, fractcomplex*
twidFactors, int factPage);
```

参数

参数	说明
log2N	以 2 为底 N（源向量中复数元素的个数）的对数
srcCV	指向源向量的指针
twidFactors	指向旋转因子的指针
factPage	用于存储旋转因子的存储器页

返回

指向目标采样的基址的指针。

备注

N 必须是 2 的整数次幂。

源复数向量中的元素应按自然顺序排列。最终变换结果按位反转顺序存储。

必须将 *srcCV* 向量分配到 Y 数据空间中，并且地址满足模 N 对齐。

为避免计算过程中发生饱和（溢出），源向量的值应在[-0.5, 0.5]范围内。

仅需要前 N/2 个旋转因子。

对于 dsPIC30F/33F/33E/33C：

如果变换因子存储在 X 数据空间中，*twidFactors* 指向因子实际分配到的地址。如果变换因子存储在程序存储器中，*twidFactors* 是因子分配到的地址相对于程序页边界的偏移量。该偏移量可以使用嵌入汇编运算符 *psvoffset()*进行计算。

如果变换因子存储在 X 数据空间中，*factPage* 必须设置为 0xFF00（COEFFS_IN_DATA 的定义值）。如果旋转因子存储在程序存储器中，*factPage* 为包含旋转因子的程序页编号。该程序页编号可以使用嵌入汇编运算符 *psvpage()*进行计算。

对于 dsPIC33E/33C，当 *factPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + *TABLE_SIZE* < *__YDATA_BASE*

其中：

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 STACK_GUARD 时必须小心。

- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 STACK_GUARD 的值。
- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 STACK_GUARD 的值。

由于 dsPIC33A 系列器件实现了非分页线性 RAM/程序存储空间，无论因子存储在程序存储器还是 RAM 中，*twidFactors* 参数都指向实际位置，因此将忽略 *fractpage* 参数。

必须在 *conjFlag* 设置为零的情况下对旋转因子进行初始化。

按因子 N 对输出进行缩放。

有关 dsPIC DSC 的详细 FFT 实现指南，请参见技术简介 TB3141。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `fft.s`
- 对于 dsPIC33A
  `fft_aa.s`

函数配置文件

程序字数	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
	65	131	46

	变换长度（N）	dsPIC30F/33F	
		旋转因子位于 X 存储区中时的周期数	旋转因子位于 P 存储区中时的周期数
周期数	32	1,669	1,835
	64	3,807	4,197
	128	8,617	9,519
	256	19,315	21,369
	512	42,877	47,491
	1024	94,357	104,603
	2048	206,011	228,544

	变换长度（N）	dsPIC33E/33C		
		旋转因子位于 X 存储区中时的周期数	旋转因子从 P 存储区复制到堆栈时的周期数	旋转因子位于 P 存储区中时的周期数
周期数	32	1,717	1,779	3,011
	64	3,889	3,983	6,975
	128	8,765	8,923	15,947
	256	19,593	19,879	35,991
	512	43,413	43,955	80,316
	1024	95,418	96,472	177,374
	2048	208,120	210,198	388,407

	变换长度（N）	dsPIC33A	
		旋转因子位于 X 存储区中时的周期数	旋转因子位于 P 存储区中时的周期数
周期数	32	1,780	2,318
	64	4,060	5,356
	128	9,188	12,228
	256	20,588	27,564
	512	45,684	61,428
	1024	100,476	135,548
	2048	219,268	296,580

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W7*——已使用，未恢复
  - *W8...W14*——已保存，已使用，已恢复
  - *ACCA/ACCB*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *PSVPAG/DSRPAG*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用
    - 一级 *DO* 指令
    - 一条 *REPEAT* 指令
- 对于 dsPIC33A
  - *W0...W7*——已使用，未恢复
  - *W8...W14*——已保存，已使用，已恢复
  - *ACCA/ACCB*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——1 条

6.4.8. IFFTComplex

说明

`IFFTComplex` 用于计算源复数向量的逆快速傅立叶变换。

原型

```
fractcomplex* IFFTComplex(int log2N, fractcomplex* dstCV, fractcomplex*
srcCV, fractcomplex* twidFactors, int factPage);
```

参数

参数	说明
`log2N`	以 2 为底 N（源向量中复数元素的个数）的对数
`dstCV`	指向目标向量的指针
`srcCV`	指向源向量的指针
`twidFactors`	指向旋转因子的指针
`factPage`	用于存储旋转因子的存储器页

返回

指向目标采样的基址的指针。

备注

N 必须是 2 的整数次幂。

该函数在内部调用 *VectorCopy* 和 *IFFTComplexIP* 函数。

该函数并非就地运算。必须已分配一个大小足够接收运算结果的复数向量并将其指定给 *dstCV*。

源数组应按自然顺序排列。类似地，结果数组也将按照自然顺序存储。

必须将 *dstCV* 向量分配到 Y 数据空间中，并且地址满足模 N 对齐。

为避免计算过程中发生饱和（溢出），源向量的值应在[-0.5, 0.5]范围内。

仅需要前 N/2 个旋转因子。

对于 dsPIC30F/33F/33E/33C：

如果变换因子存储在 X 数据空间中，*twidFactors* 指向因子实际分配到的地址。如果变换因子存储在程序存储器中，*twidFactors* 是因子分配到的地址相对于程序页边界的偏移量。该偏移量可以使用嵌入汇编运算符 *psvoffset()*进行计算。

如果变换因子存储在 X 数据空间中，*factPage* 必须设置为 0xFF00（COEFFS_IN_DATA 的定义值）。如果旋转因子存储在程序存储器中，*factPage* 为包含旋转因子的程序页编号。该程序页编号可以使用嵌入汇编运算符 *psvpage()*进行计算。

对于 dsPIC33E/33C，当 *factPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + *TABLE_SIZE* < *__YDATA_BASE*

其中，

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 STACK_GUARD 时必须小心。

- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 STACK_GUARD 的值。
- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 STACK_GUARD 的值。

由于 dsPIC33A 系列器件实现了非分页线性 RAM/程序存储空间，无论因子存储在程序存储器还是 RAM 中，*twidFactors* 参数都指向实际位置，因此将忽略 *fractpage* 参数。

必须在 *conjFlag* 设置为非零值的情况下对旋转因子进行初始化。

按因子 N 对输出进行缩放。

有关 dsPIC DSC 的详细 FFT 实现指南，请参见技术简介 TB3141。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `ifftoop.s`
- 对于 dsPIC33A
  `ifftoop_aa.s`

函数配置文件

	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
程序字数	12	12	10
周期数	15	24	30

注：

1. 上述程序字数和周期数仅与 *IFFTComplex* 有关。但是，由于该函数本身利用 *VectorCopy* 和 *IFFTComplexIP*，因此还必须考虑这两个函数的程序字数和周期数。
2. 在 *VectorCopy* 和 *IFFTComplexIP* 的说明中，报告的周期数包括四个周期的 C 函数调用开销。因此，从这两个函数加到 *IFFTComplex* 的实际周期数比单独 *IFFTComplex* 报告的周期数少 2x4 个。

系统资源的使用

以下系统资源的使用不包括 *VectorCopy* 和 *IFFTComplexIP*。

- 对于 dsPIC30F/33F/33E/33C/33A
  - *W0...W4*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用——无

6.4.9. IFFTComplexIP

说明

`IFFTComplexIP` 用于就地计算源复数向量的逆快速傅立叶变换。

原型

