CSI DSP 接口说明手册

I 系列 VDSP2 版本 Release v2.8.0

2018 年 12 月 12 日

Copyright © 2018 杭州中天微系统有限公司，保留所有权利。

本文档的产权属于杭州中天微系统有限公司 (下称中天公司)。本文档仅能分布给:(i) 拥有合法雇佣关系，并需要本文档的信息的中天微系统员工，或 (ii) 非中天微组织但拥有合法合作关系，并且其需要本文档的信息的合作方。对于本文档，禁止任何在专利、版权或商业秘密过程中，授予或暗示的可以使用该文档。在没有得到杭州中天微系统有限公司的书面许可前，不得复制本文档的任何部分，传播、转录、储存在检索系统中或翻译成任何语言或计算机语言。

商标申明

杭州中天微系统的 LOGO 和其它所有商标归杭州中天微系统有限公司所有，所有其它产品或服务名称归其所有者拥有。

注意

您购买的产品、服务或特性等应受中天公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，中天公司对本文档内容不做任何明示或默示的声明或保证。由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

This document is the property of Hangzhou C-SKY MicroSystems Co.,Ltd. This document may only be distributed to: (i) a C-SKY party having a legitimate business need for the information contained herein, or (ii) a non-C-SKY party having a legitimate business need for the information contained herein. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document. No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise without the prior written permission of C-SKY MicroSystems Co.,Ltd.

Trademarks and Permissions

The C-SKY Logo and all other trademarks indicated as such herein are trademarks of Hangzhou C-SKY MicroSystems Co.,Ltd. All other products or service names are the property of their respective owners.

Notice

The purchased products, services and features are stipulated by the contract made between C-SKY and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided ”AS IS” without warranties, guarantees or representations of any kind, either express or implied. The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

杭州中天微系统有限公司 C-SKY MicroSystems Co.,LTD

地址: 杭州市西湖区西斗门路 3 号天堂软件园 A 座 15 楼

邮编: 310012

网址: www.c-sky.com

Contents:

第一章 CSI DSP 软件库 (I 系列 VDSP2 版本) 1

1.1 简介 . . . 1
1.2 如何使用库 . . . 1
1.3 示例 . . . 1

第二章 基本数学函数 2

2.1 向量绝对值 . . . 2
2.2 向量绝对值最大值 . . . 4
2.3 向量加法 . . . 6
2.4 向量点积 . . . 8
2.5 向量乘法 . . . 11
2.6 向量相反数 . . . 14
2.7 向量偏移 . . . 16
2.8 向量缩放 . . . 18
2.9 向量移位 . . . 20
2.10 向量减法 . . . 22
2.11 向量求和 . . . 24

第三章 复数函数 25

3.1 复数共轭 . . . 25
3.2 复数点积 . . . 27
3.3 复数幅度 . . . 29
3.4 复数幅度平方 . . . 31
3.5 复数与复数相乘 . . . 33
3.6 复数与实数相乘 . . . 35

第四章 滤波函数 37

4.1 卷积 . . . 37
4.2 部分卷积 . . . 44
4.3 相关分析 . . . 49
4.4 有限冲激响应（FIR）滤波器 . . . 55
4.5 有限冲激响应（FIR）抽取器 . . . 62
4.6 有限冲激响应 (FIR) 格型滤波器 . . . 68
4.7 有限冲激响应 (FIR) 稀疏滤波器 . . . 71
4.8 有限冲激响应 (FIR) 插值滤波器 . . . 76
4.9 无限冲激响应 (IIR) 格型滤波器 . . . 80

4.10 最小均方 (LMS) 滤波器 84
4.11 归一化 LMS 滤波器 89

第五章 矩阵函数 **94**
5.1 矩阵初始化 95
5.2 矩阵加法 96
5.3 矩阵减法 98
5.4 复数矩阵乘法 100
5.5 矩阵乘法 102
5.6 矩阵缩放 105
5.7 矩阵转置 107

第六章 统计函数 **109**
6.1 最大值 109
6.2 最小值 111
6.3 平均值 113
6.4 平方和 115
6.5 均方根 (RMS) 118
6.6 标准偏差 120
6.7 方差 122

第七章 辅助函数 **124**
7.1 向量复制 124
7.2 向量填充 126
7.3 转换 Q15 的值 128
7.4 转换 Q31 的值 130
7.5 转换 Q7 的值 133

第八章 变换函数 **135**
8.1 复数 FFT 函数 135
8.2 实数 FFT 函数 138
8.3 DCT IV 型函数 144

第一章 CSI DSP 软件库 (I 系列 VDSP2 版本)

1.1 简介

这份手册描述的 CSI DSP 软件库的 VDSP2 版本, 是一些用于 I805/CK805 设备的，使用了 VDSP2 指令集加速的通用信号处理函数的集合。

库内的各个函数可以分类为:

- 基本数学函数
- 复数函数
- 滤波函数
- 矩阵函数
- 变换函数
- 统计函数
- 辅助函数

库内的大多数函数都有 Q7，Q15，Q31 三种版本。

1.2 如何使用库

Lib 目录内提供了一些预编译的版本，分别是:

- libcsky_I805_vdsp2_math.a
- libcsky_CK805_vdsp2_math.a

函数库的函数声明在头文件 `csky_vdsp2_math.h` 中，`csky_vdsp2_math.h` 源码放在 Include 目录中。

在应用程序中包括 `csky_vdsp2_math.h` 头文件, 就可以直接调用 DSP 库函数; 链接的时候指定应用对应的库版本, 就可以将库函数链接进应用程序。

1.3 示例

CSI SDK 里面带了一些示例演示如何使用库函数。

第二章 基本数学函数

2.1 向量绝对值

2.1.1 函数

- *csky_vdsp2_abs_q15* : Q15 向量绝对值.
- *csky_vdsp2_abs_q31* : Q31 向量绝对值.
- *csky_vdsp2_abs_q7* : Q7 向量绝对值.

2.1.2 简要说明

向量的每个元素取绝对值.

```
pDst[n] = abs(pSrc[n]),   0 <= n < blockSize.
```

这些函数可以在原值上做计算，也就是说，允许源和目的向量指针指向相同地址。
为 Q7，Q15，Q31 每种类型都提供了不同的函数。

2.1.3 函数说明

2.1.3.1 csky_vdsp2_abs_q15

```
void csky_vdsp2_abs_q15 (q15_t *pSrc, q15_t *pDst, uint32_t blockSize)
```

参数:

`*pSrc`: 指向输入向量
`*pDst`: 指向输出向量
`blockSize`: 向量中的元素数

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. Q15 的值为-1 (0x8000) 时会饱和为最大的正数值 0x7FFF.

2.1.3.2 csky_vdsp2_abs_q31

```
void csky_vdsp2_abs_q31 (q31_t *pSrc, q31_t *pDst, uint32_t blockSize)
```

参数:

*pSrc: 指向输入向量
*pDst: 指向输出向量
blockSize: 向量中的元素数

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法。Q31 的值为-1 (0x80000000) 时会饱和为最大的正数值 0x7FFFFFFF。

2.1.3.3 csky_vdsp2_abs_q7

```
void csky_vdsp2_abs_q7 (q7_t *pSrc, q7_t *pDst, uint32_t blockSize)
```

参数:

*pSrc: 指向输入向量
*pDst: 指向输出向量
blockSize: 向量中的元素数

返回值:

无

最佳性能的条件:

输入和输出 buffer 都是 32 位对齐.

缩放和溢出时的行为:

函数使用饱和算法。Q7 的值为 -1 (0x80) 时会饱和为最大的正数值 0x7F 。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

2.2 向量绝对值最大值

2.2.1 函数

- *csky_vdsp2_abs_max_q15*：Q15 向量绝对值最大值.
- *csky_vdsp2_abs_max_q31*：Q31 向量元素绝对值最大值.

2.2.2 简要说明

向量的每个元素取绝对值，然后返回绝对值中的最大值.

```
*pDst = max(abs(pSrc[n])),   0 <= n < blockSize.
```

为 Q15，Q31 两种类型提供了不同的函数。

2.2.3 函数说明

2.2.3.1 csky_vdsp2_abs_max_q15

```
void csky_vdsp2_abs_max_q15 (q15_t *pSrc, q15_t *pDst, uint32_t blockSize)
```

参数:

`*pSrc`: 指向输入向量

`*pDst`: 指向最大值

`blockSize`: 向量中的元素数

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. Q15 的值为-1 (0x8000) 时会饱和为最大的正数值 0x7FFF.

2.2.3.2 csky_vdsp2_abs_max_q31

```
void csky_vdsp2_abs_max_q31 (q31_t *pSrc, q31_t *pDst, uint32_t blockSize)
```

参数:

`*pSrc`: 指向输入向量

`*pDst`: 指向最大值

`blockSize`: 向量中的元素数

返回值:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

无

缩放和溢出时的行为:

函数使用饱和算法。Q31 的值为-1 (0x80000000) 时会饱和为最大的正数值 0x7FFFFFFF。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

2.3 向量加法

2.3.1 函数

- *csky_vdsp2_add_q15* : Q15 向量加法.
- *csky_vdsp2_add_q31* : Q31 向量加法.
- *csky_vdsp2_add_q7* : Q7 向量加法.

2.3.2 简要说明

两个向量的元素逐个相加.

```
pDst[n] = pSrcA[n] + pSrcB[n],   0 <= n < blockSize.
```

为 Q7，Q15，Q31 每种类型都提供了不同的函数.

2.3.3 函数说明

2.3.3.1 csky_vdsp2_add_q15

```
void csky_vdsp2_add_q15 (q15_t *pSrcA, q15_t *pSrcB, q15_t *pDst, uint32_t blockSize)
```

参数:

`*pSrcA`: 指向第一个输入向量

`*pSrcB`: 指向第二个输入向量

`*pDst`: 指向输出向量

`blockSize`: 向量中的元素数

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. 返回值超出 Q15 的最大范围 [0x8000 0x7FFF] 时会被饱和处理.

2.3.3.2 csky_vdsp2_add_q31

```
void csky_vdsp2_add_q31 (q31_t *pSrcA, q31_t *pSrcB, q31_t *pDst, uint32_t blockSize)
```

参数:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

*pSrcA: 指向第一个输入向量

*pSrcB: 指向第二个输入向量

*pDst: 指向输出向量

blockSize: 向量中的元素数

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. 返回值超出 Q31 的最大范围 [0x80000000 0x7FFFFFFF] 时会被饱和处理.

2.3.3.3 csky_vdsp2_add_q7

```
void csky_vdsp2_add_q7 (q7_t *pSrcA, q7_t *pSrcB, q7_t *pDst, uint32_t blockSize)
```

参数:

*pSrcA: 指向第一个输入向量

*pSrcB: 指向第二个输入向量

*pDst: 指向输出向量

blockSize: 向量中的元素数

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. 返回值超出 Q7 的最大范围 [0x80 0x7F] 时会被饱和处理.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

2.4 向量点积

2.4.1 函数

- *csky_vdsp2_dot_prod_q15*：Q15 向量的点积.
- *csky_vdsp2_dot_prod_q31*：Q31 向量的点积.
- *csky_vdsp2_dot_prod_q7*：Q7 向量点积.
- *csky_vdsp2_dot_prod_u64xu8*：Uint_t64, Uint8_t 向量点积.

2.4.2 简要说明

计算两个向量的点积. 向量的每个元素相乘，然后累加.

```
sum = pSrcA[0]*pSrcB[0] + pSrcA[1]*pSrcB[1] + ... + pSrcA[blockSize-1]*pSrcB[blockSize-1]
```

为 Q7，Q15，Q31 每种类型都提供了不同的函数.

其中，U64xU8 比较特殊，因为指令中没有 64 位乘法，其计算过程如下：

```
pSrcB = Zero_Extend(pSrcB)(8bit->32bit)
tmp   = pSrcA[i][31:0]*pSrcB[i] + (pSrcA[i][63:32]*pSrcB[i] << 32)
sum   += tmp
```

2.4.3 函数说明

2.4.3.1 csky_vdsp2_dot_prod_q15

```
void csky_vdsp2_dot_prod_q15 (q15_t *pSrcA, q15_t *pSrcB, uint32_t blockSize, q63_t *result)
```

参数:

`*pSrcA`: 指向第一个输入向量

`*pSrcB`: 指向第二个输入向量

`blockSize`: 向量中的元素数量

`*result`: 输出的返回结果

返回值:

无

缩放和溢出时的行为:

中间步骤的乘法用的是 1.15 x 1.15 = 2.30 格式，然后乘法结果累加成一个 64 位 34.30 格式定点数。累加结果有 33 个保护位，相加的时候不需要使用饱和算法，因为不会出现溢出. 返回结果是 34.30 格式.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

2.4.3.2 csky_vdsp2_dot_prod_q31

```
void csky_vdsp2_dot_prod_q31 (q31_t *pSrcA, q31_t *pSrcB, uint32_t blockSize, q63_t *result)
```

参数:

*pSrcA: 指向第一个输入向量

*pSrcB: 指向第二个输入向量

blockSize: 向量中的元素数量

*result: 输出的返回结果

返回值:

无

缩放和溢出时的行为:

中间步骤的乘法用的是 1.31 x 1.31 = 2.62 格式，并且通过丢弃低 14 位截断成 2.48. 2.48 的乘法结果累加成一个 64 位 16.48 格式定点数。累加结果有 15 个保护位，相加的时候不需要使用饱和算法，因为不会出现溢出. 返回结果是 16.48 格式.

2.4.3.3 csky_vdsp2_dot_prod_q7

```
void csky_vdsp2_dot_prod_q7 (q7_t *pSrcA, q7_t *pSrcB, uint32_t blockSize, q31_t *result)
```

参数:

*pSrcA: 指向第一个输入向量

*pSrcB: 指向第二个输入向量

blockSize: 向量中的元素数量

*result: 输出的返回结果

返回值:

无

缩放和溢出时的行为:

中间步骤的乘法用的是 1.7 x 1.7 = 2.14 格式，然后乘法结果累加成一个 18.14 格式定点数。累加结果有 17 个保护位，相加的时候不需要使用饱和算法，因为不会出现溢出. 返回结果是 18.14 格式.

2.4.3.4 csky_vdsp2_dot_prod_u64xu8

```
void csky_vdsp2_dot_prod_u64xu8 (uint8_t *pSrcA, uint64_t *pSrcB, uint32_t blockSize, uint64_t␣
↪*result)
```

参数:

*pSrcA: 指向第一个输入向量

*pSrcB: 指向第二个输入向量

blockSize: 向量中的元素数量

*result: 输出的返回结果

返回值:

无

缩放和溢出时的行为:

中间步骤的乘法用的是 64 位累加器，累加结果没有保护位，且未使用饱和算法，使用时需要限制输入值的范围，以防溢出.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

2.5 向量乘法

2.5.1 函数

- *csky_vdsp2_mult_q15*：Q15 向量乘法.
- *csky_vdsp2_mult_q31*：Q31 向量乘法.
- *csky_vdsp2_mult_q7*：Q7 向量乘法.
- *csky_vdsp2_mult_rnd_q15*：Q15 带舍入向量相乘.
- *csky_vdsp2_mult_q15xq31_sht*：Q15xQ31 带移位向量相乘.

2.5.2 简要说明

两个向量的元素逐个相乘.

```
pDst[n] = pSrcA[n] * pSrcB[n],   0 <= n < blockSize.
```

为 Q7，Q15，Q31 每种类型都提供了不同的函数.

2.5.3 函数说明

2.5.3.1 csky_vdsp2_mult_q15

```
void csky_vdsp2_mult_q15 (q15_t *pSrcA, q15_t *pSrcB, q15_t *pDst, uint32_t blockSize)
```

参数:

*pSrcA: 指向第一个输入向量
*pSrcB: 指向第二个输入向量
*pDst: 指向输出向量
blockSize: 每个向量的元素数量

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. 结果超出 Q15 的最大范围 [0x8000 0x7FFF] 时会被饱和处理.

2.5.3.2 csky_vdsp2_mult_q31

```
void csky_vdsp2_mult_q31 (q31_t *pSrcA, q31_t *pSrcB, q31_t *pDst, uint32_t blockSize)
```

参数:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

*pSrcA: 指向第一个输入向量

*pSrcB: 指向第二个输入向量

*pDst: 指向输出向量

blockSize: 每个向量的元素数量

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. 结果超出 Q31 的最大范围 [0x80000000 0x7FFFFFFF] 时会被饱和处理.

2.5.3.3 csky_vdsp2_mult_q7

```
void csky_vdsp2_mult_q7 (q7_t *pSrcA, q7_t *pSrcB, q7_t *pDst, uint32_t blockSize)
```

参数:

*pSrcA: 指向第一个输入向量

*pSrcB: 指向第二个输入向量

*pDst: 指向输出向量

blockSize: 每个向量的元素数量

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. 结果超出 Q7 的最大范围 [0x80 0x7F] 时会被饱和处理.

2.5.3.4 csky_vdsp2_mult_rnd_q15

```
void csky_vdsp2_mult_rnd_q15 (q15_t *pSrcA, q15_t *pSrcB, q15_t *pDst, uint32_t blockSize)
```

参数:

*pSrcA: 指向第一个输入向量

*pSrcB: 指向第二个输入向量

*pDst: 指向输出向量

blockSize: 每个向量的元素数量

返回值:

无

缩放和溢出时的行为:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

函数使用了饱和算法。输出值超出了 Q15 的允许范围 [0x8000 0x7FFF] 就会被饱和，且在饱和之前，结果会被舍入，舍入方式为向正无穷舍入。

2.5.3.5 csky_vdsp2_mult_q15xq31_sht

```
void csky_vdsp2_mult_q15xq31_sht (q15_t *pSrcA, q31_t *pDst, uint32_t shiftValue, uint32_t␣
↪blockSize)
```

参数:

`*pSrcA`: 指向第一个输入向量

`*pSrcB`: 指向第二个输入向量和输出向量

`shiftVale`: 输出结果移位值

`blockSize`: 每个向量的元素数量

返回值:

无

缩放和溢出时的行为:

函数使用了饱和算法。输出值超出了 Q31 的允许范围 [0x80000000 0x7FFFFFFF] 就会被饱和，且在饱和之前，需要对结果右移 shiftValue，舍入方式为向正无穷舍入。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

2.6 向量相反数

2.6.1 函数

- *csky_vdsp2_negate_q15*：Q15 向量的所有元素取相反数.
- *csky_vdsp2_negate_q31*：Q31 向量的所有元素取相反数.
- *csky_vdsp2_negate_q7*：Q7 向量的所有元素取相反数.