```
fractcomplex* IFFTComplexIP(int log2N, fractcomplex* srcV, fractcomplex*
twidFactors, int factPage);
```

参数

参数	说明
log2N	以 2 为底 N（源向量中复数元素的个数）的对数
srcV	指向源向量的指针
twidFactors	指向旋转因子的指针
factPage	用于存储旋转因子的存储器页

返回

指向目标采样的基址的指针。

备注

N 必须是 2 的整数次幂。

该函数在内部调用 *BitReversalComplex* 和 *FFTComplexIP* 函数。

该函数就地运算。

源复数向量中的元素应按自然顺序排列，并且最终变换结果也将按自然顺序存储。

必须将 *dstV* 向量分配到 Y 数据空间中，并且地址满足模 N 对齐。

为避免计算过程中发生饱和（溢出），源向量的值应在[-0.5, 0.5]范围内。

仅需要前 N/2 个旋转因子。

对于 dsPIC30F/33F/33E/33C：

如果变换因子存储在 X 数据空间中，*twidFactors* 指向因子实际分配到的地址。如果变换因子存储在程序存储器中，*twidFactors* 是因子分配到的地址相对于程序页边界的偏移量。该偏移量可以使用嵌入汇编运算符 *psvoffset()*进行计算。

如果变换因子存储在 X 数据空间中，*factPage* 必须设置为 0xFF00（COEFFS_IN_DATA 的定义值）。如果旋转因子存储在程序存储器中，*factPage* 为包含旋转因子的程序页编号。该程序页编号可以使用嵌入汇编运算符 *psvpage()*进行计算。

对于 dsPIC33E/33C，当 *factPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + *TABLE_SIZE* < *__YDATA_BASE*

其中：

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 STACK_GUARD 时必须小心。

- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 STACK_GUARD 的值。
- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 STACK_GUARD 的值。

由于 dsPIC33A 系列器件实现了非分页线性 RAM/程序存储空间，无论因子存储在程序存储器还是 RAM 中，*twidFactors* 参数都指向实际位置，因此将忽略 *fractpage* 参数。

必须在 *conjFlag* 设置为非零值的情况下对旋转因子进行初始化。

按因子 N 对输出进行缩放。

有关 dsPIC DSC 的详细 FFT 实现指南，请参见技术简介 TB3141。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `ifft.s`
- 对于 dsPIC33A
  `ifft_aa.s`

函数配置文件

	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
程序字数	11	11	6

周期数	15	20	27

注：

1. 上述程序字数和周期数仅与 *IFFTComplexIP* 有关。但是，由于该函数本身利用 *BitReversalComplex* 和 *FFTComplexIP*，因此还必须考虑这两个函数的程序字数和周期数。
2. 在 *BitReversalComplex* 和 *FFTComplexIP* 的说明中，报告的周期数包括四个周期的 C 函数调用开销。因此，从这两个函数加到 *IFFTComplexIP* 的实际周期数比单独 *IFFTComplexIP* 报告的周期数少 2x4 个。

系统资源的使用

以下系统资源的使用不包括 *BitReversalComplex* 和 *FFTComplexIP*。

- 对于 dsPIC30F/33F/33E/33C/33A
  - *W0...W1*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用——无

6.4.10. FFTReal32b

说明

FFTReal32b 用于计算 32 位源实数向量的 32 位快速傅立叶变换并将结果存入 32 位实数目标向量中。该算法基于使用 N 点复数 FFT 与称为拆分函数的附加计算对 2N 点实数向量进行的高效 FFT 计算。

原型

```
long* FFTReal32b (int log2N-1, int N, long* dstCV, long* srcCV,
long* twidFactors, int factPage);
```

参数

参数	说明
log2N-1	以 2 为底 N 的对数减 1（log_2(N) - 1）
N	源实数向量中 32 位元素的个数
dstV	指向目标向量的指针
srcV	指向源向量的指针
twidFactors	指向旋转因子的指针
factPage	用于存储旋转因子的存储器页

返回

指向 32 位目标采样的基址的指针。

备注

N 必须是 2 的整数次幂。

该函数并非就地运算。必须已分配一个大小足够接收运算结果的 32 位实数向量并将其指定给 dstCV。

32 位源实数向量中的元素应按自然顺序排列。

32 位目标实数向量中的元素按自然顺序生成。

为避免计算过程中发生饱和（溢出），32 位源向量的幅值应在[-0.5, 0.5]范围内。

仅需要前 N/2 个旋转因子。

如果旋转因子存储在 X 数据空间中，*twidFactors* 指向因子实际分配到的地址。如果旋转因子存储在程序存储器中，*twidFactors* 是因子分配到的地址相对于程序页边界的偏移量。该偏移量可以使用嵌入汇编运算符 *psvoffset()*进行计算。

如果旋转因子存储在 X 数据空间中，*factPage* 必须设置为 0xFF00（COEFFS_IN_DATA 的定义值）。如果旋转因子存储在程序存储器中，*factPage* 为包含旋转因子的程序页编号。该程序页编号可以使用嵌入汇编运算符 *psvpage()*进行计算。

可以通过“*c:\Program Files\Microchip\xc-dsc\3.xy\support\generic\h*”路径下的 *dsp_factors_32b.h* 导入 32 位 FFT/iFFT 的旋转因子。

对于 dsPIC33E/33C，当 *factPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + *TABLE_SIZE* < *__YDATA_BASE*

其中：

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 STACK_GUARD 时必须小心。

- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 STACK_GUARD 的值。
- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 STACK_GUARD 的值。

按因子 N 对输出进行缩放。

注：

1. 该函数目前仅支持对最大大小为 1024 的源向量进行运算。
2. 该函数仅适用于 dsPIC30F/33F/33E/33C 系列器件。有关 dsPIC33A 器件中的等效功能，请参见 FFTReal 函数。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `fft32oop.c`

函数配置文件

	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
程序字数	38	38	N/A
周期数	11	13	N/A

注：

1. 上述程序字数和周期数仅与 *FFTReal32b* 有关。但是，由于该函数本身利用 *VectorCopy* 和 *FFTReal32IP*，因此还必须考虑这两个函数的程序字数和周期数。
2. 在 *VectorCopy* 和 *FFTReal32b* 的说明中，报告的周期数包括四个周期的 C 函数调用开销。因此，从这两个函数加到 *FFTReal32b* 的实际周期数比单独 *FFTReal32b* 报告的周期数少 2x4 个。

6.4.11. FFTReal32bIP

说明

FFTReal32bIP 用于就地计算 32 位源实数向量的 32 位快速傅立叶变换。该算法基于使用 N 点复数 FFT 与称为拆分函数的附加计算对 2N 点实数向量进行的高效 FFT 计算。

原型

```
long* FFTReal32b (int log2N-1, int N, long* srcCV, long* twidFactors,
int factPage);
```

参数

参数	说明
log2N-1	以 2 为底 N 的对数减 1（log_2(N) - 1）
N	源实数向量中 32 位元素的个数
srcV	指向源向量的指针
twidFactors	指向旋转因子的指针
factPage	用于存储旋转因子的存储器页

返回

指向 32 位目标采样的基址的指针。

备注

N 必须是 2 的整数次幂。

该函数就地运算。

32 位源实数向量中的元素应按自然顺序排列。

32 位目标实数向量中的元素按自然顺序生成。

为避免计算过程中发生饱和（溢出），32 位源向量的幅值应在[-0.5, 0.5]范围内。

仅需要前 N/2 个旋转因子。

如果旋转因子存储在 X 数据空间中，*twidFactors* 指向因子实际分配到的地址。如果旋转因子存储在程序存储器中，*twidFactors* 是因子分配到的地址相对于程序页边界的偏移量。该偏移量可以使用嵌入汇编运算符 *psvoffset()*进行计算。

如果旋转因子存储在 X 数据空间中，*factPage* 必须设置为 0xFF00（COEFFS_IN_DATA 的定义值）。如果旋转因子存储在程序存储器中，*factPage* 为包含旋转因子的程序页编号。该程序页编号可以使用嵌入汇编运算符 *psvpage()*进行计算。

可以通过“*c:\Program Files\Microchip\xc-dsc\3.xy\support\generic\h*”路径下的 *dsp_factors_32b.h* 导入 32 位 FFT/iFFT 的旋转因子。

对于 dsPIC33E/33C，当 *factPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + TABLE_SIZE + STACK_GUARD < SPLIM 且

SP + TABLE_SIZE < __YDATA_BASE

其中：

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 STACK_GUARD 时必须小心。

- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 STACK_GUARD 的值。
- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 STACK_GUARD 的值。

按因子 N 对输出进行缩放。

注：

1. 该函数目前仅支持对最大大小为 1024 的源向量进行运算。
2. 该函数仅适用于 dsPIC30F/33F/33E/33C 系列器件。有关 dsPIC33A 器件中的等效功能，请参见 FFTReal 函数。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `fft32.c`

函数配置文件

	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
程序字数	23	23	N/A

注：

1. 上述程序字数仅与 *FFTReal32b* 有关。但是，由于该函数本身利用*位反转函数*、*N 点复数 FFT 函数*和*拆分函数*，因此还必须考虑这三个函数的程序字数。

6.4.12. IFFTReal32b

说明

`IFFTReal32b` 用于计算 32 位源复数向量（通过对实数向量使用 FFTReal32b 计算得出）的 32 位逆快速傅立叶变换，并将结果存入 32 位目标实数向量中。该算法基于使用 N 点复数 IFFT 与称为解拆分函数的附加计算对 2N 点实数向量进行的高效 IFFT 计算。

原型

```
long* IFFTReal32b (int log2N-1, int N, long* dstCV, long* srcCV,
long* twidFactors, int factPage);
```

参数

参数	说明
log2N-1	以 2 为底 N 的对数减 1（log_2(N) - 1）
N	源向量中 32 位元素的个数

IFFTReal32b （续）	
参数	**说明**
dstV	指向目标实数向量的指针
srcV	指向源复数向量的指针
twidFactors	指向旋转因子的指针
factPage	用于存储旋转因子的存储器页

返回

指向 32 位目标采样的基址的指针。

备注

N 必须是 2 的整数次幂。

该函数并非就地运算。必须已分配一个大小足够接收运算结果的 32 位实数向量并将其指定给 dstCV。

32 位源复数向量（FFTReal32b 的输出）中的元素应按自然顺序排列。

32 位目标实数向量中的元素按自然顺序生成。

为避免计算过程中发生饱和（溢出），32 位源向量的幅值应在[-0.5, 0.5]范围内。

如果旋转因子存储在 X 数据空间中，*twidFactors* 指向因子实际分配到的地址。如果旋转因子存储在程序存储器中，*twidFactors* 是因子分配到的地址相对于程序页边界的偏移量。该偏移量可以使用嵌入汇编运算符 *psvoffset()*进行计算。

如果旋转因子存储在 X 数据空间中，*factPage* 必须设置为 0xFF00（COEFFS_IN_DATA 的定义值）。如果旋转因子存储在程序存储器中，*factPage* 为包含旋转因子的程序页编号。该程序页编号可以使用嵌入汇编运算符 *psvpage()*进行计算。

可以通过“*c:\Program Files\Microchip\xc-dsc\3.xy\support\generic\h*”路径下的 *dsp_factors_32b.h* 导入 32 位 FFT/iFFT 的旋转因子。

对于 dsPIC33E/33C，当 *factPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + *TABLE_SIZE* < *__YDATA_BASE*

其中：

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 STACK_GUARD 时必须小心。

- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 STACK_GUARD 的值。
- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 STACK_GUARD 的值。

按因子 N 对输出进行缩放。

注：

1. 该函数目前仅支持对最大大小为 1024 的源向量进行运算。
2. 该函数仅适用于 dsPIC30F/33F/33E/33C 系列器件。有关 dsPIC33A 器件中的等效功能，请参见 IFFTReal 函数。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `ifft32oop.c`

函数配置文件

	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
程序字数	38	38	N/A
周期数	11	13	N/A

注：

1. 上述程序字数和周期数仅与 *IFFTReal32b* 有关。但是，由于该函数本身利用 *VectorCopy* 和 *IFFTReal32IP*，因此还必须考虑这两个函数的程序字数和周期数。
2. 在 *VectorCopy* 和 *IFFTReal32b* 的说明中，报告的周期数包括四个周期的 C 函数调用开销。因此，从这两个函数加到 *IFFTReal32b* 的实际周期数比单独 *IFFTReal32b* 报告的周期数少 2x4 个。

6.4.13. IFFTReal32bIP

说明

`IFFTReal32bIP` 用于就地计算 32 位源复数向量（通过对实数向量使用 FFTReal32b 计算得出）的 32 位逆快速傅立叶变换并存储结果。该算法基于使用 N 点复数 IFFT 与称为解拆分函数的附加计算对 2N 点实数向量进行的高效 IFFT 计算。

原型

```
long* IFFTReal32b (int log2N-1, int N, long* srcCV, long* twidFactors,
int factPage);
```

参数

参数	说明
log2N-1	以 2 为底 N 的对数减 1（log_2(N) - 1）
N	源实数向量中 32 位元素的个数
srcV	指向源复数向量的指针
twidFactors	指向旋转因子的指针
factPage	用于存储旋转因子的存储器页

返回

指向 32 位目标采样的基址的指针。

备注

N 必须是 2 的整数次幂。

该函数就地运算。

32 位源复数向量（FFTReal32b 的输出）中的元素应按自然顺序排列。

32 位实数向量中的元素按自然顺序生成。

为避免计算过程中发生饱和（溢出），32 位源向量的幅值应在[-0.5, 0.5]范围内。

如果旋转因子存储在 X 数据空间中，*twidFactors* 指向因子实际分配到的地址。如果旋转因子存储在程序存储器中，*twidFactors* 是因子分配到的地址相对于程序页边界的偏移量。该偏移量可以使用嵌入汇编运算符 *psvoffset()*进行计算。

如果旋转因子存储在 X 数据空间中，*factPage* 必须设置为 0xFF00（COEFFS_IN_DATA 的定义值）。如果旋转因子存储在程序存储器中，*factPage* 为包含旋转因子的程序页编号。该程序页编号可以使用嵌入汇编运算符 *psvpage()*进行计算。

可以通过“*c:\Program Files\Microchip\xc-dsc\3.xy\support\generic\h*”路径下的 *dsp_factors_32b.h* 导入 32 位 FFT/iFFT 的旋转因子。

对于 dsPIC33E/33C，当 *factPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + *TABLE_SIZE* < *__YDATA_BASE*

其中：

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 STACK_GUARD 时必须小心。

- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 STACK_GUARD 的值。
- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 STACK_GUARD 的值。

按因子 N 对输出进行缩放。

注：

1. 该函数目前仅支持对最大大小为 1024 的源向量进行运算。
2. 该函数仅适用于 dsPIC30F/33F/33E/33C 系列器件。有关 dsPIC33A 器件中的等效功能，请参见 IFFTRealIP 函数。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `ifft32.c`

函数配置文件

	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
程序字数	23	23	N/A

注：

1. 上述程序字数仅与 *`IFFTReal32b`* 有关。但是，由于该函数本身利用 *位反转函数*、*N 点复数 `IFFT` 函数* 和 *解拆分函数*，因此还必须考虑这三个函数的程序字数。

6.4.14. FFTComplex32bIP

说明

`FFTComplex32bIP` 用于就地计算 32 位源复数向量的快速傅立叶变换并存储结果。

原型