2.6.2 简要说明

向量的所有元素取相反数。

```
pDst[n] = -pSrc[n],   0 <= n < blockSize.
```

这些函数可以在原值上做计算，也就是说，允许源和目的向量指针指向相同地址。

为 Q7，Q15，Q31 每种类型都提供了不同的函数。

2.6.3 函数说明

2.6.3.1 csky_vdsp2_negate_q15

```
void csky_vdsp2_negate_q15 (q15_t *pSrc, q15_t *pDst, uint32_t blockSize)
```

参数:

*pSrc: 指向输入向量

*pDst: 指向输出向量

blockSize: 向量中的元素数量

返回值:

无

最佳性能的条件

输入和输出 buffer 都是 32 位对齐

缩放和溢出时的行为:

函数使用饱和算法. Q15 的值为 -1 (0x8000) 时会饱和为最大的正数值 0x7FFF.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

2.6.3.2 csky_vdsp2_negate_q31

```
void csky_vdsp2_negate_q31 (q31_t *pSrc, q31_t *pDst, uint32_t blockSize)
```

参数:

*pSrc: 指向输入向量

*pDst: 指向输出向量

blockSize: 向量中的元素数量

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. Q31 的值为 -1 (0x80000000) 时会饱和为最大的正数值 0x7FFFFFFF.

2.6.3.3 csky_vdsp2_negate_q7

```
void csky_vdsp2_negate_q7 (q7_t *pSrc, q7_t *pDst, uint32_t blockSize)
```

参数:

*pSrc: 指向输入向量

*pDst: 指向输出向量

blockSize: 向量中的元素数量

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. Q7 的值为 -1 (0x80) 时会饱和为最大的正数值 0x7F.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

2.7 向量偏移

2.7.1 函数

- *csky_vdsp2_offset_q15*：Q15 向量添加一个常数偏移量
- *csky_vdsp2_offset_q31*：Q31 向量添加一个常数偏移量
- *csky_vdsp2_offset_q7*：Q7 向量添加一个常数偏移量.

2.7.2 简要说明

向量的每个元素添加一个常数偏移量。

```
pDst[n] = pSrc[n] + offset,   0 <= n < blockSize.
```

这些函数可以在原值上做计算，也就是说，允许源和目的向量指针指向相同地址。

为 Q7，Q15，Q31 每种类型都提供了不同的函数.

2.7.3 函数说明

2.7.3.1 csky_vdsp2_offset_q15

```
void csky_vdsp2_offset_q15 (q15_t *pSrc, q15_t offset, q15_t *pDst, uint32_t blockSize)
```

参数:

`*pSrc`: 指向输入向量

`offset`: 添加的偏移量

`*pDst`: 指向输出向量

`blockSize`: 向量中的元素数

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. 结果超出 Q15 的最大范围 [0x8000 0x7FFF] 时会被饱和处理.

2.7.3.2 csky_vdsp2_offset_q31

```
void csky_vdsp2_offset_q31 (q31_t *pSrc, q31_t offset, q31_t *pDst, uint32_t blockSize)
```

参数:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

`*pSrc`: 指向输入向量

`offset`: 添加的偏移量

`*pDst`: 指向输出向量

`blockSize`: 向量中的元素数

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. 结果超出 Q31 的最大范围 [0x80000000 0x7FFFFFFF] 时会被饱和处理.

2.7.3.3 csky_vdsp2_offset_q7

```
void csky_vdsp2_offset_q7 (q7_t *pSrc, q7_t offset, q7_t *pDst, uint32_t blockSize)
```

参数:

`*pSrc`: 指向输入向量

`offset`: 添加的偏移量

`*pDst`: 指向输出向量

`blockSize`: 向量中的元素数

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. 结果超出 Q7 的最大范围 [0x80 0x7F] 时会被饱和处理.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

2.8 向量缩放

2.8.1 函数

- *csky_vdsp2_scale_q15* : Q15 向量缩放.
- *csky_vdsp2_scale_q31* : Q31 向量缩放.
- *csky_vdsp2_scale_q7* : Q7 向量缩放.

2.8.2 简要说明

在定点函数 Q7, Q15 和 Q31 中, scale 表现为一个分数乘法 scaleFract 和一个算术移位 shift. 该偏移允许缩放操作的增益超过 1.0:

```
pDst[n] = (pSrc[n] * scaleFract) << shift,   0 <= n < blockSize.
```

应用于定点数据的总比例因子是

```
scale = scaleFract * 2^shift.
```

这些函数可以在原值上做计算，也就是说，允许源和目的向量指针指向相同地址。

2.8.3 函数说明

2.8.3.1 csky_vdsp2_scale_q15

```
void csky_vdsp2_scale_q15 (q15_t *pSrc, q15_t scaleFract, int8_t shift, q15_t *pDst, uint32_t␣
↪blockSize)
```

参数:

`*pSrc`: 指向输入向量

`scaleFract`: 小数部分的比例值

`shift`: 将结果移位的位数

`*pDst`: 指向输出向量

`blockSize`: 向量中的元素数

返回值:

无

缩放和溢出时的行为:

输入数据 pSrc 和 scaleFract 是 1.15 格式. 相乘的中间结果是 2.30 格式，然后做饱和移位到 1.15 格式.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

2.8.3.2 csky_vdsp2_scale_q31

```
void csky_vdsp2_scale_q31 (q31_t *pSrc, q31_t scaleFract, int8_t shift, q31_t *pDst, uint32_t␣
↪blockSize)
```

参数:

*pSrc: 指向输入向量

scaleFract: 小数部分的比例值

shift: 将结果移位的位数

*pDst: 指向输出向量

blockSize: 向量中的元素数

返回值:

无

缩放和溢出时的行为:

输入数据 pSrc 和 scaleFract 是 1.31 格式. 相乘的中间结果是 2.62 格式，然后做饱和移位到 1.31 格式.

2.8.3.3 csky_vdsp2_scale_q7

```
void csky_vdsp2_scale_q7 (q7_t *pSrc, q7_t scaleFract, int8_t shift, q7_t *pDst, uint32_t␣
↪blockSize)
```

参数:

*pSrc: 指向输入向量

scaleFract: 小数部分的比例值

shift: 将结果移位的位数

*pDst: 指向输出向量

blockSize: 向量中的元素数

返回值:

无

缩放和溢出时的行为:

输入数据 pSrc 和 scaleFract 是 1.7 格式. 相乘的中间结果是 2.14 格式，然后做饱和移位到 1.7 格式.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

2.9 向量移位

2.9.1 函数

- *csky_vdsp2_shift_q15*：Q15 向量的所有元素移位指定位数
- *csky_vdsp2_shift_q31*：Q31 向量的所有元素移位指定位数
- *csky_vdsp2_shift_q7*：Q7 向量的所有元素移位指定位数

2.9.2 简要说明

将定点向量的元素移位指定的位数. 为 Q7，Q15，Q31 每种类型都提供了不同的函数. 使用的算法如下:

```
pDst[n] = pSrc[n] << shift,   0 <= n < blockSize.
```

如果 shift 是正数，则向量的元素向左移位. 如果 shift 是负数，则向量的元素向右移位.

这些函数可以在原值上做计算，也就是说，允许源和目的向量指针指向相同地址.

2.9.3 函数说明

2.9.3.1 csky_vdsp2_shift_q15

```
void csky_vdsp2_shift_q15 (q15_t *pSrc, int8_t shiftBits, q15_t *pDst, uint32_t blockSize)
```

参数:

`*pSrc`: 指向输入向量

`shiftBits`: 移位的数量. 正数是左移，负数是右移.

`*pDst`: 指向输出向量

`blockSize`: 向量中元素的数量

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. 结果超出 Q15 的最大范围 [0x8000 0x7FFF] 时会被饱和处理.

2.9.3.2 csky_vdsp2_shift_q31

```
void csky_vdsp2_shift_q31 (q31_t *pSrc, int8_t shiftBits, q31_t *pDst, uint32_t blockSize)
```

参数:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

`*pSrc`: 指向输入向量

`shiftBits`: 移位的数量. 正数是左移，负数是右移

`*pDst`: 指向输出向量

`blockSize`: 向量中元素的数量

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. 结果超出 Q31 的最大范围 [0x80000000 0x7FFFFFFF] 时会被饱和处理.

2.9.3.3 csky_vdsp2_shift_q7

```
void csky_vdsp2_shift_q7 (q7_t *pSrc, int8_t shiftBits, q7_t *pDst, uint32_t blockSize)
```

参数:

`*pSrc`: 指向输入向量

`shiftBits`: 移位的数量. 正数是左移，负数是右移

`*pDst`: 指向输出向量

`blockSize`: 向量中元素的数量

返回值:

无

最佳性能的条件

输入和输出 buffer 都是 32 位对齐

缩放和溢出时的行为:

函数使用饱和算法. 结果超出 Q7 的最大范围 [0x80 0x7F] 时会被饱和处理.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

2.10 向量减法

2.10.1 函数

- *csky_vdsp2_sub_q15*：Q15 向量减法.
- *csky_vdsp2_sub_q31*：Q31 向量减法.
- *csky_vdsp2_sub_q7*：Q7 向量减法.

2.10.2 简要说明

两个向量的元素逐个相减.

```
pDst[n] = pSrcA[n] - pSrcB[n],   0 <= n < blockSize.
```

为 Q7，Q15，Q31 每种类型都提供了不同的函数.

2.10.3 函数说明

2.10.3.1 csky_vdsp2_sub_q15

```
void csky_vdsp2_sub_q15 (q15_t *pSrcA, q15_t *pSrcB, q15_t *pDst, uint32_t blockSize)
```

参数:

`*pSrcA`: 指向第一个输入向量

`*pSrcB`: 指向第二个输入向量

`*pDst`: 指向输出向量

`blockSize`: 向量中元素的数量

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. 结果超出 Q15 的最大范围 [0x8000 0x7FFF] 时会被饱和处理.

2.10.3.2 csky_vdsp2_sub_q31

```
void csky_vdsp2_sub_q31 (q31_t *pSrcA, q31_t *pSrcB, q31_t *pDst, uint32_t blockSize)
```

参数:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

`*pSrcA`: 指向第一个输入向量

`*pSrcB`: 指向第二个输入向量

`*pDst`: 指向输出向量

`blockSize`: 向量中元素的数量

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. 结果超出 Q31 的最大范围 [0x80000000 0x7FFFFFFF] 时会被饱和处理.

2.10.3.3 csky_vdsp2_sub_q7

```
void csky_vdsp2_sub_q7 (q7_t *pSrcA, q7_t *pSrcB, q7_t *pDst, uint32_t blockSize)
```

参数:

`*pSrcA`: 指向第一个输入向量

`*pSrcB`: 指向第二个输入向量

`*pDst`: 指向输出向量

`blockSize`: 向量中元素的数量

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. 结果超出 Q7 的最大范围 [0x80 0x7F] 时会被饱和处理.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

2.11 向量求和

2.11.1 函数

- *csky_vdsp2_sum_q15* : Q15 向量求和.

2.11.2 简要说明

向量的每个元素累加求和.

```
sum = pSrcA[0] + pSrcA[1] + ... + pSrcA[n],   0 <= n < blockSize.
```

为 Q15 类型都提供了函数.

2.11.3 函数说明

2.11.3.1 csky_vdsp2_sum_q15

```
void csky_vdsp2_sum_q15 (q15_t *pSrcA, q63_t *pDst, uint32_t blockSize)
```

参数:

`*pSrcA`: 指向第一个输入向量

`*pDst`: 指向输出地址

`blockSize`: 向量中元素的数量

返回值:

无

缩放和溢出时的行为:

函数使用了 64 位累加器，有 48 个保护位，因此不会溢出. 不需要进行饱和处理.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

第三章　复数函数

这组函数操作复数向量。复数向量的元素以交错的方式保存 (real, imag, real, imag, ···). 接口函数中，指定的样本数量是复数元素的个数；向量实际包括了两倍数量的数值。

3.1　复数共轭

3.1.1　函数

- *csky_vdsp2_cmplx_conj_q15*：Q15 复数共轭
- *csky_vdsp2_cmplx_conj_q31*：Q31 复数共轭

3.1.2　简要说明

复数向量的所有元素共轭

pSrc 指向源数据，pDst 指向结果写入的目的地址. numSamples 指定复数元素的个数，复数数据是交错方式保存的 (real, imag, real, imag, ···). 每个向量一共有 2*numSamples 个值.

使用的算法如下:

```
for (n=0; n<numSamples; n++) {
    pDst[(2 * n) + 0)] = pSrc[(2 * n) + 0];     // real part
    pDst[(2 * n) + 1)] = -pSrc[(2 * n) + 1];    // imag part
}
```

为 Q15 和 Q31 类型都提供了不同的函数.

3.1.3　函数说明

3.1.3.1　csky_vdsp2_cmplx_conj_q15

```
void csky_vdsp2_cmplx_conj_q15 (q15_t *pSrc, q15_t *pDst, uint32_t numSamples)
```

参数:

*pSrc: 指向输入向量

*pDst: 指向输出向量

numSamples: 向量中的复数元素个数

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. Q15 的值为 -1 (0x8000) 时会饱和为最大的正数值 0x7FFF.

3.1.3.2 csky_vdsp2_cmplx_conj_q31

```
void csky_vdsp2_cmplx_conj_q31 (q31_t *pSrc, q31_t *pDst, uint32_t numSamples)
```

参数:

*pSrc: 指向输入向量

*pDst: 指向输出向量

numSamples: 向量中的复数元素个数

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. Q31 的值为 -1 (0x80000000) 时会饱和为最大的正数值 0x7FFFFFFF.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

3.2 复数点积

3.2.1 函数

- *csky_vdsp2_cmplx_dot_prod_q15*：Q15 复数点积
- *csky_vdsp2_cmplx_dot_prod_q31*：Q31 复数点积

3.2.2 简要说明

计算两个复数向量的点积. 向量的元素逐个相乘，然后累加.

pSrcA 指向第一个复数输入向量，pSrcB 指向第二个复数输入向量. numSamples 指定复数元素的个数，复数数据是交错方式保存的 (real, imag, real, imag, ⋯). 每个向量一共有 2*numSamples 个值.

使用的算法如下:

```
realResult = 0;
imagResult = 0;
for (n = 0; n < numSamples; n++) {
    realResult += pSrcA[(2 * n)+0] * pSrcB[(2 * n)+0] - pSrcA[(2 * n)+1]* pSrcB[(2 * n)+1];
    imagResult += pSrcA[(2 * n)+0] * pSrcB[(2 * n)+1] + pSrcA[(2 * n)+1]* pSrcB[(2 * n)+0];
}
```

为 Q15 和 Q31 类型都提供了不同的函数.

3.2.3 函数说明

3.2.3.1 csky_vdsp2_cmplx_dot_prod_q15

```
void csky_vdsp2_cmplx_dot_prod_q15 (q15_t *pSrcA, q15_t *pSrcB, uint32_t numSamples, q31_t␣
↪*realResult, q31_t *imagResult)
```

参数:

*pSrcA: 指向第一个输入向量

*pSrcB: 指向第二个输入向量

numSamples: 向量中的复数元素数量

*realResult: 结果的实部

*imagResult: 结果的虚部

返回值:

无

缩放和溢出时的行为:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

函数的实现使用了一个内部的 64 位累加器. 1.15 格式和 1.15 格式相乘的中间结果用的是全精度, 产生 2.30 格式的结果. 64 位累加器将结果累加位 34.30 格式的值. 最后, 累加结果转换为 8.24 格式. 返回结果的 realResult 和 imagResult 是 8.24 格式.

3.2.3.2 csky_vdsp2_cmplx_dot_prod_q31

```
void  csky_vdsp2_cmplx_dot_prod_q31 (q31_t *pSrcA, q31_t *pSrcB, uint32_t numSamples, q63_t␣
↪*realResult, q63_t *imagResult)
```

参数:

*pSrcA: 指向第一个输入向量

*pSrcB: 指向第二个输入向量

numSamples: 向量中的复数元素数量

*realResult: 结果的实部

*imagResult: 结果的虚部

返回值:

无

缩放和溢出时的行为:

函数的实现使用了一个内部的 64 位累加器. 1.31 格式和 1.31 格式相乘的结果是 64 位精度, 移位成 16.48 格式. 中间结果的实部和虚部累加用的都是 16.48 格式. 只要 numSamples 少于 32768, 加法就不会溢出, 也就不需要饱和操作. 返回结果 realResult 和 imagResult 是 16.48 格式. 不需要输入向下缩放.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

3.3 复数幅度

3.3.1 函数

- *csky_vdsp2_cmplx_mag_q15* : Q15 复数幅度
- *csky_vdsp2_cmplx_mag_q31* : Q31 复数幅度

3.3.2 简要说明

计算复数向量的每个元素的幅度.

pSrc 指向源数据，pDst 指向结果写入的地址. numSamples 指定输入向量的复数元素个数，复数数据是交错方式保存的 (real, imag, real, imag, ···). 输入向量一共有 2*numSamples 个值; 输出向量一共有 numSamples 个值.

使用的算法如下:

```
for(n=0; n<numSamples; n++) {
    pDst[n] = sqrt(pSrc[(2*n)+0]^2 + pSrc[(2*n)+1]^2);
}
```

为 Q15 和 Q31 类型都提供了不同的函数.