```
long* FFTComplex32bIP (int log2N-1, int N, long* srcCV, long* twidFactors,
int factPage);
```

参数

参数	说明
log2N-1	以 2 为底 N 的对数减 1（log_2(N) - 1）
N	源复数向量中 32 位元素的个数
srcV	指向源向量的指针
twidFactors	指向旋转因子的指针
factPage	用于存储旋转因子的存储器页

返回

指向 32 位目标采样的基址的指针。

备注

N 必须是 2 的整数次幂。

源复数向量中的元素应按自然顺序排列。最终变换结果按位反转顺序存储。

为避免计算过程中发生饱和（溢出），源复数向量的幅值应在[-0.5, 0.5]范围内。

仅需要前 N/2 个旋转因子。

如果旋转因子存储在 X 数据空间中，*twidFactors* 指向因子实际分配到的地址。如果旋转因子存储在程序存储器中，*twidFactors* 是因子分配到的地址相对于程序页边界的偏移量。该偏移量可以使用嵌入汇编运算符 *psvoffset()*进行计算。

如果旋转因子存储在 X 数据空间中，*factPage* 必须设置为 0xFF00（COEFFS_IN_DATA 的定义值）。如果旋转因子存储在程序存储器中，*factPage* 为包含旋转因子的程序页编号。该程序页编号可以使用嵌入汇编运算符 *psvpage()*进行计算。

可以通过“*c:\Program Files\Microchip\xc-dsc\3.xy\support\generic\h*”路径下的 *dsp_factors_32b.h* 导入 32 位 FFT/iFFT 的旋转因子。

对于 dsPIC33E/33C，当 *factPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + *TABLE_SIZE* < *__YDATA_BASE*

其中：

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 STACK_GUARD 时必须小心。

- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 STACK_GUARD 的值。
- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 STACK_GUARD 的值。

按因子 N 对输出进行缩放。

注：

1. 该函数目前仅支持对最大大小为 1024 的源向量进行运算。
2. 该函数仅适用于 dsPIC30F/33F/33E/33C 系列器件。该函数的功能在 dsPIC33A 器件中通过 *FFTComplex* 函数实现。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `cplxFft32b.s`

系统资源

- *W0...W7*——已使用，未恢复
- *W8...W14*——已保存，已使用，已恢复
- *ACCA/ACCB*——已使用，未恢复
- *CORCON*——已保存，已使用，已恢复
- *DSRPAG/PSVPAG*——已保存，已使用，已恢复
- *DO* 和 *REPEAT* 指令
  - 两级 *DO* 指令

6.4.15. IFFTComplex32bIP

说明

`IFFTComplex32bIP` 用于就地计算 32 位源复数向量的逆快速傅立叶变换并存储结果。

原型

```
long* IFFTComplex32bIP (int log2N-1, int N, long* srcCV, long* twidFactors,
int factPage);
```

参数

参数	说明
log2N-1	以 2 为底 N 的对数减 1（log_2(N) - 1）
N	源复数向量中 32 位元素的个数
srcV	指向源向量的指针
twidFactors	指向旋转因子的指针
factPage	用于存储旋转因子的存储器页

返回

指向 32 位目标采样的基址的指针。

备注

N 必须是 2 的整数次幂。

源复数向量中的元素应按自然顺序排列。最终变换结果按位反转顺序存储。

为避免计算过程中发生饱和（溢出），源复数向量的幅值应在[-0.5, 0.5]范围内。

仅需要前 N/2 个旋转因子。

如果旋转因子存储在 X 数据空间中，*twidFactors* 指向因子实际分配到的地址。如果旋转因子存储在程序存储器中，*twidFactors* 是因子分配到的地址相对于程序页边界的偏移量。该偏移量可以使用嵌入汇编运算符 *psvoffset()*进行计算。

如果旋转因子存储在 X 数据空间中，*factPage* 必须设置为 0xFF00（COEFFS_IN_DATA 的定义值）。如果旋转因子存储在程序存储器中，*factPage* 为包含旋转因子的程序页编号。该程序页编号可以使用嵌入汇编运算符 *psvpage()*进行计算。

可以通过“*c:\Program Files\Microchip\xc-dsc\3.xy\support\generic\h*”路径下的 *dsp_factors_32b.h* 导入 32 位 FFT/iFFT 的旋转因子。

对于 dsPIC33E/33C，当 *factPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + *TABLE_SIZE* + *STACK_GUARD* < *SPLIM* 且

SP + *TABLE_SIZE* < *__YDATA_BASE*

其中：

- *SP*——堆栈指针
- *TABLE_SIZE*——PSV 中系数向量的大小
- *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
- *SPLIM*——堆栈指针限制
- *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 STACK_GUARD 时必须小心。

- 值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 STACK_GUARD 的值。
- 值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 STACK_GUARD 的值。

按因子 N 对输出进行缩放。

注：

1. 该函数目前仅支持对最大大小为 1024 的源向量进行运算。
2. 该函数仅适用于 dsPIC30F/33F/33E/33C 系列器件。该函数的功能在 dsPIC33A 器件中通过 *FFTComplex* 函数实现。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `cplxiFft32b.s`

系统资源

- *W0...W7*——已使用，未恢复
- *W8...W14*——已保存，已使用，已恢复
- *ACCA/ACCB*——已使用，未恢复

- *CORCON*——已保存，已使用，已恢复
- *DSRPAG/PSVPAG*——已保存，已使用，已恢复
- *DO* 和 *REPEAT* 指令
  - 两级 *DO* 指令

6.4.16. FFTRealIP

说明

`FFTRealIP` 用于就地计算源实数向量的快速傅立叶变换。该算法基于使用 N 点复数 FFT 与称为拆分函数的附加计算对 2N 点实数向量进行的高效 FFT 计算。

原型

```
fractcomplex* FFTRealIP (int log2N, fractional* srcV, fractcomplex*
twidFactors);
```

参数

参数	说明
`log2N`	以 2 为底 N（源向量中复数元素的个数）的对数
`srcV`	指向源实数向量的指针
`twidFactors`	指向复数旋转因子的指针

返回

指向目标复数采样的基址的指针。

备注

N 必须是 2 的整数次幂。

源复数向量中的元素应按自然顺序排列，并且最终向量也按自然顺序返回。最终变换结果是大小为 N/2 + 1 的复数向量，存储位置与源向量相同。由于最终变换结果的后半部分为前半部分的共轭复数，因此仅返回 N/2 个复数元素。

该函数就地运算。*srcV* 必须是包含 N 个元素的非复数向量。必须为 *srcV* 额外分配 2 个字的空间，以存储目标复数向量中的第(N/2)个元素。

必须将 *srcV* 向量分配到 Y 数据空间中，并且地址满足模 N 对齐。

为避免计算过程中发生饱和（溢出），源向量的值应在[-0.5, 0.5]范围内。

仅需要前 N/2 个旋转因子。

该函数在内部利用 *BitReversal* 函数。

必须在 *conjFlag* 设置为零的情况下对旋转因子进行初始化。

按因子 N 对输出进行缩放。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `N/A`
- 对于 dsPIC33A
  `rfft_aa.s`

函数配置文件

程序字数	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
	N/A	N/A	115 + *BitReversalComplex* 函数的程序字数

周期数	变换长度（N）	dsPIC33A	
		旋转因子位于 X 存储区中时的周期数	旋转因子位于 P 存储区中时的周期数
	32	1,198	1,692
	64	2,551	3,726
	128	5,559	8,328
	256	12,122	18,587
	512	26,418	40,824
	1024	57,319	89,418
	2048	123,684	194,600

注：上述周期数值包括在内部使用的 *BitReversal* 函数的周期数。

系统资源的使用

- 对于 dsPIC33A
  - *W0...W7*——已使用，未恢复
  - *W8...W14*——已保存，已使用，已恢复
  - *ACCA/ACCB*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——1 条
  - 加上 *BitReversalComplex* 函数使用的资源

6.4.17. FFTReal

说明

`FFTReal` 用于非就地计算源实数向量的快速傅立叶变换。该算法基于使用 N 点复数 FFT 与称为拆分函数的附加计算对 2N 点实数向量进行的高效 FFT 计算。

原型