3.3.3 函数说明

3.3.3.1 csky_vdsp2_cmplx_mag_q15

```
void csky_vdsp2_cmplx_mag_q15 (q15_t *pSrc, q15_t *pDst, uint32_t numSamples)
```

参数:

`*pSrc`: 指向复数输入向量
`*pDst`: 指向输出向量
`numSamples`: 输入向量的复数元素个数

返回值:

无

缩放和溢出时的行为:

函数实现 1.15 和 1.15 的乘法，最后输出转换成 2.14 格式。当实部和虚部都是 0x8000 时，中间过程可能溢出。

3.3.3.2 csky_vdsp2_cmplx_mag_q31

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

```
void csky_vdsp2_cmplx_mag_q31 (q31_t *pSrc, q31_t *pDst, uint32_t numSamples)
```

参数:

*pSrc: 指向复数输入向量
*pDst: 指向输出向量
numSamples: 输入向量的复数元素个数

返回值:

无

缩放和溢出时的行为:

函数实现 1.31 和 1.31 乘法，最后输出的结果转换为 2.30 格式. 输入不需要向下缩放.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

3.4 复数幅度平方

3.4.1 函数

- *csky_vdsp2_cmplx_mag_squared_q15*：Q15 复数幅度平方
- *csky_vdsp2_cmplx_mag_squared_q31*：Q31 复数幅度平方
- *csky_vdsp2_cmplx_mag_squared_q31_basic*：Q31 复数幅度平方，且保留全部精度

3.4.2 简要说明

计算复数向量元素的幅度平方.

pSrc 指向源数据, pDst 指向结果写入的地址. numSamples 指定复数元素的个数, 复数数据是交错方式保存的 (real, imag, real, imag, …). 输入向量总共有 2*numSamples 个值; 输出向量总共有 numSamples 个值.

使用的算法如下:

```
for (n = 0; n < numSamples; n++) {
    pDst[n] = pSrc[(2 * n) + 0] ^ 2 + pSrc[(2 * n) + 1] ^ 2;
}
```

为 Q15 和 Q31 类型都提供了不同的函数.

3.4.3 函数说明

3.4.3.1 csky_vdsp2_cmplx_mag_squared_q15

```
void csky_vdsp2_cmplx_mag_squared_q15 (q15_t *pSrc, q15_t *pDst, uint32_t numSamples)
```

参数:

`*pSrc`: 指向输入的复数向量

`*pDst`: 指向输出的向量

`numSamples`: 输入向量中的复数数量

返回值:

无

缩放和溢出时的行为:

函数实现了 1.15 和 1.15 的乘法，最后将结果转换为 3.13 格式输出.

3.4.3.2 csky_vdsp2_cmplx_mag_squared_q31

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

```
void csky_vdsp2_cmplx_mag_squared_q31 (q31_t *pSrc, q31_t *pDst, uint32_t numSamples)
```

参数:

`*pSrc`: 指向输入的复数向量
`*pDst`: 指向输出的向量
`numSamples`: 输入向量中的复数数量

返回值:

无

缩放和溢出时的行为:

函数实现了 1.31 和 1.31 的乘法，最后转换成 3.29 格式输出. 输入不需要向下缩放.

3.4.3.3 csky_vdsp2_cmplx_mag_squared_q31_basic

```
void csky_vdsp2_cmplx_mag_squared_q31_basic (q31_t *pSrc, q63_t *pDst, uint32_t numSamples)
```

参数:

`*pSrc`: 指向输入的复数向量
`*pDst`: 指向输出的向量
`numSamples`: 输入向量中的复数数量

返回值:

无

缩放和溢出时的行为:

函数实现了 1.31 和 1.31 的乘法，并保留了全部精度，其输出格式为 2.62.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

3.5 复数与复数相乘

3.5.1 函数

- *csky_vdsp2_cmplx_mult_cmplx_q15*：Q15 复数相乘
- *csky_vdsp2_cmplx_mult_cmplx_q31*：Q31 复数相乘

3.5.2 简要说明

将一个复向量与另一个复向量相乘，并生成复数结果. 复数向量中数据是交错方式保存的 (real, imag, real, imag, ···). 参数 numSamples 表示复数元素的数量. 复数向量总共有 2*numSamples 个值

使用的算法如下:

```
for(n = 0; n < numSamples; n++) {
    pDst[(2 * n) + 0] = pSrcA[(2 * n) + 0] * pSrcB[(2 * n) + 0] - pSrcA[(2 * n) + 1] * pSrcB[(2 *␣
↪n) + 1];
    pDst[(2 * n) + 1] = pSrcA[(2 * n) + 0] * pSrcB[(2 * n) + 1] + pSrcA[(2 * n) + 1] * pSrcB[(2 *␣
↪n) + 0];
}
```

为 Q15 和 Q31 类型都提供了不同的函数.

3.5.3 函数说明

3.5.3.1 csky_vdsp2_cmplx_mult_cmplx_q15

```
void csky_vdsp2_cmplx_mult_cmplx_q15 (q15_t *pSrcA, q15_t *pSrcB, q15_t *pDst, uint32_t numSamples)
```

参数:

`*pSrcA`: 指向第一个输入向量
`*pSrcB`: 指向第二个输入向量
`*pDst`: 指向输出向量
`numSamples`: 向量中的复数元素个数

返回值:

无

缩放和溢出时的行为:

函数实现了 1.15 和 1.15 乘法，最后的结果转换为 3.13 格式输出.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

3.5.3.2 csky_vdsp2_cmplx_mult_cmplx_q31

```
void csky_vdsp2_cmplx_mult_cmplx_q31 (q31_t *pSrcA, q31_t *pSrcB, q31_t *pDst, uint32_t numSamples)
```

参数:

*pSrcA: 指向第一个输入向量

*pSrcB: 指向第二个输入向量

*pDst: 指向输出向量

numSamples: 向量中的复数元素个数

返回值:

无

缩放和溢出时的行为:

函数实现了 1.31 和 1.31 乘法，最后结果转换为 3.29 格式输出. 输入不需要向下缩放.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

3.6 复数与实数相乘

3.6.1 函数

- *csky_vdsp2_cmplx_mult_real_q15*：Q15 复数和实数相乘
- *csky_vdsp2_cmplx_mult_real_q31*：Q31 复数和实数相乘

3.6.2 简要说明

将一个复数向量与一个实数向量相乘，并生成一个复数向量. 复数向量数据是交错方式保存的 (real, imag, real, imag, ···). 参数 numSamples 表示处理的复数元素数量。复数向量总共有 2*numSamples 个值，实数向量总共有 numSamples 个值。

使用的算法如下:

```
for (n=0; n<numSamples; n++) {
    pCmplxDst[(2*n)+0] = pSrcCmplx[(2*n)+0] * pSrcReal[n];
    pCmplxDst[(2*n)+1] = pSrcCmplx[(2*n)+1] * pSrcReal[n];
}
```

为 Q15 和 Q31 类型都提供了不同的函数.

3.6.3 函数说明

3.6.3.1 csky_vdsp2_cmplx_mult_real_q15

```
void csky_vdsp2_cmplx_mult_real_q15 (q15_t *pSrcCmplx, q15_t *pSrcReal, q15_t *pCmplxDst, uint32_t␣
↪numSamples)
```

参数:

`*pSrcCmplx`: 指向输入的复数向量

`*pSrcReal`: 指向输入的实数向量

`*pCmplxDst`: 指向输出的复数向量

`numSamples`: 向量中的元素数量

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. 结果超出 Q15 的最大范围 [0x8000 0x7FFF] 时会被饱和处理.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

3.6.3.2 csky_vdsp2_cmplx_mult_real_q31

```
void csky_vdsp2_cmplx_mult_real_q31 (q31_t *pSrcCmplx, q31_t *pSrcReal, q31_t *pCmplxDst, uint32_t␣
↪numSamples)
```

参数:

*pSrcCmplx: 指向输入的复数向量

*pSrcReal: 指向输入的实数向量

*pCmplxDst: 指向输出的复数向量

numSamples: 向量中的元素数量

返回值:

无

缩放和溢出时的行为:

函数使用饱和算法. 结果超出 Q15 的最大范围 [0x80000000 0x7FFFFFFF] 时会被饱和处理.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

第四章　滤波函数

4.1　卷积

4.1.1　函数

- *csky_vdsp2_conv_q31*：Q31 序列的卷积
- *csky_vdsp2_conv_q15*：Q15 序列的卷积
- *csky_vdsp2_conv_q7*：Q7 序列的卷积
- *csky_vdsp2_conv_fast_q31*：Q31 序列的卷积
- *csky_vdsp2_conv_fast_q15*：Q15 序列的卷积
- *csky_vdsp2_conv_fast_opt_q15*：Q15 序列的卷积
- *csky_vdsp2_conv_opt_q15*：Q15 序列的卷积
- *csky_vdsp2_conv_opt_q7*：Q7 序列的卷积

4.1.2　简要说明

卷积是一种控制两个有限向量生成一个有限向量的数学操作。卷积类似于相关分析，经常用在过滤和数据分析。CSI DSP 库包括了 Q7，Q15，Q31 数据类型的卷积函数。

算法

令 a[n] 和 b[n] 分别是长度 srcALen 和 srcBLen 的样本序列. 则卷积是

```
c[n] = a[n] * b[n]
```

如下定义

$$c[n] = \sum_{k=0}^{srcALen} a[k] b[n-k]$$

注意：c[n] 的长度是 srcALen + srcBLen - 1，并且在区间 n=0, 1, 2, ⋯, srcALen + srcBLen - 2 内. pSrcA 指向第一个输入长度向量 srcALen，pSrcB 指向第二个输入长度向量 srcBLen. 输出结果写入到 pDst，调用函数必须为结果分配 srcALen+srcBLen-1 个字的空间。

概念上，当两个信号 a[n] 和 b[n] 卷积时，信号 b[n] 在 a[n] 上滑动。对于每个偏移 n ，a[n] 和 b[n] 的重叠部分被相乘并相加在一起。

注意，卷积是可交换的操作:

```
a[n] * b[n] = b[n] * a[n].
```

这意味着交换 A 和 B 参数不会对卷积函数造成影响。

定点行为

卷积操作会产生大量的中间结果。因此，Q7，Q15 和 Q31 函数会有溢出和饱和的风险。参考每个函数各自的文档说明，使用的算法的细节情况。

快速版本

Q31 和 Q15 支持快速版本。快速版本需要的周期数更少，但是需要缩放输入信号，确保不会引起中间值溢出

Opt 版本

Q15 和 Q7 有 Opt 版本. 设计使用内部 buffer 来达到更好的优化效果。这些版本优化了速度，但是消耗更多的内存。

4.1.3 函数说明

4.1.3.1 csky_vdsp2_conv_q15

```
void csky_vdsp2_conv_q15 (q15_t *pSrcA, uint32_t srcALen, q15_t *pSrcB, uint32_t srcBLen, q15_t
↪*pDst)
```

参数:

`*pSrcA`: 指向第一个输入序列

`srcALen`: 第一个输入序列的长度

`*pSrcB`: 指向第二个输入序列

`srcBLen`: 第二个输入序列的长度

`*pDst`: 指向输出结果的地址，长度是 srcALen+srcBLen-1

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。输入都表示为 1.15 格式，相乘的结果是 2.30 格式。2.30 的中间结果在 34.30 格式的 64 位累加器中累加。由于有 33 个守护位，所以应该不会有溢出 34.30 格式的结果丢弃低 15 位，截断为 34.15 格式，然后再饱和为 1.15 格式。

函数*csky_vdsp2_conv_fast_q15()* 是一个快速版本，但是丢失了更多的精度。

函数*csky_vdsp2_conv_opt_q15()* 是一个快速版本，使用了额外的临时缓存空间。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.1.3.2 csky_vdsp2_conv_q31

```
void csky_vdsp2_conv_q31 (q31_t *pSrcA, uint32_t srcALen, q31_t *pSrcB, uint32_t srcBLen, q31_t
↪*pDst)
```

参数:

*pSrcA：指向第一个输入序列

srcALen：第一个输入序列的长度

*pSrcB：指向第二个输入序列

srcBLen：第二个输入序列的长度

*pDst：指向输出结果的地址，长度是 srcALen+srcBLen-1

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。累加器使用 2.62 格式维持所有的中间乘法结果的精度，但是只有一个保护位累加过程中没有饱和操作。如果累加器溢出，会往符号位溢出，扭曲结果。因此，输入信号需要缩小来防止溢出。因为加法过程中最多会有 min(srcALen, srcBLen) 次进位，所以，输入需要缩小 log2(min(srcALen, srcBLen)) (log2 是 2 为底的对数) 倍来防止溢出。2.62 格式的累加器，右移 31 位，再饱和生成 1.31 格式的最后结果。

csky_vdsp2_conv_fast_q31() 是一个快速版本，但是丢失了更多精度。

4.1.3.3 csky_vdsp2_conv_q7

```
void csky_vdsp2_conv_q7 (q7_t *pSrcA, uint32_t srcALen, q7_t *pSrcB, uint32_t srcBLen, q7_t *pDst)
```

参数:

*pSrcA：指向第一个输入序列

srcALen：第一个输入序列的长度

*pSrcB：指向第二个输入序列

srcBLen：第二个输入序列的长度

*pDst：指向输出结果的地址，长度是 srcALen+srcBLen-1

返回值:

无

缩放和溢出行为:

函数使用了一个内部 32 位累加器。输入都用 1.7 格式表示，相乘的结果是 2.14 格式。2.14 的中间结果在 18.14 格式的 32 位累加器中累加。由于有 17 个守护位，除非 max(srcALen, srcBLen) 大于 131072，否则不会溢出。18.14 格式的结果丢弃低 7 位，截断为 18.7 格式，然后再饱和成 1.7 格式。

函数*csky_vdsp2_conv_opt_q7()* 是这个函数的一个快速版本。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.1.3.4 csky_vdsp2_conv_fast_opt_q15

```
void csky_vdsp2_conv_fast_opt_q15 (q15_t *pSrcA, uint32_t srcALen, q15_t *pSrcB, uint32_t srcBLen,␣
↪q15_t *pDst, q15_t *pScratch1, q15_t *pScratch2)
```

参数:

*pSrcA：指向第一个输入序列

srcALen：第一个输入序列的长度

*pSrcB：指向第二个输入序列

srcBLen：第二个输入序列的长度

*pDst：指向输出结果的地址，长度是 srcALen+srcBLen-1

*pScratch1 指向临时 buffer，大小是 max(srcALen, srcBLen) + 2*min(srcALen, srcBLen) - 2

*pScratch2 指向临时 buffer，大小是 min(srcALen, srcBLen)

返回值:

无

限制:

如果芯片不支持分对齐访问，输入，输出，临时 buffer，都应该是 32 位对齐

缩放和溢出行为:

这个快速版本使用一个格式为 2.30 的 32 位累加器。累加器维持了中间相乘结果的所有精度，但是只有一个保护位。因为中间加法没有饱和操作，所以，累加器如果溢出的话，会往符号位溢出，扭曲结果。

输入信号需要缩放到防止中间结果溢出。因为，最多可能有 min(srcALen, srcBLen) 个内部加法进位，所以输入需要缩放 log2(min(srcALen, srcBLen)) (log2 是 2 为底的对数) 倍。2.30 格式的累加器右移 15 位，并且饱和到 1.15 格式生成最后的结果。

函数*csky_vdsp2_conv_q15()* 是一个慢速实现，使用了 64 位累加器，来防止精度丢失。

4.1.3.5 csky_vdsp2_conv_fast_q15

```
void csky_vdsp2_conv_fast_q15 (q15_t *pSrcA, uint32_t srcALen, q15_t *pSrcB, uint32_t srcBLen, q15_
↪t *pDst)
```

参数:

*pSrcA：指向第一个输入序列

srcALen：第一个输入序列的长度

*pSrcB：指向第二个输入序列

srcBLen：第二个输入序列的长度

*pDst：指向输出结果的地址，长度是 srcALen+srcBLen-1

返回值:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

无

缩放和溢出行为:

这个快速版本使用一个格式为 2.30 的 32 位累加器。累加器维持了中间相乘结果的所有精度，但是只有一个保护位。因为中间加法没有饱和操作，所以，累加器如果溢出的话，会往符号位溢出，扭曲结果。

输入信号需要缩放到防止中间结果溢出。因为，最多可能有 min(srcALen, srcBLen) 个内部加法进位，所以输入需要缩放 log2(min(srcALen, srcBLen)) (log2 是 2 为底的对数) 倍。2.30 格式的累加器右移 15 位，并且饱和到 1.15 格式生成最后的结果。

函数*csky_vdsp2_conv_q15()* 是一个慢速实现，使用了 64 位累加器，来防止精度丢失。

4.1.3.6 csky_vdsp2_conv_fast_q31

```
void csky_vdsp2_conv_fast_q31 (q31_t *pSrcA, uint32_t srcALen, q31_t *pSrcB, uint32_t srcBLen, q31_
↪t *pDst)
```

参数:

`*pSrcA`：指向第一个输入序列

`srcALen`：第一个输入序列的长度

`*pSrcB`：指向第二个输入序列

`srcBLen`：第二个输入序列的长度

`*pDst`：指向输出结果的地址，长度是 srcALen+srcBLen-1

返回值:

无

缩放和溢出行为:

该函数针对速度进行了优化，牺牲了定点精度和溢出保护. 每个 1.31 和 1.31 相乘的结果截断为 2.30 格式. 这些中间结果以 2.30 格式在一个 32 位寄存器中累加. 最后，累加器饱和转换为 1.31 结果.