```
fractcomplex* FFTReal (int log2N, fractional* srcV, fractcomplex* dstCV,
fractcomplex* twidFactors);
```

参数

参数	说明
log2N	以 2 为底 N（源向量中复数元素的个数）的对数
srcV	指向源实数向量的指针。
dstCV	指向目标复数向量的指针。
twidFactors	指向复数旋转因子的指针。

返回

指向目标复数采样的基址的指针。

备注

N 必须是 2 的整数次幂。

srcV 必须是包含 N 个元素的非复数向量。

dstCV 必须是大小为 N/2 + 1 的复数向量。

源复数向量中的元素应按自然顺序排列，并且最终向量也按自然顺序返回。最终变换结果是大小为 N/2 + 1 的复数向量。由于最终变换结果的后半部分为前半部分的共轭复数，因此仅返回 N/2 个复数元素。

必须在 y 存储空间中按照模 N 对齐的方式分配 *srcV* 向量。

为避免计算过程中发生饱和（溢出），源向量的值应在[-0.5, 0.5]范围内。

仅需要前 N/2 个旋转因子。

该函数在内部利用 *VectorCopy* 和 *FFTRealIP* 函数。

必须在 *conjFlag* 设置为零的情况下对旋转因子进行初始化。

按因子 N 对输出进行缩放。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `N/A`
- 对于 dsPIC33A
  `rfft_aa.s`

函数配置文件

	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
程序字数	`N/A`	`N/A`	`10 + FFTRealIP` 和 `VectorCopy` 函数的程序字数。

	dsPIC33A
周期数	*VectorCopy* 的周期数 + *FFTRealIP* 的周期数 + 16

注：*VectorCopy* 和 *FFTRealIP* 的周期数（见各自对应的章节）包括函数调用和返回的开销。因此，在计算 *FFTReal* 函数的总周期数时，需从这两项中各减去约 4 个周期。

系统资源的使用

- 对于 dsPIC33A
  - *W0...W3*——已使用，未恢复
  - 加上 *VectorCopy* 和 *FFTRealIP* 函数使用的资源

6.4.18. IFFTRealIP

说明

`IFFTRealIP` 用于计算源复数向量（通过对实数向量使用 FFTReal 函数计算得出）的逆快速傅立叶变换。该算法基于使用 N 点复数 FFT 与称为拆分函数的附加计算对 2N 复数向量进行的高效 IFFT 计算。

原型

```
fractional* IFFTRealIP (int log2N, fractcomplex* srcCV, fractcomplex*
twidFactors);
```

参数

参数	说明
`log2N`	以 2 为底 N（源向量中复数元素的个数）的对数

IFFTRealIP （续）	
参数	**说明**
srcCV	指向源复数向量的指针。
twidFactors	指向复数旋转因子的指针。

返回

指向目标实数采样的基址的指针。

备注

N 必须是 2 的整数次幂。

复数 srcCV 向量的大小必须为 N/2 + 1，其中包含 0 到 N/2 个元素。

源复数向量中的元素应按自然顺序排列，并且最终向量也按自然顺序返回。最终变换结果是大小为 N 的实数向量，存储位置与源复数向量相同。

该函数就地运算。*srcV* 必须是包含 N/2 + 1 个元素的复数向量。最终向量将是大小为 *N* 的实数向量，存储位置与复数 *srcCV* 向量相同。

必须在 y 存储空间中按照模 N 对齐的方式分配 *dstCV* 向量。

为避免计算过程中发生饱和（溢出），源向量的值应在[-0.5, 0.5]范围内。

仅需要前 N/2 个旋转因子。

该函数在内部利用 *BitReversal* 函数和 *FFTReal* 函数的一部分。

必须在 *conjFlag* 设置为非零值的情况下对旋转因子进行初始化。

按因子 N 对输出进行缩放。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `N/A`
- 对于 dsPIC33A
  `irfft_aa.s`

函数配置文件

程序字数	**dsPIC30F/33F**	**dsPIC33E/33C**	**dsPIC33A**
	N/A	N/A	68 + *BitReversalComplex* 函数的程序字数

周期数	**变换长度（N）**	**dsPIC33A**	
		旋转因子位于 X 存储区中时的周期数	旋转因子位于 P 存储区中时的周期数
	32	1,229	1,712
	64	2,599	3,770
	128	5,629	8,390
	256	12,250	18,618
	512	26,768	41,160
	1024	57,970	90,095
	2048	125,050	195,958

注：上述周期数值包括在内部使用的 *BitReversalComplex* 函数的周期数。

系统资源的使用

- 对于 dsPIC33A
  - *W0...W7*——已使用，未恢复
  - *W8...W9*——已保存，已使用，已恢复
  - *ACCA/ACCB*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——1 条
  - 加上 *BitReversalComplex* 函数使用的资源。

6.4.19. IFFTReal

说明

`IFFTReal` 用于计算源复数向量（通过对实数向量使用 FFTReal 函数计算得出）的逆快速傅立叶变换。该算法基于使用 N 点复数 FFT 与称为拆分函数的附加计算对 2N 复数向量进行的高效 IFFT 计算。

原型

```
fractional * IFFTReal (int log2N, fractcomplex* srcCV, fractional* dstV,
fractcomplex* twidFactors);
```

参数

参数	说明
`log2N`	以 2 为底 N（源向量中复数元素的个数）的对数
`srcCV`	指向大小为 N/2 + 1 的源复数向量的指针。
`dstV`	指向大小为 N 的目标实数向量的指针。
`twidFactors`	指向复数旋转因子的指针。

返回

指向目标复数采样的基址的指针。

备注

N 必须是 2 的整数次幂。

srcCV 必须是包含 N/2 + 1 个元素的复数向量。

dstV 必须是大小为 N 的实数向量。

源复数向量中的元素应按自然顺序排列，并且最终向量也按自然顺序返回。最终变换结果是大小为 N 的实数向量。

必须在 y 存储空间中按照模 N 对齐的方式分配 *dstV* 向量。

为避免计算过程中发生饱和（溢出），源向量的值应在[-0.5, 0.5]范围内。

仅需要前 N/2 个旋转因子。

该函数在内部利用 *VectorCopy* 和 *IFFTRealIP* 函数。

必须在 *conjFlag* 设置为非零值的情况下对旋转因子进行初始化。

按因子 N 对输出进行缩放。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `N/A`
- 对于 dsPIC33A
  `irfft_aa.s`

函数配置文件

程序字数	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
	N/A	N/A	11

周期数	dsPIC33A
	VectorCopy 的周期数 + *IFFTRealIP* 的周期数 + 17

注：*VectorCopy* 和 *FFTRealIP* 的周期数（见各自对应的章节）包括函数调用和返回的开销。因此，在计算 *FFTReal* 函数的总周期数时，需从这两项中各减去约 4 个周期。

系统资源的使用

- 对于 dsPIC33A
  - *W0...W3*——已使用，未恢复
  - 加上 *VectorCopy* 和 *IFFTRealIP* 函数使用的资源。

6.4.20. SquareMagnitudeComplex

说明

`SquareMagnitudeCplx` 用于计算源复数向量中每个元素的平方幅值。

原型

```
fractional* SquareMagnitudeCplx (numElems, fractcomplex* srcV,
fractional* dstV);
```

参数

参数	说明
numElems	源向量中复数元素的个数
srcV	指向 fractcomplex 源向量的指针
dstV	指向小数目标向量的指针

返回

指向目标采样的基址的指针。

备注

如果源向量中复数元素实部和虚部的平方和大于所支持小数数据的最大值（$(1 - 2^{-15})$或$(1 - 2^{-31})$），则该运算会导致饱和。

该函数可用于对源数据集进行就地运算。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `cplxsqrmag.s`
- 对于 dsPIC33A
  `cplxsqrmag_aa.s`

函数配置文件

	dsPIC30F/33F	dsPIC33E/33C	dsPIC33A
程序字数	19	26	15
周期数	25 + 4(*numElems*)	34 + 4(*numElems*)	34 + 3.5(*numElems*)

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0*、*W1*、*W2*、*W4* 和 *W5*——已使用，未恢复
  - *W10*——已保存，已使用，已恢复
  - *ACCA*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用
  - • 一级 *DO* 指令
- 对于 dsPIC33A
  - *W0...W4*——已使用，未恢复
  - *W13*——已保存，已使用，已恢复
  - *ACCA/ACCB*——已使用，未恢复
  - *REPEAT* 指令的使用——无

6.4.21. SquareMagnitudeComplex32bIP

说明

`SquareMagnitudeCplx32bIP` 用于计算 32 位源复数向量中每个元素的 32 位平方幅值。

原型

```
long* MagnitudeCplx32bIP (int numElems, long* srcV);
```

参数

参数	说明
numElems	源向量中复数元素的个数。
srcV	指向 32 位源向量的指针

返回

无。

备注

如果源向量中复数元素实部和虚部的平方和大于 $1-2^{-31}$，则该运算会导致饱和。

该函数对源数据集进行就地运算。

注：该函数的功能在 dsPIC33A 中通过 *squareMagnitudeComplex* 函数实现。因此，该函数不适用于 dsPIC33A 系列器件。

源文件

- 对于 dsPIC30F/33F/33E/33C
  `cplxmag32b.s`

6.4.22. TwidFactorInit

说明

TwidFactorInit 用于生成离散傅立叶变换或离散余弦变换所需的旋转因子集的前半部分，并将结果放入目标复数向量中。

实际上，旋转因子集包含以下值：

- 当 conjFlag = 0 时：

$$w[k] = e^{\frac{-k \times 2\pi}{N}}, \quad for\ 0 \le k < \frac{N}{2}$$

- 当 conjFlag != 0 时：

$$w[k] = e^{\frac{k \times 2\pi}{N}}, \quad for\ 0 \le k < \frac{N}{2}$$

原型

```
fractcomplex* TwidFactorInit (int log2N, fractcomplex* twidFactors, int
conjFlag);
```

参数

参数	说明
log2N	以 2 为底 N 的对数（N = FFT 所需的复数因子的数量）
twidFactors	指向复数旋转因子的指针
conjFlag	指示是否生成共轭值的标志

返回

指向旋转因子的基址的指针。

备注

N 必须是 2 的整数次幂。

仅生成前 N/2 个旋转因子。

conjFlag 的值决定指数函数的参数的符号。对于正傅立叶变换，*conjFlag* 应设置为 0。对于逆傅立叶变换和离散余弦变换，*conjFlag* 应设置为 1。

在调用该函数之前，必须已分配一个大小为 N/2 的复数向量并将其指定给 *twidFactors*。复数向量应分配到 X 数据存储区中。

因子以浮点算术进行计算并转换为 1.15/1.31 复数小数。

源文件

- 对于 dsPIC30F/33F/33C/33E——inittwid.c
- 对于 dsPIC33A——inittwid_aa.c

7. 控制函数

函数	说明
PIDInit	清除 Y 数据空间中由 controlHistory 指向的 3 元素数组中的延时线元素。此外，还会清除当前 PID 输出元素 controlOutput。
PIDCoeffCalc	PIDInit 根据用户提供的 Kp、Ki 和 Kd 值计算 PID 系数。
PID	计算 controlOutput。

7.1. 比例积分微分（PID）控制

本节介绍 DSP 库中提供的有助于实现闭环控制系统的函数。关于比例积分微分（Proportional Integral Derivative，PID）控制器的完整讨论超出了本文档的范围，本节只提供一些调整 PID 控制器的相关指南。

7.1.1. PID 控制器背景

PID 控制器响应闭环控制中的误差信号，并尝试对控制量进行调节，以获得期望的系统响应。被控参数可以是任何可测量系统量，例如速度、电压或电流。PID 控制器的输出可以控制一个或多个影响被控系统量的系统参数。例如，在无传感器无刷直流电机应用中，速度控制环既可以直接控制 PWM 占空比，也可以为调节电机电流的内层控制环设置期望电流值。PID 控制器的优点在于，它可通过调节一个或多个增益值并观察系统响应的变化以根据经验进行调节。

数字 PID 控制器以周期性采样间隔执行，并且假定其执行频率足够高以确保系统能够得到适当的控制。例如，在无传感器无刷直流电机应用中，电流控制器每个 PWM 周期执行一次，因为电机电流变化可能非常快速；而速度控制器则以中等事件速率（100 Hz）执行，因为受机械时间常数影响，电机转速变化相对缓慢。

通过将被控参数的实际测量值减去该参数的期望设定值即可获得误差信号。该误差的符号表示控制输入所需的变化方向。

控制器的比例（P）项由误差信号与 P 增益相乘得到。因此，PID 控制器产生的控制响应为误差幅值的函数。当误差信号变大时，控制器的 P 项也会变大以提供更大的校正量。

随着时间的流逝，P 项的作用趋向于减小总误差。但是，P 项的作用将随着误差趋近于零而减小。在大部分系统中，被控参数的误差会非常接近于零，但是并不会收敛。因此始终会存在一个较小的稳态误差。控制器的积分（I）项用于修正较小的稳态误差。I 项获取连续运行的总误差信号。因此，较小的稳态误差会随时间累积为一个较大的误差值。该累积误差信号与一个 I 增益因子相乘，即成为 PID 控制器的 I 输出项。

PID 控制器的微分（D）项用于提升控制器响应速度（与误差信号的变化速率相关）。D 项输入通过计算当前误差值与前一误差值的差值获得。该增量误差值与 D 增益因子相乘，即成为 PID 控制器的 D 输出项。系统误差变化速率越快，控制器的 D 项生成的控制输出量越大。

请注意，并非所有 PID 控制器都会实现 D 项，I 项的实现则更为少见。例如 Microchip 应用笔记 AN901 中所述的无刷直流电机应用，由于电机转速变化响应相对较慢，因此未实现 D 项。在这种情况下，D 项可能引发 PWM 占空比剧烈波动，进而影响无传感器算法的正常运行并触发过流保护。

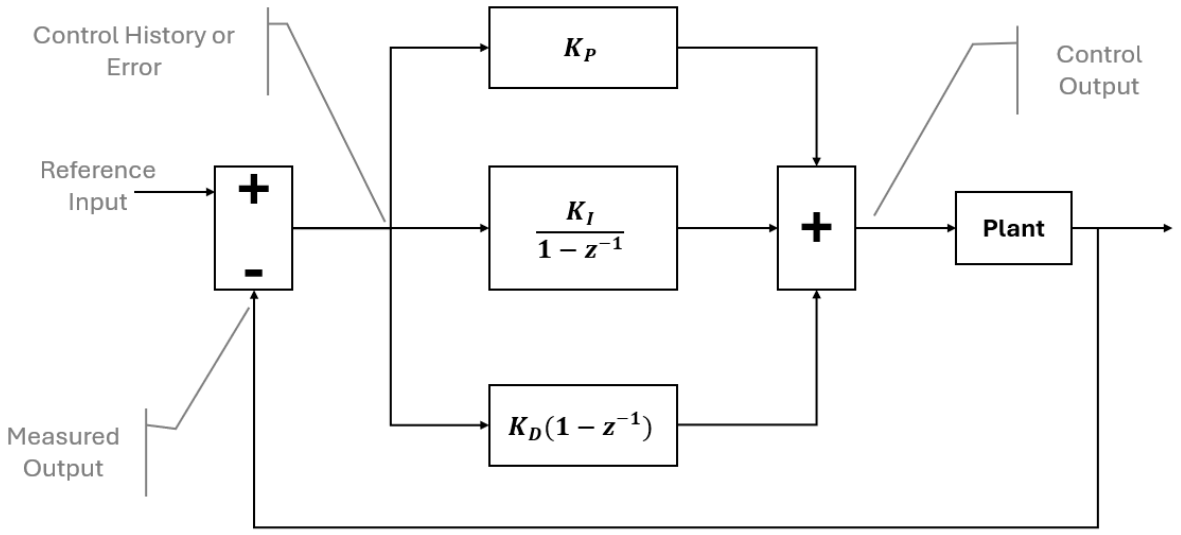
7.1.2. 调节 PID 增益

PID 控制器的 P 增益将设置整个系统的响应。初次调整控制器时，应将 I 增益和 D 增益设置为零。然后逐步增大 P 增益，直到系统对设定值变化的响应达到良好状态且无明显过冲或振荡现象为止。使用较小的 P 增益值将“宽松地”控制系统，而较大的值则会“更严格地”控制系统。此时，系统将有可能不收敛到设定值。

选择了合适的“P”增益后，可缓慢地增大 I 增益以将系统误差强制变为零。在大多数系统中，只需要较小的 I 增益。请注意，如果 I 增益取值过大，则可能抵消 P 增益的作用，减缓整个控制系统的响应，并使系统在设定值附近振荡。如果发生这种情况，通过减小 I 增益并增大 P 增益通常可解决问题。

当 P 增益和 I 增益完成设置后，即可设置 D 增益。D 项将加快控制变化的响应，但应谨慎使用，否则会导致控制器输出发生非常快速的变化。这种行为称为“设定值突跳”。之所以发生设定值突跳是因为当控制设定值发生改变时，系统误差会瞬间发生剧变。在某些情况下，可能会损坏系统硬件。如果将 D 增益设置为零时的系统响应已达到可接受范围，则省略 D 增益。

图 7-1. PID 控制系统



7.1.3. *PID 库函数和数据结构*

DSP 库提供了 PID 控制器函数 `PID`（`tPID*`）来执行 PID 运算。该函数使用头文件 `dsp.h` 中定义的数据结构，其形式如下：