快速版本具有与标准版本相同的溢出行为，但提供较小的精度，因为它丢弃每个乘法结果的低 32 位为了避免溢出，输入信号必须缩放。因为，最多可能有 min(srcALen, srcBLen) 个内部加法进位，所以输入需要缩放 log2(min(srcALen, srcBLen)) (log2 是 2 为底的对数) 倍。

见*csky_vdsp2_conv_q31()* ，这个函数是一个慢速实现，使用了 64 位累加器，来防止精度丢失。

4.1.3.7 csky_vdsp2_conv_opt_q15

```
void csky_vdsp2_conv_opt_q15 (q15_t *pSrcA, uint32_t srcALen, q15_t *pSrcB, uint32_t srcBLen, q15_
↪t *pDst, q15_t *pScratch1, q15_t *pScratch2)
```

参数:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

`*pSrcA`: 指向第一个输入序列

`srcALen`: 第一个输入序列的长度

`*pSrcB`: 指向第二个输入序列

`srcBLen`: 第二个输入序列的长度

`*pDst`: 指向输出结果的地址，长度是 srcALen+srcBLen-1

`*pScratch1` 指向临时 buffer，大小是 max(srcALen, srcBLen) + 2*min(srcALen, srcBLen) - 2

`*pScratch2` 指向临时 buffer，大小是 min(srcALen, srcBLen)

返回值:

无

限制:

如果芯片不支持分对齐访问，输入，输出，临时 buffer，都应该是 32 位对齐。

缩放和溢出行为:

函数实现使用了一个内部 64 位内部累加器。输入都是 1.15 格式，相乘结果是 2.30 格式。2.30 的中间结果在 64 位累加器中以 34.30 格式保存。这种方法提供了 33 个保护位，因此没有溢出的风险。最后，34.30 格式的结果丢弃低 15 位截断为 34.15 格式，然后饱和成 1.15 格式的结果。

参考 *csky_vdsp2_conv_fast_q15()* ，是一个快速，但是降低了精度的实现版本。

4.1.3.8 csky_vdsp2_conv_opt_q7

```
void csky_vdsp2_conv_opt_q7 (q7_t *pSrcA, uint32_t srcALen, q7_t *pSrcB, uint32_t srcBLen, q7_t␣
↪*pDst, q15_t *pScratch1, q15_t *pScratch2)
```

参数:

`*pSrcA`: 指向第一个输入序列

`srcALen`: 第一个输入序列的长度

`*pSrcB`: 指向第二个输入序列

`srcBLen`: 第二个输入序列的长度

`*pDst`: 指向输出结果的地址，长度是 srcALen+srcBLen-1

`*pScratch1` 指向临时 buffer，大小是 max(srcALen, srcBLen) + 2*min(srcALen, srcBLen) - 2

`*pScratch2` 指向临时 buffer，大小是 min(srcALen, srcBLen)

返回值:

无

限制:

如果芯片不支持分对齐访问，输入，输出，临时 buffer，都应该是 32 位对齐。

缩放和溢出行为:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

函数实现使用了一个内部 32 位累加器。输入都表示为 1.7 格式，相乘的结果是 2.14 格式。2.14 的中间结果在 18.14 格式的 32 位累加器中累加。由于有 17 个守护位，除非 max(srcALen, srcBLen) 大于 131072，否则不会溢出。18.14 格式的结果丢弃低 7 位，截断为 18.7 格式，然后再饱和成 1.7 格式。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.2 部分卷积

4.2.1 函数

- *csky_vdsp2_conv_partial_q31*：Q31 序列的部分卷积
- *csky_vdsp2_conv_partial_q15*：Q15 序列的部分卷积
- *csky_vdsp2_conv_partial_q7*：Q7 序列的部分卷积
- *csky_vdsp2_conv_partial_fast_q31*：Q31 序列的部分卷积
- *csky_vdsp2_conv_partial_fast_q15*：Q15 序列的部分卷积
- *csky_vdsp2_conv_partial_fast_opt_q15*：Q15 序列的部分卷积
- *csky_vdsp2_conv_partial_opt_q15*：Q15 序列的部分卷积
- *csky_vdsp2_conv_partial_opt_q7*：Q7 序列的部分卷积

4.2.2 简要说明

部分卷积等价于卷积，只是生成的输出样本是卷积的子集。每个函数有两个额外添加的参数。firstIndex 指定输出样本子集的开始序号。numPoints 是需要计算的输出样本的数量函数计算的输出范围在：[firstIndex, …, firstIndex+numPoints-1]. 输出数组 pDst 包括 numPoints 个值.

可选的输出范围在 [0 srcALen+srcBLen-2]. 如果请求的自己并不在这个范围，则函数返回 CSKY_MATH_ARGUMENT_ERROR. 否则函数返回 CSKY_MATH_SUCCESS.

定点行为

定点的行为参考卷积。

快速版本

Q31 和 Q15 的部分卷积有快速版本。快速版本需要的周期更多，但是，设计为需要输入信号缩放到不会引起中间计算发生溢出。

Opt 版本

Q15 和 Q7 有 Opt 版本. 设计使用内部 buffer 来达到更好的优化效果。这些版本优化了速度，但是相对的消耗更多的内存。

4.2.3 函数说明

4.2.3.1 csky_vdsp2_conv_partial_q31

```
csky_vdsp2_status csky_vdsp2_conv_partial_q31 (q31_t *pSrcA, uint32_t srcALen, q31_t *pSrcB,␣
↪uint32_t srcBLen, q31_t *pDst, uint32_t firstIndex, uint32_t numPoints)
```

参数:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

`*pSrcA`：指向第一个输入序列

`srcALen`：第一个输入序列的长度

`*pSrcB`：指向第二个输入序列

`srcBLen`：第二个输入序列的长度

`*pDst`：指向输出结果的地址

`firstIndex`：输出样本的起始索引

`numPoints`：输出样本的数量

返回值:

函数正确完成返回 CSKY_MATH_SUCCESS ，如果请求的子集超出范围 [0 srcALen+srcBLen-2]，则返回 CSKY_MATH_ARGUMENT_ERROR。

4.2.3.2 csky_vdsp2_conv_partial_q15

```
csky_vdsp2_status csky_vdsp2_conv_partial_q15 (q15_t *pSrcA, uint32_t srcALen, q15_t *pSrcB,␣
↪uint32_t srcBLen, q15_t *pDst, uint32_t firstIndex, uint32_t numPoints)
```

参数:

`*pSrcA`：指向第一个输入序列

`srcALen`：第一个输入序列的长度

`*pSrcB`：指向第二个输入序列

`srcBLen`：第二个输入序列的长度

`*pDst`：指向输出结果的地址

`firstIndex`：输出样本的起始索引

`numPoints`：输出样本的数量

返回值:

函数正确完成返回 CSKY_MATH_SUCCESS ，如果请求的子集超出范围 [0 srcALen+srcBLen-2]，则返回 CSKY_MATH_ARGUMENT_ERROR

4.2.3.3 csky_vdsp2_conv_partial_q7

```
csky_vdsp2_status csky_vdsp2_conv_partial_q7 (q7_t *pSrcA, uint32_t srcALen, q7_t *pSrcB, uint32_t␣
↪srcBLen, q7_t *pDst, uint32_t firstIndex, uint32_t numPoints)
```

参数:

`*pSrcA`：指向第一个输入序列

`srcALen`：第一个输入序列的长度

`*pSrcB`：指向第二个输入序列

`srcBLen`：第二个输入序列的长度

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

*pDst：指向输出结果的地址

firstIndex：输出样本的起始索引

numPoints：输出样本的数量

返回值:

函数正确完成返回 CSKY_MATH_SUCCESS ，如果请求的子集超出范围 [0 srcALen+srcBLen-2]，则返回 CSKY_MATH_ARGUMENT_ERROR

4.2.3.4 csky_vdsp2_conv_partial_fast_opt_q15

```
csky_vdsp2_status csky_vdsp2_conv_partial_fast_opt_q15 (q15_t *pSrcA, uint32_t srcALen, q15_t␣
↪*pSrcB, uint32_t srcBLen, q15_t *pDst, uint32_t firstIndex, uint32_t numPoints, q15_t *pScratch1,
↪ q15_t *pScratch2)
```

参数:

*pSrcA：指向第一个输入序列

srcALen：第一个输入序列的长度

*pSrcB：指向第二个输入序列

srcBLen：第二个输入序列的长度

*pDst：指向输出结果的地址

firstIndex：输出样本的起始索引

numPoints：输出样本的数量

*pScratch1 指向缓存地址，大小为 max(srcALen, srcBLen) + 2*min(srcALen, srcBLen) - 2

*pScratch2 指向缓存地址，大小为 min(srcALen, srcBLen)

返回值:

函数正确完成返回 CSKY_MATH_SUCCESS ，如果请求的子集超出范围 [0 srcALen+srcBLen-2]，则返回 CSKY_MATH_ARGUMENT_ERROR

限制:

如果芯片不支持分对齐访问，输入，输出，临时 buffer，都应该是 32 位对齐。

4.2.3.5 csky_vdsp2_conv_partial_fast_q15

```
csky_vdsp2_status csky_vdsp2_conv_partial_fast_q15 (q15_t *pSrcA, uint32_t srcALen, q15_t *pSrcB,␣
↪uint32_t srcBLen, q15_t *pDst, uint32_t firstIndex, uint32_t numPoints)
```

参数:

*pSrcA：指向第一个输入序列

srcALen：第一个输入序列的长度

*pSrcB：指向第二个输入序列

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

srcBLen：第二个输入序列的长度

*pDst：指向输出结果的地址

firstIndex：输出样本的起始索引

numPoints：输出样本的数量

返回值:

函数正确完成返回 CSKY_MATH_SUCCESS，如果请求的子集超出范围 [0 srcALen+srcBLen-2]，则返回 CSKY_MATH_ARGUMENT_ERROR

4.2.3.6 csky_vdsp2_conv_partial_fast_q31

```
csky_vdsp2_status csky_vdsp2_conv_partial_fast_q31 (q31_t *pSrcA, uint32_t srcALen, q31_t *pSrcB,
↪uint32_t srcBLen, q31_t *pDst, uint32_t firstIndex, uint32_t numPoints)
```

参数:

*pSrcA：指向第一个输入序列

srcALen：第一个输入序列的长度

*pSrcB：指向第二个输入序列

srcBLen：第二个输入序列的长度

*pDst：指向输出结果的地址

firstIndex：输出样本的起始索引

numPoints：输出样本的数量

返回值:

函数正确完成返回 CSKY_MATH_SUCCESS，如果请求的子集超出范围 [0 srcALen+srcBLen-2]，则返回 CSKY_MATH_ARGUMENT_ERROR

4.2.3.7 csky_vdsp2_conv_partial_opt_q15

```
csky_vdsp2_status csky_vdsp2_conv_partial_opt_q15 (q15_t *pSrcA, uint32_t srcALen, q15_t *pSrcB,
↪uint32_t srcBLen, q15_t *pDst, uint32_t firstIndex, uint32_t numPoints, q15_t *pScratch1, q15_t
↪*pScratch2)
```

参数:

*pSrcA：指向第一个输入序列

srcALen：第一个输入序列的长度

*pSrcB：指向第二个输入序列

srcBLen：第二个输入序列的长度.

*pDst：指向输出结果的地址

firstIndex：输出样本的起始索引

numPoints：输出样本的数量

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

`*pScratch1` 指向临时缓存，大小为 max(srcALen, srcBLen) + 2*min(srcALen, srcBLen) - 2

`*pScratch2` 指向临时缓存，大小为 min(srcALen, srcBLen)

返回值:

函数正确完成返回 CSKY_MATH_SUCCESS ，如果请求的子集超出范围 [0 srcALen+srcBLen-2]，则返回 CSKY_MATH_ARGUMENT_ERROR

限制:

如果芯片不支持分对齐访问，输入，输出，临时 buffer，都应该是 32 位对齐。

4.2.3.8 csky_vdsp2_conv_partial_opt_q7

```
csky_vdsp2_conv_partial_opt_q7 (q7_t *pSrcA, uint32_t srcALen, q7_t *pSrcB, uint32_t srcBLen, q7_t
↪*pDst, uint32_t firstIndex, uint32_t numPoints, q15_t *pScratch1, q15_t *pScratch2)
```

参数:

`*pSrcA`: 指向第一个输入序列

`srcALen`: 第一个输入序列的长度

`*pSrcB`: 指向第二个输入序列

`srcBLen`: 第二个输入序列的长度.

`*pDst`: 指向输出结果的地址

`firstIndex`: 输出样本的起始索引

`numPoints`: 输出样本的数量

`*pScratch1` 指向临时缓存，大小为 max(srcALen, srcBLen) + 2*min(srcALen, srcBLen) - 2

`*pScratch2` 指向临时缓存，大小为 min(srcALen, srcBLen)

返回值:

函数正确完成返回 CSKY_MATH_SUCCESS ，如果请求的子集超出范围 [0 srcALen+srcBLen-2]，则返回 CSKY_MATH_ARGUMENT_ERROR

限制:

如果芯片不支持分对齐访问，输入，输出，临时 buffer，都应该是 32 位对齐。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.3 相关分析

4.3.1 函数

- *csky_vdsp2_correlate_q31*：Q31 序列的相关
- *csky_vdsp2_correlate_q15*：Q15 序列的相关
- *csky_vdsp2_correlate_q7*：Q7 序列的相关
- *csky_vdsp2_correlate_fast_q31*：Q31 序列的相关
- *csky_vdsp2_correlate_fast_q15*：Q15 序列的相关
- *csky_vdsp2_correlate_fast_opt_q15*：Q15 序列的相关
- *csky_vdsp2_correlate_opt_q15*：Q15 序列的相关
- *csky_vdsp2_correlate_opt_q7*：Q7 序列的相关

4.3.2 简要说明

相关是一种与卷积类似的数学操作。跟卷积一样，相关用两个信号生成一个新的信号。相关和卷积中的基本算法是相同的，除了其中一个输入在卷积中被翻转。相关一般用来测量两个信号的相似性。广泛应用在模式识别，密码分析和搜索。CSI 库为 Q7, Q15, Q31 数据类型提供相关函数。Q15 和 Q31 还提供了快速版本函数。

算法

令 a[n] 和 b[n] 分别是长度为 srcALen 和 srcBLen 样本序列. 两个信号的卷积表示为:

```
c[n] = a[n] * b[n]
```

相关则让其中一个信号翻转:

```
c[n] = a[n] * b[-n]
```

下面是数学定义

$$c[n] = \sum_{k=0}^{srcALen} a[k]\, b[k-n]$$

pSrcA 指向第一个输入序列，序列长度是 srcALen ，pSrcB 指向第二个输入序列，序列长度是 srcBLen. 结果 c[n] 的长度是 2 * max(srcALen, srcBLen) - 1 ，所有结果都落在区间 n=0, 1, 2, ⋯, (2 * max(srcALen, srcBLen) - 2) 中. 输出结果写入到 pDst ，调用函数必须分配好 2 * max(srcALen, srcBLen) - 1 个字保存结果。

注意

pDst 在使用前需要先清零。

定点行为

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

相关会生成大量的中间值。因此，Q7, Q15, 和 Q31 函数有可能会发生溢出和饱和。参考具体函数的文档，了解使用特定算法的详细情况。

快速版本

Q31 和 Q15 支持快速版本。快速版本需要更少的周期数，但是设计为，需要输入信号缩小到不会引起中间结果溢出。

Opt 版本

Q15 和 Q7 支持 Opt 版本。设计使用临时缓存来获取更好的优化效果。这些版本优化了速度，但是相对的消耗了更多的内存 (临时缓存)。

4.3.3 函数说明

4.3.3.1 csky_vdsp2_correlate_q31

```
void csky_vdsp2_correlate_q31 (q31_t *pSrcA, uint32_t srcALen, q31_t *pSrcB, uint32_t srcBLen, q31_
↪t *pDst)
```

参数:

`*pSrcA`: 指向第一个输入序列

`srcALen`: 第一个输入序列的长度

`*pSrcB`: 指向第二个输入序列

`srcBLen`: 第二个输入序列的长度

`*pDst`: 指向输出结果的地址，长度是 2 * max(srcALen, srcBLen) - 1

返回值:

无

缩放和溢出行为:

函数实现使用了一个 64 位内部累加器。累加器使用 2.62 格式，维持了中间乘法结果的所有精度，但是只有一个保护位。累加过程中没有饱和操作。因此，如果累加器溢出，会往符号位溢出，扭曲结果。输入信号需要缩小来防止中间结果溢出。因为加法最多发生 min(srcALen, srcBLen) 次进位，所以需要缩小输入 1/min(srcALen, srcBLen) 来防止溢出。2.62 累加器右移 31 位，然后饱和生成为 1.31 格式的结果。

4.3.3.2 csky_vdsp2_correlate_q15

```
void csky_vdsp2_correlate_q15 (q15_t *pSrcA, uint32_t srcALen, q15_t *pSrcB, uint32_t srcBLen, q15_
↪t *pDst)
```

参数:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

`*pSrcA`：指向第一个输入序列

`srcALen`：第一个输入序列的长度

`*pSrcB`：指向第二个输入序列

`srcBLen`：第二个输入序列的长度

`*pDst`：指向输出结果的地址，长度是 2 * max(srcALen, srcBLen) - 1

返回值:

无

缩放和溢出行为:

函数实现使用了一个 64 位内部累加器。输入都是 1.15 格式，相乘的结果是 2.30 格式。2.30 格式的结果在 34.30 格式的 64 位累加器中累加。由于有 33 个守护位，不会有溢出风险。34.30 格式的结果丢弃低 15 位，截断为 34.15 格式，然后再饱和成 1.15 格式。

4.3.3.3 csky_vdsp2_correlate_q7

```
void csky_vdsp2_correlate_q7 (q7_t *pSrcA, uint32_t srcALen, q7_t *pSrcB, uint32_t srcBLen, q7_t␣
↪*pDst)
```

参数:

`*pSrcA`：指向第一个输入序列

`srcALen`：第一个输入序列的长度

`*pSrcB`：指向第二个输入序列

`srcBLen`：第二个输入序列的长度

`*pDst`：指向输出结果的地址，长度是 2 * max(srcALen, srcBLen) - 1

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 32 位累加器。输入都是 1.7 格式，相乘的结果是 2.14 格式。2.14 格式的结果在 18.14 格式的 32 位累加器中累加。由于有 17 个保护位，除非 max(srcALen, srcBLen) 大于 131072，否则不会溢出 18.14 的结果丢弃低 7 位，截断为 18.7 格式，最后饱和为 1.7 格式。

4.3.3.4 csky_vdsp2_correlate_fast_opt_q15

```
void csky_vdsp2_correlate_fast_opt_q15 (q15_t *pSrcA, uint32_t srcALen, q15_t *pSrcB, uint32_t␣
↪srcBLen, q15_t *pDst, q15_t *pScratch)
```

参数:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

`*pSrcA`: 指向第一个输入序列

`srcALen`: 第一个输入序列的长度

`*pSrcB`: 指向第二个输入序列

`srcBLen`: 第二个输入序列的长度

`*pDst`: 指向输出结果的地址，长度是 2 * max(srcALen, srcBLen) - 1

`*pScratch` 指向临时缓存，大小是 max(srcALen, srcBLen) + 2*min(srcALen, srcBLen) - 2.

返回值:

无

限制:

如果芯片不支持分对齐访问，输入，输出，临时 buffer，都应该是 32 位对齐。

缩放和溢出行为:

这个快速版本使用了一个 2.30 格式的 32 位累加器。累加器维持了中间乘法结果的全部精度，但是只有 1 个保护位。累加过程中没有饱和操作。因此，如果累加器溢出，会扭曲结果。为了防止中间结果溢出，必须缩小输入信号。因为加法最多会有 min(srcALen, srcBLen) 个进位，所以，输入信号需要缩小 1/min(srcALen, srcBLen) 防止溢出。2.30 格式累加器右移 15 位，然后饱和生成 1.15 的最后结果。

函数*csky_vdsp2_correlate_q15()* 是这个函数的一个慢速版本，使用了一个 64 位累加器来防止溢出。

4.3.3.5 csky_vdsp2_correlate_fast_q15

```
void csky_vdsp2_correlate_fast_q15 (q15_t *pSrcA, uint32_t srcALen, q15_t *pSrcB, uint32_t srcBLen,
↪ q15_t *pDst)
```

参数:

`*pSrcA`: 指向第一个输入序列

`srcALen`: 第一个输入序列的长度

`*pSrcB`: 指向第二个输入序列

`srcBLen`: 第二个输入序列的长度

`*pDst`: 指向输出结果的地址，长度是 2 * max(srcALen, srcBLen) - 1

返回值:

无

缩放和溢出行为:

这个快速版本使用一个 2.30 格式的 32 位累加器。累加器维持了中间乘法结果的全部精度，但是只有 1 个保护位。累加过程中没有饱和操作。因此，如果累加器溢出，会扭曲结果。为了防止中间结果溢出，必须缩小输入信号。因为加法最多会有 min(srcALen, srcBLen) 个进位，所以，输入信号需要缩小 1/min(srcALen, srcBLen) 防止溢出。2.30 格式累加器右移 15 位，然后饱和生成 1.15 的最后结果。

函数*csky_vdsp2_correlate_q15()* 是这个函数的一个慢速版本，使用了一个 64 位累加器来防止溢出。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.3.3.6 csky_vdsp2_correlate_fast_q31

```
void csky_vdsp2_correlate_fast_q31 (q31_t *pSrcA, uint32_t srcALen, q31_t *pSrcB, uint32_t srcBLen,
↪ q31_t *pDst)
```

参数:

*pSrcA：指向第一个输入序列

srcALen：第一个输入序列的长度

*pSrcB：指向第二个输入序列

srcBLen：第二个输入序列的长度

*pDst：指向输出结果的地址，长度是 2 * max(srcALen, srcBLen) - 1

返回值:

无

缩放和溢出行为:

该函数针对速度进行了优化，牺牲了定点精度和溢出保护。每个 1.31 和 1.31 乘法的结果截断为 2.30 格式. 中间结果在 32 位寄存器中累加为 2.30 格式的结果。最后，累加器饱和和转换为 1.31 的结果。

快速版本和标准版本的溢出行为一样，不过，丢弃低 32 位，导致精度会更低。为了避免中间结果溢出，输入信号必须缩放。因为加法最多发生 min(srcALen, srcBLen) 次进位，所以需要缩小输入 1/min(srcALen, srcBLen) 来防止溢出。

函数*csky_vdsp2_correlate_q31()* 是这个函数的一个慢速版本，使用了 64 位累加器提供更高的精度。

4.3.3.7 csky_vdsp2_correlate_opt_q15

```
void csky_vdsp2_correlate_opt_q15 (q15_t *pSrcA, uint32_t srcALen, q15_t *pSrcB, uint32_t srcBLen,␣
↪q15_t *pDst, q15_t *pScratch)
```

参数:

*pSrcA：指向第一个输入序列

srcALen：第一个输入序列的长度

*pSrcB：指向第二个输入序列

srcBLen：第二个输入序列的长度

*pDst：指向输出结果的地址，长度是 2 * max(srcALen, srcBLen) - 1

*pScratch 指向临时缓存，大小是 max(srcALen, srcBLen) + 2*min(srcALen, srcBLen) - 2.

返回值:

无

限制:

如果芯片不支持分对齐访问，输入，输出，临时 buffer，都应该是 32 位对齐。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。输入都表示为 1.15 格式，相乘的结果是 2.30 格式。2.30 的中间结果在 34.30 格式的 64 位累加器中累加。由于有 33 个守护位，不会有溢出风险。34.30 格式的结果丢弃低 15 位，截断为 34.15 格式，然后再饱和成 1.15 格式。

4.3.3.8 csky_vdsp2_correlate_opt_q7

```
void csky_vdsp2_correlate_opt_q7 (q7_t *pSrcA, uint32_t srcALen, q7_t *pSrcB, uint32_t srcBLen, q7_
↪t *pDst, q15_t *pScratch1, q15_t *pScratch2)
```

参数:

`*pSrcA`: 指向第一个输入序列

`srcALen`: 第一个输入序列的长度

`*pSrcB`: 指向第二个输入序列

`srcBLen`: 第二个输入序列的长度

`*pDst`: 指向输出结果的地址，长度是 2 * max(srcALen, srcBLen) - 1

`*pScratch1` 指向临时缓存，大小是 max(srcALen, srcBLen) + 2*min(srcALen, srcBLen) - 2.

`*pScratch2` 指向临时 buffer，大小是 min(srcALen, srcBLen).

返回值:

无

限制:

如果芯片不支持分对齐访问，输入，输出，临时 buffer，都应该是 32 位对齐。

缩放和溢出行为:

函数实现使用了一个内部 32 位累加器。输入都表示为 1.7 格式，相乘的结果是 2.14 格式。2.14 的中间结果在 18.14 格式的 32 位累加器中累加。由于有 17 个守护位，除非 max(srcALen, srcBLen) 大于 131072，否则不会溢出。18.14 格式的结果丢弃低 7 位，截断为 18.7 格式，然后再饱和成 1.7 格式。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.4 有限冲激响应（FIR）滤波器

4.4.1 函数

- *csky_vdsp2_fir_q31*：Q31 FIR 滤波器处理函数
- *csky_vdsp2_fir_q15*：Q15 FIR 滤波器处理函数
- *csky_vdsp2_fir_q7*：Q7 FIR 滤波器处理函数
- *csky_vdsp2_fir_fast_q31*：Q31 FIR 滤波器处理函数
- *csky_vdsp2_fir_fast_q15*：Q15 FIR 滤波器处理函数
- *csky_vdsp2_fir_init_q31*：Q31 FIR 滤波器初始化函数
- *csky_vdsp2_fir_init_q15*：Q15 FIR 滤波器初始化函数
- *csky_vdsp2_fir_init_q7*：Q7 FIR 滤波器初始化函数

4.4.2 简要说明

这些函数实现了 Q7，Q15，Q31 数据类型的的有限冲激响应（FIR）滤波器，还有 Q15 和 Q31 的快速版本。还是以块为单位处理输入输出数据，每次调用滤波器函数处理 blockSize 个样本. pSrc 和 pDst 指向输入和输出数组，数组有 blockSize 个值。

算法:

FIR 滤波器的算法建立在一系列的乘加（MAC）操作之上。每个滤波器系数 b[n] 和一个状态变量相乘，状态变量与之前的输入样本 x[n] 相同。

```
y[n] = b[0] * x[n] + b[1] * x[n-1] + b[2] * x[n-2] + ...+ b[numTaps-1] * x[n-numTaps+1]
```

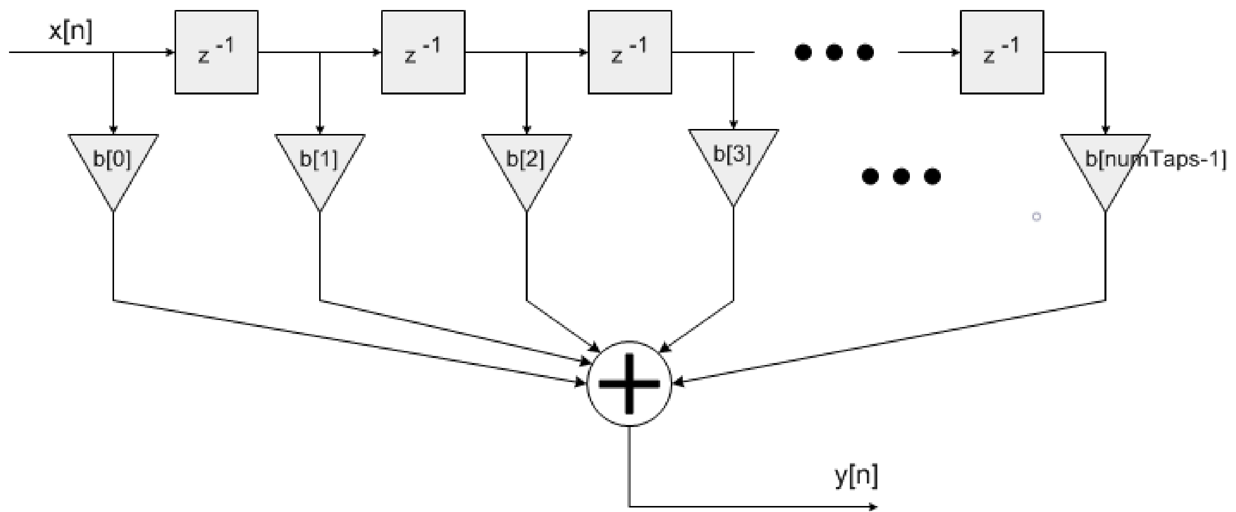

图 4.1: 有限冲激响应滤波器

pCoeffs 指向系数数组，数组的大小是 numTaps. 系数按以下顺序保存:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

```
{b[numTaps-1], b[numTaps-2], b[N-2], ..., b[1], b[0]}
```

pState 指向状态数组，数组的大小是 numTaps + blockSize - 1. 状态数组的样本保存顺序如下：

```
{x[n-numTaps+1], x[n-numTaps], x[n-numTaps-1], x[n-numTaps-2]....x[0], x[1], ...,␣
↪x[blockSize-1]}
```

注意：状态缓存的长度超过了系数数组的长度 blockSize-1. 扩展之后的状态缓存长度，可以避免循环寻址，并且显著提升速度。循环寻址是用在传统 FIR 滤波器的一种寻址方式。状态变量在每块数据操作之后更新，系数不更新。

结构体实例

滤波器的系数和状态变量都保存在数据结构的实例中。每个滤波器都必须有一个单独的结构体实例。系数数组可能可以在几个实例之间共享，但是状态变量数组不能共享。为支持的 4 种数据类型分别提供了不同的结构体实例声明。

初始化函数

There is also an associated initialization function for each data type. The initialization function performs the following operations:

- Sets the values of the internal structure fields.
- Zeros out the values in the state buffer. To do this manually without calling the init function, assign the follow subfields of the 结构体实例: numTaps, pCoeffs, pState. Also set all of the values in pState to zero.

Use of the initialization function is optional. However, if the initialization function is used, then the 结构体实例 cannot be placed into a const data section. To place an 结构体实例 into a const data section, the 结构体实例 must be manually initialized. Set the values in the state buffer to zeros before static initialization. The code below statically initializes each of the 4 different data type filter 结构体实例 s
为每种支持的数据类型都提供了一个相应的初始化函数。初始化函数处理以下操作：

- 设置内部结构体字段的值
- 清零状态缓存中的值如果手动初始化，而不调用初始化函数，需要指定结构体实例的以下字段：numTaps, pCoeffs, pState. pState 中的所有值置 0.

是否使用初始化函数是可选的。但是，使用了初始化函数，则不能将结构体实例放在常量数据段。要将结构体实例放在常量数据段，则必须手动初始化结构体实例。在静态初始化之前，要确保状态缓存中的值已经清零。下面的代码，为 4 种不同的滤波器，静态的初始化了结构体实例。

```
*csky_vdsp2_fir_instance_q31 S = {numTaps, pState, pCoeffs};
*csky_vdsp2_fir_instance_q15 S = {numTaps, pState, pCoeffs};
*csky_vdsp2_fir_instance_q7 S =  {numTaps, pState, pCoeffs};
```

其中 numTaps 是滤波器中的系数数量; pState 是状态缓存的地址; pCoeffs 是系数缓存的地址

定点行为

使用定点 FIR 滤波器函数需要注意。特别是要考虑，在每个函数内使用的累加器的溢出和饱和行为。具体参考每个函数各自的文档和使用说明。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.4.3 函数说明

4.4.3.1 csky_vdsp2_fir_q31

```
void csky_vdsp2_fir_q31 (const csky_vdsp2_fir_instance_q31 *S, q31_t *pSrc, q31_t *pDst, uint32_t␣
↪blockSize)
```

参数:

*S: 指向 FIR 滤波器结构体实例

*pSrc: 指向输入数据

*pDst: 指向输出数据

blockSize: 输入数据的数量

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。输入数据表示为 1.31 格式。中间乘法生成 2.62 格式的结果，结果在 2.62 格式的 64 位累加器累加。因为只有一个保护位，所以需要缩小输入信号来防止溢出。需要缩小 log2(numTaps) 才可以保证没有溢出。最后，2.62 格式右移 31 位，然后饱和生成 1.15 格式的结果。

函数*csky_vdsp2_fir_fast_q31()* 是这个函数的一个快速版本，但是丢失了更多的精度

4.4.3.2 csky_vdsp2_fir_q15

```
void csky_vdsp2_fir_q15 (const csky_vdsp2_fir_instance_q15 *S, q15_t *pSrc, q15_t *pDst, uint32_t␣
↪blockSize)
```

参数:

*S: 指向 FIR 滤波器结构体实例

*pSrc: 指向输入数据

*pDst: 指向输出数据

blockSize: 输入数据的数量

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。输入都是 1.15 格式的，乘法生成 2.30 结果。2.30 格式的中间结果在 34.30 格式的 64 位累加器累加。由于有 33 个保护位，所以不会有溢出风险。34.30 的结果丢弃低 15 位截断为 34.15 格式，然后饱和成为 1.15 格式的结果。

函数*csky_vdsp2_fir_fast_q15()* 是这个函数的一个快速版本，但是丢失了更多的精度

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.4.3.3 csky_vdsp2_fir_q7

```
void csky_vdsp2_fir_q7 (const csky_vdsp2_fir_instance_q7 *S, q7_t *pSrc, q7_t *pDst, uint32_t␣
↪blockSize)
```

参数:

*S：指向 FIR 滤波器结构体实例

*pSrc：指向输入数据

*pDst：指向输出数据

blockSize：输入数据的数量

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 32 位累加器。输入都是 1.7 格式，相乘的结果是 2.14 格式。2.14 格式的结果在 18.14 格式的 32 位累加器中累加。中间结果不会有溢出风险，而且可以保留所有的精度。18.14 的结果丢弃低 7 位，截断为 18.7 格式，最后饱和为 1.7 格式。

4.4.3.4 csky_vdsp2_fir_fast_q15

```
void csky_vdsp2_fir_fast_q15 (const csky_vdsp2_fir_instance_q15 *S, q15_t *pSrc, q15_t *pDst,␣
↪uint32_t blockSize)
```

参数:

*S：指向 FIR 滤波器结构体实例

*pSrc：指向输入数据

*pDst：指向输出数据

blockSize：输入数据的数量

返回值:

无

缩放和溢出行为:

这个快速版本使用一个 2.30 格式的 32 位累加器。累加器维持了中间乘法的所有精度，但是只有一个保护位。因此为了防止溢出，输入信号必须缩小 log2(numTaps) 位. 最后 2.30 格式累加器截断为 2.15 格式，并且饱和为 1.15 格式.

函数*csky_vdsp2_fir_q15()* 是这个函数的一个慢速版本，使用了一个 64 位累加器，防止溢出。慢速和快速版本使用了相同的结构体实例。可以使用函数*csky_vdsp2_fir_init_q15()* 初始化滤波器结构体。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.4.3.5 csky_vdsp2_fir_fast_q31

```
void csky_vdsp2_fir_fast_q31 (const csky_vdsp2_fir_instance_q31 *S, q31_t *pSrc, q31_t *pDst,␣
↪uint32_t blockSize)
```

参数:

*S: 指向 FIR 滤波器结构体实例

*pSrc: 指向输入数据

*pDst: 指向输出数据

blockSize: 输入数据的数量

返回值:

无

缩放和溢出行为:

函数为了优化速度，舍弃了一些定点精度和溢出保护。每个 1.31 和 1.31 相乘的结果截断为 2.30 格式. 这些中间结果在一个 2.30 格式累加器相加。最后，累加器饱和转换为 1.31 的结果。快速版本和标准版本有一样的溢出行为，因为丢弃了每次相乘结果的低 32 位，所以相对提供了更少的精度。为了防止溢出，输入信号必须缩小 log2(numTaps) 个位.

函数*csky_vdsp2_fir_q31()* 是这个函数的一个慢速版本，使用了一个 64 位累加器，提供了更高的精度。慢速和快速版本使用了相同的结构体实例。可以使用函数*csky_vdsp2_fir_init_q31()* 初始化滤波器的结构体.