```
typedef struct {
 fractional* abcCoefficients;
 fractional* controlHistory;
 fractional controlOutput;
 fractional measuredOutput;
 fractional controlReference;
} tPID;
```

调用 `PID()` 函数之前，应用程序应初始化 `tPID` 类型的数据结构，具体操作步骤如下：

1. 根据 PID 增益值计算系数。tPID 类型的数据结构中的元素 `abcCoefficients` 是指向 X 数据空间中 A、B 和 C 系数的指针。这些系数基于 PID 增益值 Kp、Ki 和 Kd（如图 7-1 所示）计算得出，计算公式为：`A = Kp + Ki + Kd B = -(Kp + 2*Kd) C = Kd` 。为计算 `A`、`B` 和 `C` 系数，DSP 库提供了 `PIDCoeffCalc` 函数。
2. 清除 PID 状态变量。结构元素 `controlHistory` 是指向 Y 数据空间中三个采样历史记录的指针，其中第一个采样是最新（当前）采样。这三个采样构成了被控对象函数的参考输入与被测输出之间的当前与过去差值历史记录。`PIDInit` 函数用于清除 `controlHistory` 指向的元素。此外，还会清除 `tPID` 数据结构中的 `controlOutput` 元素。

注：对于 dsPIC30F、dsPIC33F、dsPIC33C 和 dsPIC33E 系列器件，所有小数输入和输出参数均采用 1.15 定点数据类型格式。而对于 dsPIC33A 系列器件，这些参数则使用 1.31 定点数据类型格式。

7.2. 函数

7.2.1. PIDInit

说明

该程序用于清除 Y 数据空间中由 *controlHistory* 指向的三元素数组中的延时线元素。此外，还会清除当前 PID 输出元素 *controlOutput*。

原型

```
void PIDInit ( tPID *fooPIDStruct);
```

参数

参数	说明
fooPIDStruct	指向 tPID 类型的 PID 数据结构的指针

返回

无

备注

无

源文件

- 对于 dsPIC30F/33F/33E/33C
  pid.s
- 对于 dsPIC33A
  pid_aa.s

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	11	13
dsPIC33E/33C	14	21
dsPIC33A	6	10

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0*——已使用，未恢复
  - *DO* 和 *REPEAT* 指令的使用——无
- 对于 dsPIC33A
  - *W0*——已使用，未恢复
  - *REPEAT* 指令的使用——无

7.2.2. PIDCoeffCalc

说明

PIDCoeffCalc 用于根据用户提供的 Kp、Ki 和 Kd 值计算 PID 系数。

```
abcCoefficients[0] = Kp + Ki + Kd
abcCoefficients[1] = -(Kp + 2*Kd)
abcCoefficients[2] = Kd
```

此外，该程序还会清除数组 ControlDifference 中的延时线元素以及当前 PID 输出元素 ControlOutput。

原型

```
void PIDCoeffCalc ( fractional *fooPIDGainCoeff, tPID *fooPIDStruct);
```

参数

参数	说明
fooPIDGainCoeff	指向包含 Kp、Ki 和 Kd 系数（按[Kp, Ki, Kd]顺序排列）的输入数组的指针
fooPIDStruct	指向 tPID 类型的 PID 数据结构的指针

返回

无

备注

PID 系数数组元素可能受饱和影响，具体取决于 Kp、Ki 和 Kd 的值。

源文件

- 对于 dsPIC30F/33F/33E/33C
  pid.s
- 对于 dsPIC33A
  pid_aa.s

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	18	20
dsPIC33E/33C	21	28
dsPIC33A	15	28

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W2*——已使用，未恢复
  - *ACCA 和 ACCB*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用——无
- 对于 dsPIC33A
  - *W0*——已使用，未恢复
  - *ACCA 和 ACCB*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——无

7.2.3. PID

说明

PID 用于计算数据结构 tPID 的 controlOutput 元素：

```
controlOutput[n] = controlOutput[n-1]
```

```
                    + controlHistory[n] * abcCoefficient[0]
                    + controlHistory[n-1] * abcCoefficient[1]
                    + controlHistory[n-2] * abcCoefficient[2]
//where
abcCoefficient[0] = Kp + Ki + Kd
abcCoefficient[1] = -(Kp + 2*Kd)
abcCoefficient[2] = Kd
ControlHistory[n] =  MeasuredOutput[n] - ReferenceInput[n]
```

原型

`void PID ( tPID* fooPIDStruct );`

参数

参数	说明
fooPIDStruct	指向 tPID 类型的 PID 数据结构的指针

返回

指向 *fooPIDStruct* 的指针

备注

通过 PID() 程序更新 controlOutput 元素。controlOutput 将受饱和影响。

源文件

- 对于 dsPIC30F/33F/33E/33C
  pid.s
- 对于 dsPIC33A
  pid_aa.s

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	28	33
dsPIC33E/33C	31	42
dsPIC33A	22	34

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0...W5*——已使用，未恢复
  - *W8 和 W10*——已保存，已使用，已恢复
  - *ACCA 和 ACCB*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用——无
- 对于 dsPIC33A
  - *W0...W7*——已使用，未恢复
  - *ACCA 和 ACCB*——已使用，未恢复
  - *CORCON*——已保存，已使用，已恢复
  - *REPEAT* 指令的使用——无

8. 转换函数

8.1. 函数

函数	说明
Fract2Float	将 1.15/1.31 小数值转换为 IEEE 单精度浮点值。
Float2Fract	将 IEEE 单精度浮点值转换为 1.15/1.31 小数值。

8.1.1. Fract2Float

说明

该函数用于将 1.15/1.31 小数值转换为 IEEE 单精度浮点值。

原型