4.4.3.6 csky_vdsp2_fir_init_q15

```
void csky_vdsp2_fir_init_q15 (csky_vdsp2_fir_instance_q15 *S, uint16_t numTaps, q15_t *pCoeffs,␣
↪q15_t *pState, uint32_t blockSize)
```

参数:

*S: 指向 FIR 滤波器结构体实例

numTaps: 滤波器内系数数量

*pCoeffs: 指向滤波器系数缓存

*pState: 指向状态缓存

blockSize: 输入数据的数量

返回值:

无

简要说明:

pCoeffs 指向滤波器系数数组，系数保存的顺序如下：

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

```
{b[numTaps-1], b[numTaps-2], b[N-2], ..., b[1], b[0]}
```

注意：numTaps 必须是偶数，并且大于等于 4。实现奇数长度的滤波器，则将 numTaps 加 1，然后将最后的系数设成 0。比如，要实现一个滤波器的 numTaps=3 ，系数是：

```
{0.3, -0.8, 0.3}
```

改成 numTaps=4 ，使用系数：

```
{0.3, -0.8, 0.3, 0}.
```

类似的，实现只有两个点的滤波器

```
{0.3, -0.3}
```

改成 numTaps=4 ，使用系数：

```
{0.3, -0.3, 0, 0}.
```

pState 指向状态变量的数组。pState 的长度是 numTaps+blockSize, 其中 blockSize 作为输入样本的数量传给*csky_vdsp2_fir_q31()* 。

4.4.3.7 csky_vdsp2_fir_init_q31

```
void csky_vdsp2_fir_init_q31 (csky_vdsp2_fir_instance_q31 *S, uint16_t numTaps, q31_t *pCoeffs,␣
↪q31_t *pState, uint32_t blockSize)
```

参数:

`*S`：指向 FIR 滤波器结构体实例

`numTaps`：滤波器内系数数量

`*pCoeffs`：指向滤波器系数缓存

`*pState`：指向状态缓存

`blockSize`：输入数据的数量

返回值:

无

简要说明:

pCoeffs 指向滤波器系数数组，系数保存的顺序如下：

```
{b[numTaps-1], b[numTaps-2], b[N-2], ..., b[1], b[0]}
```

pState 指向状态变量的数组。pState 的长度是 numTaps+blockSize-1 , 其中 blockSize 作为输入样本的数量传入*csky_vdsp2_fir_q31()* 。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.4.3.8 csky_vdsp2_fir_init_q7

```
void csky_vdsp2_fir_init_q7 (csky_vdsp2_fir_instance_q7 *S, uint16_t numTaps, q7_t *pCoeffs, q7_t
↪*pState, uint32_t blockSize)
```

参数:

`*S`：指向 FIR 滤波器结构体实例

`numTaps`：滤波器内系数数量

`*pCoeffs`：指向滤波器系数缓存

`*pState`：指向状态缓存

`blockSize`：输入数据的数量

返回值:

无

简要说明:

pCoeffs 指向滤波器系数数组，系数保存的顺序如下：

```
{b[numTaps-1], b[numTaps-2], b[N-2], ..., b[1], b[0]}
```

pState 指向状态变量的数组。pState 的长度是 numTaps+blockSize-1，其中 blockSize 作为输入样本的数量传入*csky_vdsp2_fir_q7()*。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.5 有限冲激响应（FIR）抽取器

4.5.1 函数

- *csky_vdsp2_fir_decimate_q31*：Q31 FIR 抽取处理函数
- *csky_vdsp2_fir_decimate_q15*：Q15 FIR 抽取处理函数
- *csky_vdsp2_fir_decimate_fast_q31*：Q31 FIR 抽取处理函数
- *csky_vdsp2_fir_decimate_fast_q15*：Q15 FIR 抽取处理函数
- *csky_vdsp2_fir_decimate_init_q31*：Q31 FIR 抽取初始化函数
- *csky_vdsp2_fir_decimate_init_q15*：Q15 FIR 抽取初始化函数

4.5.2 简要说明

这些函数将 FIR 滤波器和抽取器组合在一起。它们用于多速率系统中，用于降低信号的采样率而不引入混叠失真。从概念上讲，这些函数等同于下面的框图：

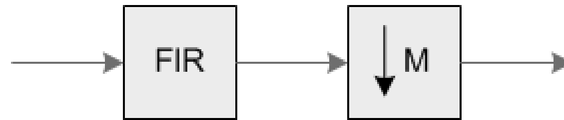

图 4.2: Components included in the FIR Decimator 函数

当用因子 M 抽取时, 信号应该通过 1/M 截止频率的低通滤波器预滤波，防止混叠失真。函数的使用者负责提供滤波器的系数。

CSI DSP 库中的 FIR 抽取器组合了 FIR 滤波器和抽取器。不会计算每个 M 的 FIR 滤波器输出，丢弃 M-1，只计算抽取器的输出样本。函数以块为单位操作输入和输出数据。pSrc 指向输入数组，数组大小是 blockSize，pDst 指向输出数组，数组大小是 blockSize/M . 为了取得整数个输出样本 blockSize， 输入必须是抽取因子 M 的整数倍数。

为 Q15，Q31 分别提供了不同的函数。

算法:

FIR 部分的算法使用的是标准形式过滤器:

```
y[n] = b[0] * x[n] + b[1] * x[n-1] + b[2] * x[n-2] + ...+ b[numTaps-1] * x[n-numTaps+1]
```

其中, b[n] 是滤波器的系数。

pCoeffs 指向系数数组，数组大小是 numTaps. 系数按如下顺序排列保存:

```
{b[numTaps-1], b[numTaps-2], b[N-2], ..., b[1], b[0]}
```

pState 指向状态数组，数组大小是 numTaps + blockSize - 1. 样本在状态缓存中保存的顺序是:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

```
{x[n-numTaps+1], x[n-numTaps], x[n-numTaps-1], x[n-numTaps-2]....x[0], x[1], ...,
↪x[blockSize-1]}
```

状态变量会在每块数据处理后更新，系数不会更新。

结构体实例

滤波器的系数和状态变量都保存在数据结构的实例中。每个滤波器都必须要有一个结构体实例。系数数组可能可以在几个实例之间共享，但是状态变量数组必须单独分配。为 Q31 和 Q15 数据类型分别提供了不同的结构体实例声明。

初始化函数

每种数据类型都有一个相应的初始化函数。初始化函数处理以下操作：

- 设置内部结构体字段的值
- 清零状态缓存
- 确保输入的大小是抽取因子的整数倍如果手动初始化，而不调用初始化函数，需要指定结构体实例的以下字段：numTaps, pCoeffs, M (抽取因子), pState. pState 中的所有值置 0.

是否使用初始化函数是可选的。但是，使用了初始化函数，则不能将结构体实例放在常量数据段。要将结构体实例放在常量数据段，则必须手动初始化结构体实例。下面的代码，为 3 种不同的滤波器，静态的初始化了结构体实例。

```
*csky_vdsp2_fir_decimate_instance_q31 S = {M, numTaps, pCoeffs, pState};
*csky_vdsp2_fir_decimate_instance_q15 S = {M, numTaps, pCoeffs, pState};
```

其中 M 是抽取因子; numTaps 是滤波器中的系数的数量；pCoeffs 是系数缓存的地址；pState 是状态缓存的地址。在静态初始化之前，要确保状态缓存中的值已经清零。

定点行为

使用定点 FIR 抽取器滤波函数需要注意。特别是要考虑，在每个函数内使用的累加器的溢出和饱和行为。具体参考每个函数各自的文档和使用说明。

4.5.3 函数说明

4.5.3.1 csky_vdsp2_fir_decimate_q15

```
void csky_vdsp2_fir_decimate_q15 (const csky_vdsp2_fir_decimate_instance_q15 *S, q15_t *pSrc, q15_
↪t *pDst, uint32_t blockSize)
```

参数:

*S：指向 FIR 抽取结构体实例

*pSrc：指向输入数据

*pDst：指向输出数据

blockSize：输入数据的数量

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

返回值:

无

缩放和溢出行为:

快速版本实现使用了 2.30 格式的 32 位累加器。累加器维持了中间乘法的所有精度，但是只有一个保护位。因此如果累加器的结果溢出，会扭曲最后结果。为了防止溢出，输入信号需要缩小 log2(numTaps) 位 (log2 是 2 为底的对数) 最后，2.30 格式的丢弃低 15 位截断为 2.15，然后饱和生成 1.15 格式的结果。

函数*csky_vdsp2_fir_decimate_fast_q15()* 是这个函数的一个快速版本，但是丢失了更多的精度。

4.5.3.2 csky_vdsp2_fir_decimate_q31

```
void csky_vdsp2_fir_decimate_q31 (const csky_vdsp2_fir_decimate_instance_q31 *S, q31_t *pSrc, q31_
↪t *pDst, uint32_t blockSize)
```

参数:

`*S`: 指向 FIR 抽取结构体实例

`*pSrc`: 指向输入数据

`*pDst`: 指向输出数据

`blockSize`: 输入数据的数量

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。输入数据表示为 1.31 格式。中间乘法生成 2.62 格式的结果，结果在 2.62 格式的 64 位累加器累加。因为只有一个保护位，所以需要缩小输入信号来防止溢出。因为最多会有 numTaps 个加法进位，所以需要缩小 log2(numTaps) 才可以保证没有溢出。最后，2.62 格式右移 31 位，然后饱和生成 1.31 格式的结果。

函数*csky_vdsp2_fir_decimate_fast_q31()* 是这个函数的一个快速版本，但是丢失了更多的精度。

4.5.3.3 csky_vdsp2_fir_decimate_fast_q15

```
void csky_vdsp2_fir_decimate_fast_q15 (const csky_vdsp2_fir_decimate_instance_q15 *S, q15_t *pSrc,␣
↪q15_t *pDst, uint32_t blockSize)
```

参数:

`*S`: 指向 FIR 抽取结构体实例

`*pSrc`: 指向输入数据

`*pDst`: 指向输出数据

`blockSize`: 输入数据的数量

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

返回值:

无

限制:

如果芯片不支持分对齐访问，则输入，输出，临时 buffer，都应该是 32 位对齐。

缩放和溢出行为:

快速版本实现使用了 2.30 格式的 32 位累加器。累加器维持了中间乘法的所有精度，但是只有一个保护位。因此如果累加器的结果溢出，会扭曲最后结果。为了防止溢出，输入信号需要缩小 log2(numTaps) 位 (log2 是 2 为底的对数) 最后，2.30 格式的丢弃低 15 位截断为 2.15，然后饱和生成 1.15 格式的结果。

函数*csky_vdsp2_fir_decimate_q15()* 是这个函数的一个慢速版本，使用了 64 位累加器防止溢出，保留了更多的精度。慢速和快速版本使用相同的结构体实例。使用函数*csky_vdsp2_fir_decimate_init_q15()* 可以初始化滤波器结构体。

4.5.3.4 csky_vdsp2_fir_decimate_fast_q31

```
void csky_vdsp2_fir_decimate_fast_q31 (csky_vdsp2_fir_decimate_instance_q31 *S, q31_t *pSrc, q31_t␣
↪*pDst, uint32_t blockSize)
```

参数:

`*S`: 指向 FIR 抽取结构体实例

`*pSrc`: 指向输入数据

`*pDst`: 指向输出数据

`blockSize`: 输入数据的数量

返回值:

无

缩放和溢出行为:

这个函数为了优化速度，舍弃了一些定点精度和溢出保护。每个 1.31 和 1.31 相乘的结果是 2.30 格式. 这些中间结果在一个 2.30 累加器相加。最后累加器饱和转换为 1.31 的结果。快速版本跟标准版本的有相同的溢出行为，由于丢弃了低 32 位相乘结果，维持了更少的精度。为了防止溢出，输入信号还需要缩小 log2(numTaps) 个位 (其中 log2 是 2 为底的对数).

函数*csky_vdsp2_fir_decimate_q31()* 是这个函数的一个慢速版本，使用了一个 64 位累加器，提供了更多的精度。慢速和快速版本使用相同的结构体实例。使用函数*csky_vdsp2_fir_decimate_init_q31()* 可以初始化滤波器结构体。

4.5.3.5 csky_vdsp2_fir_decimate_init_q15

```
csky_vdsp2_status csky_vdsp2_fir_decimate_init_q15 (csky_vdsp2_fir_decimate_instance_q15 *S,␣
↪uint16_t numTaps, uint8_t M, q15_t *pCoeffs, q15_t *pState, uint32_t blockSize)
```

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

参数:

*S：指向 FIR 抽取结构体实例

numTaps：滤波器系数的数量

M：抽取因子

*pCoeffs：指向滤波器系数

*pState：指向状态缓存

blockSize：输入样本的数量

返回值:

如果函数初始化成功，则返回 CSKY_MATH_SUCCESS ，如果 blockSize 不是 M 的整数倍，则返回 CSKY_MATH_LENGTH_ERROR.

简要说明:

pCoeffs 指向的滤波器系数数组，保存的顺序如下：

```
{b[numTaps-1], b[numTaps-2], b[N-2], ..., b[1], b[0]}
```

pState 指向状态变量数组，pState 的长度是 numTaps+blockSize-1 个字，其中 blockSize 是传递给*csky_vdsp2_fir_decimate_q15()* 的输入样本数量. M 是抽取因子.

4.5.3.6 csky_vdsp2_fir_decimate_init_q31

```
csky_vdsp2_status csky_vdsp2_fir_decimate_init_q31 (csky_vdsp2_fir_decimate_instance_q31 *S,␣
↪uint16_t numTaps, uint8_t M, q31_t *pCoeffs, q31_t *pState, uint32_t blockSize)
```

参数:

*S：指向 FIR 抽取结构体实例

numTaps：滤波器系数的数量

M：抽取因子

*pCoeffs：指向滤波器系数

*pState：指向状态缓存

blockSize：输入样本的数量

返回值:

如果函数初始化成功，则返回 CSKY_MATH_SUCCESS ，如果 blockSize 不是 M 的整数倍，则返回 CSKY_MATH_LENGTH_ERROR.

简要说明:

pCoeffs 指向的滤波器系数数组，保存的顺序如下：

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

```
{b[numTaps-1], b[numTaps-2], b[N-2], ..., b[1], b[0]}
```

pState 指向状态变量数组，pState 的长度是 numTaps+blockSize-1 个字，其中 blockSize 是传递给*csky_vdsp2_fir_decimate_q31()* 的输入样本数量. M 是抽取因子.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.6 有限冲激响应 (FIR) 格型滤波器

4.6.1 函数

- *csky_vdsp2_fir_lattice_q31*：Q31 FIR 格型滤波器处理函数
- *csky_vdsp2_fir_lattice_q15*：Q15 FIR 格型滤波器处理函数
- *csky_vdsp2_fir_lattice_init_q31*：Q31 FIR 格型滤波器初始化函数
- *csky_vdsp2_fir_lattice_init_q15*：Q15 FIR 格型滤波器初始化函数

4.6.2 简要说明

这些函数实现了 Q15，Q31 的有限冲激响应 (FIR) 格型滤波器. 格型滤波器使用在各种自适应滤波器应用。滤波器结构是前馈的，并且净脉冲响应是有限长度。函数以块为单位操作输入输出数据，每次调用滤波器函数处理 blockSize 个样本. pSrc 和 pDst 指向输入输出数组，数组包括 blockSize 个值。

算法:

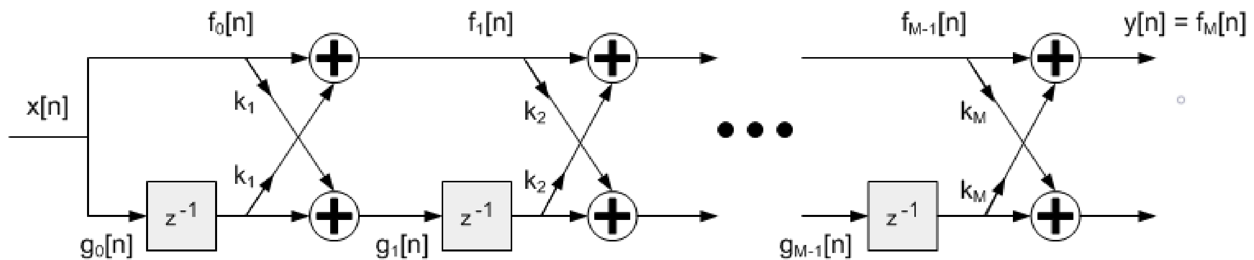

图 4.3: 有限冲激响应格型滤波器

实现了以下的差分方程:

```
f0[n] = g0[n] = x[n]
fm[n] = fm-1[n] + km * gm-1[n-1] for m = 1, 2, ...M
gm[n] = km * fm-1[n] + gm-1[n-1] for m = 1, 2, ...M
y[n] = fM[n]
```

pCoeffs 指向反射系数数组，数组大小是 numStages. 反射系数保存的顺序如下：

```
{k1, k2, ..., kM}
```

其中 M 是阶段的序号。

pState 指向状态数组，数组的大小是 numStages. 状态变量 (g 的值) 维持了之前的输入，按以下的顺序保存：

```
{g0[n], g1[n], g2[n] ...gM-1[n]}
```

状态变量在每个块数据处理后更新，系数不会被更新。

结构体实例

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

滤波器的系数和状态变量都保存在数据结构的实例中。每个滤波器都必须有一个单独的结构体实例。系数数组可能可以在几个实例之间共享，但是状态变量数组不能共享。为支持的 3 种数据类型分别提供了不同的结构体实例声明。

初始化函数

为每种支持的数据类型都提供了一个相应的初始化函数。初始化函数处理以下操作：

- 设置内部结构体字段的值
- 清零状态缓存中的值如果手动初始化，而不调用初始化函数，需要指定结构体实例的以下字段：numStages, pCoeffs, pState. pState 中的所有值置 0.

是否使用初始化函数是可选的。但是，使用了初始化函数，则不能将结构体实例放在常量数据段。要将结构体实例放在常量数据段，则必须手动初始化结构体实例。在静态初始化之前，要确保状态缓存中的值已经清零。下面的代码，为 3 种不同的滤波器，静态的初始化了结构体实例。

```
*csky_vdsp2_fir_lattice_instance_q31 S = {numStages, pState, pCoeffs};
*csky_vdsp2_fir_lattice_instance_q15 S = {numStages, pState, pCoeffs};
```

其中是 numStages 滤波器阶段的数量; pState 是状态缓存的地址; pCoeffs 是系数缓存的地址。

定点行为

使用定点 FIR 格型滤波器函数需要注意。特别是要考虑，在每个函数内使用的累加器的溢出和饱和行为。具体参考每个函数各自的文档和使用说明。

4.6.3 函数说明

4.6.3.1 csky_vdsp2_fir_lattice_q31

```
void csky_vdsp2_fir_lattice_q31 (const csky_vdsp2_fir_lattice_instance_q31 *S, q31_t *pSrc, q31_t␣
↪*pDst, uint32_t blockSize)
```

参数:

`*S`：指向 FIR 格型结构体实例

`*pSrc`：指向输入数据

`*pDst`：指向输出数据

`blockSize`：输入数据的数量

返回值:

无

缩放和溢出行为:

为了防止溢出，输入信号必须缩小 2*log2(numStages) 个位。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.6.3.2 csky_vdsp2_fir_lattice_q15

```
void csky_vdsp2_fir_lattice_q15 (const csky_vdsp2_fir_lattice_instance_q15 *S, q15_t *pSrc, q15_t␣
↪*pDst, uint32_t blockSize)
```

参数:

*S：指向 FIR 格型结构体实例

*pSrc：指向输入数据

*pDst：指向输出数据

blockSize：输入数据的数量

返回值:

无

4.6.3.3 csky_vdsp2_fir_lattice_init_q31

```
void csky_vdsp2_fir_lattice_init_q31 (csky_vdsp2_fir_lattice_instance_q31 *S, uint16_t numStages,␣
↪q31_t *pCoeffs, q31_t *pState)
```

参数:

*S：指向 FIR 格型结构体实例

numStages：滤波器阶段数量

*pCoeffs：指向系数缓存

*pState：指向状态缓存

返回值:

无

4.6.3.4 csky_vdsp2_fir_lattice_init_q15

```
void csky_vdsp2_fir_lattice_init_q15 (csky_vdsp2_fir_lattice_instance_q15 *S, uint16_t numStages,␣
↪q15_t *pCoeffs, q15_t *pState)
```

参数:

*S：指向 FIR 格型结构体实例

numStages：滤波器阶段数量

*pCoeffs：指向系数缓存

*pState：指向状态缓存

返回值:

无

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.7 有限冲激响应 (FIR) 稀疏滤波器

4.7.1 函数

- *csky_vdsp2_fir_sparse_q31* : Q31 稀疏 FIR 滤波器处理函数
- *csky_vdsp2_fir_sparse_q15* : Q15 稀疏 FIR 滤波器处理函数
- *csky_vdsp2_fir_sparse_q7* : Q7 稀疏 FIR 滤波器处理函数
- *csky_vdsp2_fir_sparse_init_q31* : Q31 稀疏 FIR 滤波器初始化函数
- *csky_vdsp2_fir_sparse_init_q15* : Q15 稀疏 FIR 滤波器初始化函数
- *csky_vdsp2_fir_sparse_init_q7* : Q7 稀疏 FIR 滤波器初始化函数

4.7.2 简要说明

这些函数实现了稀疏 FIR 滤波器。

稀疏 FIR 滤波器跟标准 FIR 滤波器相似，只不过它的大多数系数值是 0。

稀疏滤波器常用在通信和音频应用中模拟反射。

为 Q7，Q15，Q31 数据类型分别提供了不同的函数。

函数以块为单位处理输入输出数据，并且滤波器每次调用处理 blockSize 个样本. pSrc 和 pDst 指向输入输出数组，数组分别包含 blockSize 个值。

算法:

稀疏滤波器结构体实例在系数数组 b 之外，还有一个索引 pTapDelay 来指定非零系数的位置。算法相比标准 FIR 效率更高的原因是，在算法实现里面利用了非零系数索引，省略了大多数与 0 相乘的步骤。

```
y[n] = b[0] * x[n-pTapDelay[0]] + b[1] * x[n-pTapDelay[1]] + b[2] * x[n-pTapDelay[2]] +
↪...+ b[numTaps-1] * x[n-pTapDelay[numTaps-1]]
```

pCoeffs 指向系数数组, 数组的大小是 numTaps; pTapDelay 指向非零索引数组, 数组的大小也是 numTaps; pState 指向状态数组，数组的大小 maxDelay + blockSize, 其中 maxDelay 是 pTapDelay 数组中最大的可用索引值。有些处理函数还要求提供一些临时的缓存。

结构体实例

滤波器的系数和状态变量都保存在数据结构的实例中。每个滤波器都必须有一个单独的结构体实例。系数数组可能可以在几个实例之间共享，但是状态变量数组不能共享。为支持的 4 种数据类型分别提供了不同的结构体实例声明。

初始化函数

为每种支持的数据类型都提供了一个相应的初始化函数。初始化函数处理以下操作：

- 设置内部结构体字段的值
- 清零状态缓存中的值如果手动初始化，而不调用初始化函数，需要指定结构体实例的以下字段：numTaps, pCoeffs, pTapDelay, maxDelay, stateIndex, pState. pState 中的所有值置 0.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

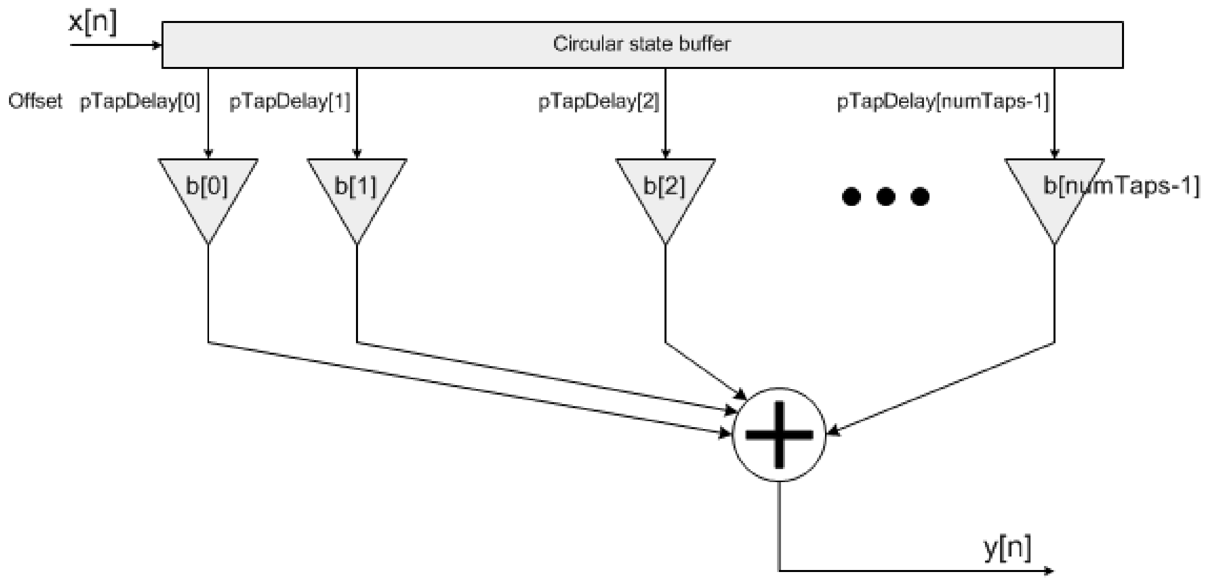

图 4.4: 稀疏 FIR 滤波器. b[n] 表示滤波器系数

是否使用初始化函数是可选的。但是，使用了初始化函数，则不能将结构体实例放在常量数据段。要将结构体实例放在常量数据段，则必须手动初始化结构体实例。在静态初始化之前，要确保状态缓存中的值已经清零。下面的代码，为 4 种不同的滤波器，静态的初始化了结构体实例。

```
*csky_vdsp2_fir_sparse_instance_q31 S = {numTaps, 0, pState, pCoeffs, maxDelay,␣
↪pTapDelay};
*csky_vdsp2_fir_sparse_instance_q15 S = {numTaps, 0, pState, pCoeffs, maxDelay,␣
↪pTapDelay};
*csky_vdsp2_fir_sparse_instance_q7 S =  {numTaps, 0, pState, pCoeffs, maxDelay,␣
↪pTapDelay};
```

定点行为

使用定点稀疏 FIR 滤波器函数需要注意。特别是要考虑，在每个函数内使用的累加器的溢出和饱和行为。具体参考每个函数各自的文档和使用说明。

4.7.3 函数说明

4.7.3.1 csky__vdsp2__fir__sparse__q31

```
void csky_vdsp2_fir_sparse_q31 (csky_vdsp2_fir_sparse_instance_q31 *S, q31_t *pSrc, q31_t *pDst,␣
↪q31_t *pScratchIn, uint32_t blockSize)
```

参数:

*S: 指向稀疏 FIR 结构体实例

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

*pSrc：指向输入数据

*pDst：指向输出数据

*pScratchIn：指向临时缓存，大小是 blockSize

blockSize：指向处理的输入样本数量

返回值：

无

缩放和溢出行为：

函数实现使用了一个内部 32 位累加器。1.31 和 1.31 相乘的结果截断为 2.30 格式. 中间乘法的过程丢失了部分精度，并且只有一个保护位如果累加器溢出，不会做饱和处理，而是往符号位方向溢出。为了防止溢出，输入信号或者系数必须缩小 log2(numTaps) 位.

4.7.3.2 csky_vdsp2_fir_sparse_q15

```
void csky_vdsp2_fir_sparse_q15 (csky_vdsp2_fir_sparse_instance_q15 *S, q15_t *pSrc, q15_t *pDst,␣
↪q15_t *pScratchIn, q31_t *pScratchOut, uint32_t blockSize)
```

参数：

*S：指向稀疏 FIR 结构体实例

*pSrc：指向输入数据

*pDst：指向输出数据

*pScratchIn：指向临时缓存，大小是 blockSize

*pScratchOut：指向临时缓存，大小是 blockSize

blockSize：指向处理的输入样本数量

返回值：

无

缩放和溢出行为：

函数实现使用了一个内部 32 位累加器。1.15 和 1.15 相乘生成 2.30 的结果，结果在 2.30 格式的累加器累加。因此相乘结果的所有精度都可以保留，但是累加器只有一个保护位。如果累加器溢出，不会做饱和处理，而是往符号位方向溢出。处理完所有的乘累加后，2.30 累加器截断成 2.15 格式，然后饱和成 1.15 格式。为了防止溢出，输入信号或者系数必须缩小 log2(numTaps) 位.

4.7.3.3 csky_vdsp2_fir_sparse_q7

```
void csky_vdsp2_fir_sparse_q7 (csky_vdsp2_fir_sparse_instance_q7 *S, q7_t *pSrc, q7_t *pDst, q7_t␣
↪*pScratchIn, q31_t *pScratchOut, uint32_t blockSize)
```

参数：

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

*S：指向稀疏 FIR 结构体实例

*pSrc：指向输入数据

*pDst：指向输出数据

*pScratchIn：指向临时缓存，大小是 blockSize

*pScratchOut：指向临时缓存，大小是 blockSize

blockSize：指向处理的输入样本数量

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 32 位累加器。系数和状态变量都用 1.7 格式表示，相乘生成 2.14 结果。2.14 的中间结果在 18.14 格式的 32 位累加器累加。这些操作可以保留所有的精度，并且不会有溢出风险累加器之后丢弃低 7 位截断为 18.7 格式。最后，结果转换为 1.7 格式。

4.7.3.4 csky_vdsp2_fir_sparse_init_q31

```
void csky_vdsp2_fir_sparse_init_q31 (csky_vdsp2_fir_sparse_instance_q31 *S, uint16_t numTaps, q31_
↪t *pCoeffs, q31_t *pState, int32_t *pTapDelay, uint16_t maxDelay, uint32_t blockSize)
```

参数:

*S：指向稀疏 FIR 结构体实例

numTaps：滤波器中非零系数数量

*pCoeffs：指向滤波器系数数组

*pState：指向状态数组

*pTapDelay：指向偏移数组

maxDelay：支持的最大偏移

blockSize：处理的样本数量

返回值:

无

简要说明:

pCoeffs 保存了滤波器系数，长度为 numTaps. pState 保存了滤波器的状态变量，长度为 maxDelay + blockSize，其中 maxDelay 是偏移支持的最大值。blockSize 是处理样本的数量，传入给*csky_vdsp2_fir_sparse_q31()* 。

4.7.3.5 csky_vdsp2_fir_sparse_init_q15

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

```
void csky_vdsp2_fir_sparse_init_q15 (csky_vdsp2_fir_sparse_instance_q15 *S, uint16_t numTaps, q15_
↪t *pCoeffs, q15_t *pState, int32_t *pTapDelay, uint16_t maxDelay, uint32_t blockSize)
```

参数:

*S: 指向稀疏 FIR 结构体实例

numTaps: 滤波器中非零系数数量

*pCoeffs: 指向滤波器系数数组

*pState: 指向状态数组

*pTapDelay: 指向偏移数组

maxDelay: 支持的最大偏移

blockSize: 处理的样本数量

返回值:

无

简要说明:

pCoeffs 保存了滤波器系数，长度为 numTaps. pState 保存了滤波器的状态变量，长度为 maxDelay + blockSize，其中 maxDelay 是偏移支持的最大值。blockSize 是处理样本的数量，传入给*csky_vdsp2_fir_sparse_q15()*。

4.7.3.6 csky_vdsp2_fir_sparse_init_q7

```
void csky_vdsp2_fir_sparse_init_q7 (csky_vdsp2_fir_sparse_instance_q7 *S, uint16_t numTaps, q7_t␣
↪*pCoeffs, q7_t *pState, int32_t *pTapDelay, uint16_t maxDelay, uint32_t blockSize)
```

参数:

*S: 指向稀疏 FIR 结构体实例

numTaps: 滤波器中非零系数数量

*pCoeffs: 指向滤波器系数数组

*pState: 指向状态数组

*pTapDelay: 指向偏移数组

maxDelay: 支持的最大偏移

blockSize: 处理的样本数量

返回值:

无

简要说明:

pCoeffs 保存了滤波器系数，长度为 numTaps. pState 保存了滤波器的状态变量，长度为 maxDelay + blockSize，其中 maxDelay 是偏移支持的最大值。blockSize 是处理样本的数量，传入给*csky_vdsp2_fir_sparse_q7()*。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.8 有限冲激响应 (FIR) 插值滤波器

4.8.1 函数

- *csky_vdsp2_fir_interpolate_q31*：Q31 FIR 插值处理函数
- *csky_vdsp2_fir_interpolate_q15*：Q15 FIR 插值处理函数
- *csky_vdsp2_fir_interpolate_init_q31*：Q31 FIR 插值初始化函数
- *csky_vdsp2_fir_interpolate_init_q15*：Q15 FIR 插值初始化函数

4.8.2 简要说明

函数组合了一个过采样器 (零填充) 和一个 FIR 滤波器。

这些函数用于多速率系统中，在不引入高频图像的情况下增加信号的采样率。

从概念上讲，这些功能等同于下面的框图：

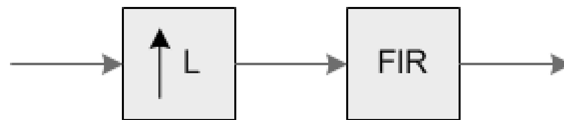

图 4.5: FIR 插值器中的组件

通过因子 L 过采样后, 信号需要用一个截止频率 1/L 的低通滤波器滤波，以便消除频谱的高频拷贝。函数的调用者负责提供滤波器的系数。

CSI DSP 库中的 FIR 插值函数提供了过采样器和 FIR 滤波器的一种组合方式。过采样器在样本中插入了 L-1 个 0. 滤波器设计成不会与这些插值出来的 0 相乘，而是直接跳过，这样可以提高效率，减少计算量。函数以块为单位操作输入输出数据。pSrc 指向一个大小为 blockSize 的输入数组，pDst 指向一个大小为 blockSize*L 的输出数组.

DSP 库为 Q15，Q31 数据类型分别提供了不同的函数。

算法:

函数使用一个多相滤波器结构:

```
y[n] = b[0] * x[n] + b[L]   * x[n-1] + ... + b[L*(phaseLength-1)] * x[n-phaseLength+1]
y[n+1] = b[1] * x[n] + b[L+1] * x[n-1] + ... + b[L*(phaseLength-1)+1] * x[n-
↪phaseLength+1]
...
y[n+(L-1)] = b[L-1] * x[n] + b[2*L-1] * x[n-1] + ....+ b[L*(phaseLength-1)+(L-1)] * x[n-
↪phaseLength+1]
```

这种方式比直接过采样，然后滤波的算法更效率。这种方式的计算量减少到标准 FIR 滤波器的 1/L.

pCoeffs 指向一个系数数组，数组大小为 numTaps. numTaps 必须是插值因子的 L 的倍数，并且会在初始化函数里检查。函数内部会将 FIR 滤波器的冲激响应除成更短的长度 phaseLength=numTaps/L. 系数的保存顺序如下：

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

```
{b[numTaps-1], b[numTaps-2], b[N-2], ..., b[1], b[0]}
```

pState 指向状态数组，数组的大小为 blockSize + phaseLength - 1. 状态缓存中的保存顺序如下:

```
{x[n-phaseLength+1], x[n-phaseLength], x[n-phaseLength-1], x[n-phaseLength-2]....x[0],␣
↪x[1], ..., x[blockSize-1]}
```

状态变量在数据块处理的时候会更新，系数不会更新。

结构体实例

滤波器的系数和状态变量都保存在数据结构的实例中。每个滤波器都必须要有一个结构体实例。系数数组可能可以在几个实例之间共享，但是状态变量数组必须单独分配。为 Q31 和 Q15 数据类型分别提供了不同的结构体实例声明。

初始化函数

每种数据类型都有一个相应的初始化函数。初始化函数处理以下操作：

- 设置内部结构体字段的值
- 清零状态缓存
- 确保滤波器的长度是插值因子的整数倍如果手动初始化，而不调用初始化函数，需要指定结构体实例的以下字段：L (插值因子), pCoeffs, phaseLength (numTaps / L), pState. pState 中的所有值置 0

是否使用初始化函数是可选的。但是，使用了初始化函数，则不能将结构体实例放在常量数据段。要将结构体实例放在常量数据段，则必须手动初始化结构体实例。下面的代码，为 3 种不同的滤波器，静态的初始化了结构体实例。

```
csky_vdsp2_fir_interpolate_instance_q31 S = {L, phaseLength, pCoeffs, pState};
csky_vdsp2_fir_interpolate_instance_q15 S = {L, phaseLength, pCoeffs, pState};
```

其中 L 是插值因子; phaseLength=numTaps/L 是 FIR 滤波器内部使用的更短的长度，pCoeffs 是系数缓存的地址; pState 是状态缓存的地址. 在静态初始化之前，要确保状态缓存中的值已经清零。

定点行为

使用定点 FIR 插值器滤波函数需要注意。特别是要考虑，在每个函数内使用的累加器的溢出和饱和行为。具体参考每个函数各自的文档和使用说明。

4.8.3 函数说明

4.8.3.1 csky_vdsp2_fir_interpolate_q31

```
void csky_vdsp2_fir_interpolate_q31 (const csky_vdsp2_fir_interpolate_instance_q31 *S, q31_t *pSrc,
↪ q31_t *pDst, uint32_t blockSize)
```

参数:

`*S`：指向 FIR 插值结构体实例

`*pSrc`：指向输入数据

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

*pDst：指向输出数据

blockSize：输入数据的数量

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。输入数据表示为 1.31 格式。中间乘法生成 2.62 格式的结果，结果在 2.62 格式的 64 位累加器累加。因为只有一个保护位，所以需要缩小输入信号来防止溢出。因为最多会有 numTaps/L 个加法进位，所以需要缩小 1/(numTaps/L) 才可以保证没有溢出。最后，2.62 格式右移 31 位，然后饱和生成 1.15 格式的结果。