```
float Fract2Float (fractional aVal);
```

参数

参数	说明
aVal	1.15/1.31 小数值，具体取决于器件系列。 • 对于 dsPIC30F/33F/33E/33C 1.15 小数值，隐式范围为[-1, 1 - 2^{-15}] • 对于 dsPIC33A 1.31 小数值，隐式范围为[-1, 1 - 2^{-31}]

返回

IEEE 单精度浮点值的范围

- [-1, 1 - 2^{-15}]（对于 1.15 小数输入）
- [-1, 4.656613 x 10^{-10}]（对于 1.31 小数输入）

备注

对于 dsPIC33A，使用硬件浮点单元执行转换；对于其他 dsPIC DSC，使用收敛舍入和饱和机制执行转换。

源文件

- 对于 dsPIC30F/33F/33E/33C
  - flt2frct.c
- 对于 dsPIC33A
  - flt2frct_aa.s

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0..W7*——已使用，未恢复
  - *W8..W14*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用——无
- 对于 dsPIC33A
  - *W0*——已使用，未恢复
  - *F0 和 F1*——已使用，未恢复

- *REPEAT* 指令的使用——无

8.1.2. Float2Fract

说明

该函数用于将 IEEE 单精度浮点值转换为 1.15/1.31 小数值。

原型

```
fractional Float2Fract (float aVal);
```

参数

参数	说明
aVal	IEEE 单精度浮点值的范围 • [-1, 1 – 2^{-15}]（对于 dsPIC30F/33F/33E/33C） • [-1, 4.656613 x 10^{-10}]（对于 dsPIC33A）

返回

1.15/1.31 小数值，具体取决于器件系列。

- 对于 dsPIC30F/33F/33E/33C——1.15 小数值，隐式范围为[-1, 1 – 2^{-15}]
- 对于 dsPIC33A——1.31 小数值，隐式范围为[-1, 1 – 2^{-31}]

备注

对于 dsPIC33A，使用硬件浮点单元执行转换；对于其他器件，使用收敛舍入和饱和机制执行转换。

源文件

- 对于 dsPIC30F/33F/33E/33C
  - frct2flt.c
- 对于 dsPIC33A
  - frct2flt_aa.s

系统资源的使用

- 对于 dsPIC30F/33F/33E/33C
  - *W0..W7*——已使用，未恢复
  - *W8..W14*——已保存，已使用，已恢复
  - *DO* 和 *REPEAT* 指令的使用——无
- 对于 dsPIC33A
  - *W0*——已使用，未恢复
  - *F0..F2*——已使用，未恢复
  - *REPEAT* 指令的使用——无

9. 堆栈函数

本章介绍用于修改堆栈保护值的 *stackGuard* 函数。

9.1. SetStackGuard

说明

该函数用于修改堆栈保护值。

原型

```
void SetStackGuard (unsigned intstackGuard);
```

参数

参数	说明
stackGuard	堆栈保护值

返回

无

备注

对于 dsPIC33E/33C，当 *factPage* 指向 PSV 页时，可根据堆栈空间可用性将系数从 PSV 复制到堆栈中。

从 PSV 复制到堆栈中的条件如下：

SP + TABLE_SIZE + STACK_GUARD < SPLIM 且

SP + TABLE_SIZE < __YDATA_BASE，其中

1. *SP*——堆栈指针
2. *TABLE_SIZE*——PSV 中系数向量的大小
3. *STACK_GUARD*——堆栈中不用于存储从 PSV 复制的系数向量的缓冲区空间
4. *SPLIM*——堆栈指针限制
5. *__YDATA_BASE*——Y 存储区的基址

STACK_GUARD 的默认值为 2048 字，但可通过 *SetStackGuard* 函数进行修改。修改 STACK_GUARD 时必须小心。

值越大，为中断等预留的堆栈空间越大，但会更容易超出 SPLIM，导致代码耗尽 PSV 资源。这会增加周期数。如果发生这种情况，需减小 STACK_GUARD 的值。

值越小，为中断等预留的堆栈空间越小，但会不容易超出 SPLIM，导致代码耗尽 RAM 资源。由于缓冲区空间较小，因此可能会发生堆栈溢出。如果发生这种情况，需增大 STACK_GUARD 的值。

源文件

```
stackguard.s
```

函数配置文件

器件	程序字数	周期数
dsPIC30F/33F	2	4
dsPIC33E/33C	2	7
dsPIC33A	N/A	N/A

系统资源的使用

W0——已使用，未修改

DO 和 *REPEAT* 指令——未使用

注：

1. 对于 dsPIC33A，系数不会被复制到堆栈中，否则将允许直接从程序存储器读取系数。因此，对于这些器件来说，`STACK_GUARD` 和 *`SetStackGuard`* 函数已过时。

Microchip 信息

商标

“Microchip”的名称和徽标组合、“M”徽标及其他名称、徽标和品牌均为 Microchip Technology Incorporated 或其关联公司和/或子公司在美国和/或其他国家或地区的注册商标或商标（“Microchip 商标”）。有关 Microchip 商标的信息，可访问 https://www.microchip.com/en-us/about/legal-information/microchip-trademarks。

ISBN： 979-8-3371-1172-8

法律声明

提供本文档的中文版本仅为了便于理解。请勿忽视文档中包含的英文部分，因为其中提供了有关 Microchip 产品性能和使用情况的有用信息。Microchip Technology Inc.及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc.的英文原版文档。

本出版物及其提供的信息仅适用于 Microchip 产品，包括设计、测试以及将 Microchip 产品集成到您的应用中。以其他任何方式使用这些信息都将被视为违反条款。本出版物中的器件应用信息仅为您提供便利，将来可能会发生更新。您须自行确保应用符合您的规范。如需额外的支持，请联系当地的 Microchip 销售办事处，或访问 www.microchip.com/en-us/support/design-help/client-support-services。

Microchip“按原样”提供这些信息。Microchip 对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对非侵权性、适销性和特定用途的适用性的暗示担保，或针对其使用情况、质量或性能的担保。

在任何情况下，对于因这些信息或使用这些信息而产生的任何间接的、特殊的、惩罚性的、偶然的或附带的损失、损害或任何类型的开销，Microchip 概不承担任何责任，即使 Microchip 已被告知可能发生损害或损害可以预见。在法律允许的最大范围内，对于因这些信息或使用这些信息而产生的所有索赔，Microchip 在任何情况下所承担的全部责任均不超出您为获得这些信息向 Microchip 直接支付的金额（如有）。如果将 Microchip 器件用于生命维持和/或生命安全应用，一切风险由买方自负。买方同意在由此引发任何一切损害、索赔、诉讼或费用时，会维护和保障 Microchip 免于承担法律责任。除非另外声明，在 Microchip 知识产权保护下，不得暗中或以其他方式转让任何许可证。

Microchip 器件代码保护功能

请注意以下有关 Microchip 产品代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术规范。
- Microchip 确信：在正常使用且符合工作规范的情况下，Microchip 系列产品非常安全。
- Microchip 注重并积极保护其知识产权。严禁任何试图破坏 Microchip 产品代码保护功能的行为，这种行为可能会违反《数字千年版权法案》（Digital Millennium Copyright Act）。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。