4.8.3.2 csky_vdsp2_fir_interpolate_q15

```
void csky_vdsp2_fir_interpolate_q15 (const csky_vdsp2_fir_interpolate_instance_q15 *S, q15_t *pSrc,
↪ q15_t *pDst, uint32_t blockSize)
```

参数:

*S：指向 FIR 插值结构体实例

*pSrc：指向输入数据

*pDst：指向输出数据

blockSize：输入数据的数量

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。输入数据表示为 1.15 格式。中间乘法生成 2.30 格式的结果，结果在 34.30 格式的 64 位累加器累加。因为有 33 位保护位，所以不会有溢出的风险。同时还可以保存所有的中间乘法结果的精度。最后，34.30 格式的丢弃低 15 位截断为 34.15，然后饱和生成 1.15 格式的结果。

4.8.3.3 csky_vdsp2_fir_interpolate_init_q31

```
void csky_vdsp2_fir_interpolate_init_q31 (csky_vdsp2_fir_interpolate_instance_q31 *S, uint8_t L,␣
↪uint16_t numTaps, q31_t *pCoeffs, q31_t *pState, uint32_t blockSize)
```

参数:

*S：指向 FIR 插值结构体实例

L：过采样因子

numTaps：滤波器中系数的数量

*pCoeffs：指向系数缓存

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

`*pState`：指向状态缓存

`blockSize`：处理的输入样本数量

返回值:

无

简要说明:

pCoeffs 指向滤波器系数数组，保存的顺序如下：

```
{b[numTaps-1], b[numTaps-2], b[numTaps-2], ..., b[1], b[0]}
```

滤波器的长度 numTaps 必须是插值因子 L 的整数倍.

pState 指向状态变量数组。pState 的长度是 (numTaps/L)+blockSize-1 。其中 blockSize 是处理的输入样本数量，传给 *csky_vdsp2_fir_interpolate_q31()* 。

4.8.3.4 csky_vdsp2_fir_interpolate_init_q15

```
void csky_vdsp2_fir_interpolate_init_q15 (csky_vdsp2_fir_interpolate_instance_q15 *S, uint8_t L,␣
↪uint16_t numTaps, q15_t *pCoeffs, q15_t *pState, uint32_t blockSize)
```

参数:

`*S`：指向 FIR 插值结构体实例

`L`：过采样因子

`numTaps`：滤波器中系数的数量

`*pCoeffs`：指向系数缓存

`*pState`：指向状态缓存

`blockSize`：处理的输入样本数量

返回值:

无

简要说明:

pCoeffs 指向滤波器系数数组，保存的顺序如下：

```
{b[numTaps-1], b[numTaps-2], b[numTaps-2], ..., b[1], b[0]}
```

滤波器的长度 numTaps 必须是插值因子 L 的整数倍.

pState 指向状态变量数组。pState 的长度是 (numTaps/L)+blockSize-1 。其中 blockSize 是处理的输入样本数量，传给 *csky_vdsp2_fir_interpolate_q15()* 。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.9 无限冲激响应 (IIR) 格型滤波器

4.9.1 函数

- *csky_vdsp2_iir_lattice_q31*：Q31 IIR 格型滤波器处理函数
- *csky_vdsp2_iir_lattice_q15*：Q15 IIR 格型滤波器处理函数
- *csky_vdsp2_iir_lattice_init_q31*：Q31 IIR 格型滤波器初始化函数
- *csky_vdsp2_iir_lattice_init_q15*：Q15 IIR 格型滤波器初始化函数

4.9.2 简要说明

这些函数实现了 Q15，Q31 的无限冲激响应 (IIR) 格型滤波器. 格型滤波器使用在各种自适应滤波器应用。滤波器结构具有前馈和反馈分量，并且净脉冲响应是无限长度。函数以块为单位操作输入输出数据，每次调用滤波器函数处理 blockSize 个样本. pSrc 和 pDst 指向输入输出数组，数组包括 blockSize 个值。

算法:

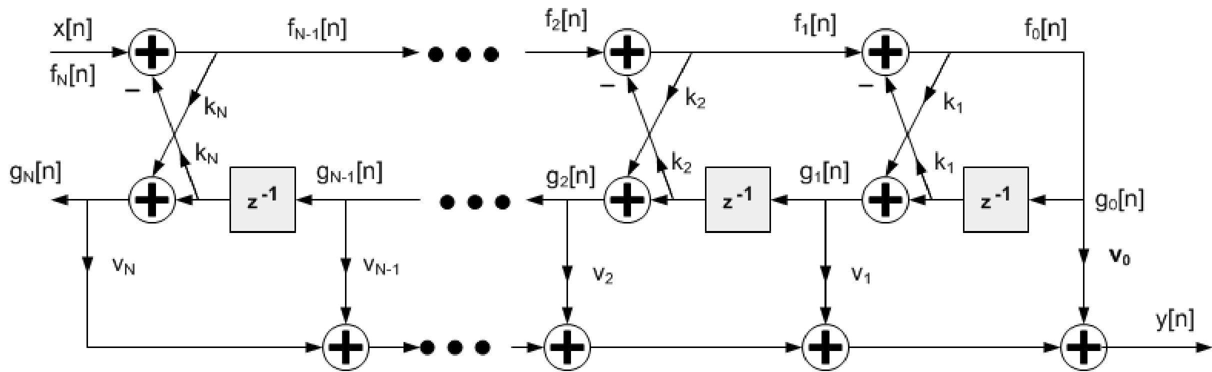

图 4.6: 无限冲激响应格型滤波器

```
fN(n)   =  x(n)
fm-1(n) = fm(n) - km * gm-1(n-1)   for m = N, N-1, ...1
gm(n)   = km * fm-1(n) + gm-1(n-1) for m = N, N-1, ...1
y(n)    = vN * gN(n) + vN-1 * gN-1(n) + ...+ v0 * g0(n)
```

pkCoeffs 指向反射系数数组，数组大小是 numStages. 反射系数保存的顺序如下：

```
{kN, kN-1, ....k1}
```

pvCoeffs 指向梯形系数数组，数组大小是 (numStages+1). 梯形系数数组保存的顺序如下：

```
{vN, vN-1, ...v0}
```

pState 指向状态数组，数组大小是 numStages + blockSize. 上面提到的状态变量（g 值）保存在 pState 数组. 状态变量在每个块数据处理后更新，系数不会被更新。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

结构体实例

滤波器的系数和状态变量都保存在数据结构的实例中。每个滤波器都必须有一个单独的结构体实例。系数数组可能可以在几个实例之间共享，但是状态变量数组不能共享。为支持的 3 种数据类型分别提供了不同的结构体实例声明。

初始化函数

为每种支持的数据类型都提供了一个相应的初始化函数。初始化函数处理以下操作：

- 设置内部结构体字段的值
- 清零状态缓存中的值如果手动初始化，而不调用初始化函数，需要指定结构体实例的以下字段：numStages, pkCoeffs, pvCoeffs, pState. pState 中的所有值置 0.

是否使用初始化函数是可选的。但是，使用了初始化函数，则不能将结构体实例放在常量数据段。要将结构体实例放在常量数据段，则必须手动初始化结构体实例。在静态初始化之前，要确保状态缓存中的值已经清零。下面的代码，为 3 种不同的滤波器，静态的初始化了结构体实例。

```
*csky_vdsp2_iir_lattice_instance_q31 S = {numStages, pState, pkCoeffs, pvCoeffs};
*csky_vdsp2_iir_lattice_instance_q15 S = {numStages, pState, pkCoeffs, pvCoeffs};
```

其中 numStages 是滤波器阶段的数量; pState 指向状态缓存数组; pkCoeffs 指向反射系数数组; pvCoeffs 指向梯形系数数组.

定点行为

使用定点 IIR 格型滤波器函数需要注意。特别是要考虑，在每个函数内使用的累加器的溢出和饱和行为。具体参考每个函数各自的文档和使用说明。

4.9.3 函数说明

4.9.3.1 csky_vdsp2_iir_lattice_q31

```
void csky_vdsp2_iir_lattice_q31 (const csky_vdsp2_iir_lattice_instance_q31 *S, q31_t *pSrc, q31_t␣
↪*pDst, uint32_t blockSize)
```

参数:

`*S`：指向 IIR 格型结构体实例

`*pSrc`：指向输入数据

`*pDst`：指向输出数据

`blockSize`：输入数据的数量

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。输入数据表示为 1.31 格式。中间乘法生成 2.62 格式的结果，结果在 2.62 格式的 64 位累加器累加。因为只有一个保护位，所以需要缩小输入信号来防止溢出。为了防止溢出，必须缩小 2*log2(numStages) 个位。最后，2.62 格式右移 31 位，然后饱和生成 1.15 格式的结果。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.9.3.2 csky_vdsp2_iir_lattice_q15

```
void csky_vdsp2_iir_lattice_q15 (const csky_vdsp2_iir_lattice_instance_q15 *S, q15_t *pSrc, q15_t
↪*pDst, uint32_t blockSize)
```

参数:

`*S`: 指向 IIR 格型结构体实例

`*pSrc`: 指向输入数据

`*pDst`: 指向输出数据

`blockSize`: 输入数据的数量

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。系数和状态变量都表示为 1.15 格式。中间乘法生成 2.30 格式的结果，结果在 34.30 格式的 64 位累加器累加。因为有 33 位保护位，所以不会有溢出的风险。同时还可以保存所有的中间乘法结果的精度。最后，34.30 格式的丢弃低 15 位截断为 34.15，然后饱和生成 1.15 格式的结果。

4.9.3.3 csky_vdsp2_iir_lattice_init_q31

```
void csky_vdsp2_iir_lattice_init_q31 (csky_vdsp2_iir_lattice_instance_q31 *S, uint16_t numStages,
↪q31_t *pkCoeffs, q31_t *pvCoeffs, q31_t *pState, uint32_t blockSize)
```

参数:

`*S`: 指向 IIR 格型结构体实例

`numStages`: 滤波器阶段的数量

`*pkCoeffs`: 指向反射系数缓存，数组的长度是 numStages

`*pvCoeffs`: 指向梯形系数缓存，数组的长度是 numStages+1

`*pState`: 指向状态缓存，数组的长度是 numStages+blockSize

`blockSize`: 处理的样本数量

返回值:

无

4.9.3.4 csky_vdsp2_iir_lattice_init_q15

```
void csky_vdsp2_iir_lattice_init_q15 (csky_vdsp2_iir_lattice_instance_q15 *S, uint16_t numStages,
↪q15_t *pkCoeffs, q15_t *pvCoeffs, q15_t *pState, uint32_t blockSize)
```

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

参数:

`*S`: 指向 IIR 格型结构体实例

`numStages`: 滤波器阶段的数量

`*pkCoeffs`: 指向反射系数缓存，数组的长度是 numStages

`*pvCoeffs`: 指向梯形系数缓存，数组的长度是 numStages+1

`*pState`: 指向状态缓存，数组的长度是 numStages+blockSize

`blockSize`: 处理的样本数量

返回值:

无

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.10 最小均方 (LMS) 滤波器

4.10.1 函数

- *csky_vdsp2_lms_q31* : Q31 LMS 滤波器处理函数
- *csky_vdsp2_lms_q15* : Q15 LMS 滤波器处理函数
- *csky_vdsp2_lms_init_q31* : Q31 LMS 滤波器初始化函数
- *csky_vdsp2_lms_init_q15* : Q15 LMS 滤波器初始化函数

4.10.2 简要说明

LMS 滤波器是一类自适应滤波器，使用的是一种梯度下降的算法，根据瞬时误差更新滤波器的系数，达到“学习”一种未知的转换方式目的。

自适应滤波器常应用在通信系统，均衡器，和噪声去除。

CSI DSP 库内的 LMS 滤波器函数支持 Q15, Q31 数据类型。

库内也有归一化 LMS 滤波器，归一化 LMS 滤波器系数自适应与输入信号的电平无关。

一个 LMS 滤波器包括以下两个部分:

- 第一部分是一个标准横向 FIR 滤波器。
- 第二部分是系数更新机制。

LMS 滤波器有两个输入信号。一个是接受的输入信号，另一个是参考的输入信号，输出两个信号，一个是 FIR 滤波器的输出信号, 另一个是与参考输入相比的误差信号。滤波器根据输出和参考输入之间的差值更新系数，直到 FIR 滤波器的 FIR 滤波器的输出跟参考输入相符。误差通过滤波器的调解倾向于 0。

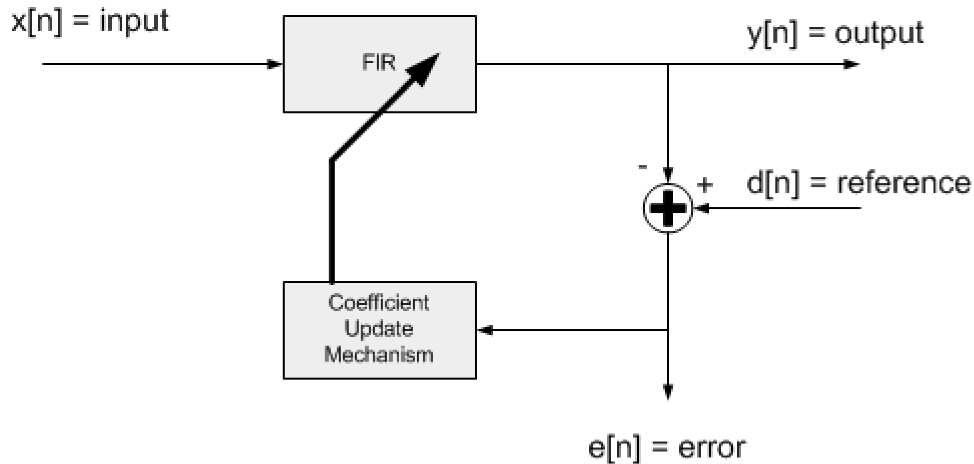

图 4.7: 最小均方滤波器结构

函数以块为单位处理数据，每次调用滤波器函数处理 blockSize 个样本。pSrc 指向输入信号, pRef 指向参考信号，pOut 指向输出信号和 pErr 指向误差信号。所有的数组都包括 blockSize 个值。

函数以块为单位操作。滤波器内部系数 b[n] 以样本为单位更新。

算法:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

输出信号 y[n] 通过标准 FIR 滤波器计算:

```
y[n] = b[0] * x[n] + b[1] * x[n-1] + b[2] * x[n-2] + ...+ b[numTaps-1] * x[n-numTaps+1]
```

误差信号等于参考信号 d[n] 和滤波器输出的差值:

```
e[n] = d[n] - y[n].
```

计算每个误差信号的每个样本后，滤波器系数 b[k] 以样本为单位更新:

```
b[k] = b[k] + e[n] * mu * x[n-k],  for k=0, 1, ..., numTaps-1
```

其中 mu 是步进大小，控制系数收敛速率。
接口中, pCoeffs 指向系数数组，大小是 numTaps. 系数保存的顺序如下:

```
{b[numTaps-1], b[numTaps-2], b[N-2], ..., b[1], b[0]}
```

pState 指向状态数组，数组大小是 numTaps + blockSize - 1. 样本在状态缓存中的保存顺序是:

```
{x[n-numTaps+1], x[n-numTaps], x[n-numTaps-1], x[n-numTaps-2]....x[0], x[1], ...,
↪x[blockSize-1]}
```

注意: 状态缓存的长度超过了系数数组 blockSize-1 个样本. 增长的状态缓存长度可以用来取代传统 FIR 滤波器使用的循环寻址，从而显著提高速度。状态变量在每块数据处理后更新。

结构体实例

滤波器的系数和状态变量都保存在数据结构的实例中。每个滤波器都必须有一个单独的结构体实例。系数数组可能可以在几个实例之间共享，但是状态变量数组不能共享。为支持的 3 种数据类型分别提供了不同的结构体实例声明。

初始化函数

为每种支持的数据类型都提供了一个相应的初始化函数。初始化函数处理以下操作:

- 设置内部结构体字段的值
- 清零状态缓存中的值如果手动初始化，而不调用初始化函数，需要指定结构体实例的以下字段: numTaps, pCoeffs, mu, postShift , pState. pState 中的所有值置 0

是否使用初始化函数是可选的。但是，使用了初始化函数，则不能将结构体实例放在常量数据段。要将结构体实例放在常量数据段，则必须手动初始化结构体实例。在静态初始化之前，要确保状态缓存中的值已经清零。下面的代码，为 3 种不同的滤波器，静态的初始化了结构体实例。

```
csky_vdsp2_lms_instance_q31 S = {numTaps, pState, pCoeffs, mu, postShift};
csky_vdsp2_lms_instance_q15 S = {numTaps, pState, pCoeffs, mu, postShift};
```

其中 numTaps 是滤波器系数的数量; pState 是状态缓存的地址; pCoeffs 是系数缓存的地址; mu 是步进大小; postShift 是系数的移位数.

定点行为:

使用 Q15 和 Q31 版本的 LMS 滤波器函数需要注意。下列问题必须考虑:

- 系数缩放

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

- 溢出和饱和

系数缩放:

滤波器系数表示为一个小数值,被限制在范围 [-1 +1) 之间。定点函数有一个附加的缩放参数 postShift。滤波器的输出累加器是一个可移位的寄存器，结果移动 postShift 位。基本上就是将滤波器系数缩放 2^postShift ，允许将滤波器的系数扩展到超过范围 [+1 -1)。postShift 的值根据用户系统模型期望的增益设定。

溢出和饱和:

Q15 和 Q31 版本的溢出和饱和分别描述在各个函数各自的文档部分。

4.10.3 函数说明

4.10.3.1 csky_vdsp2_lms_q31

```
void csky_vdsp2_lms_q31 (const csky_vdsp2_lms_instance_q31 *S, q31_t *pSrc, q31_t *pRef, q31_t
↪*pOut, q31_t *pErr, uint32_t blockSize)
```

参数:

*S：指向 LMS 滤波器结构体实例

*pSrc：指向输入数据

*pRef：指向参考数据

*pOut：指向输出数据

*pErr：指向误差数据

blockSize：处理的样本的数量

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。累加器是 2.62 格式，并且维持了中间乘法结果的所有精度，但是只有一个保护位。为了防止溢出，输入信号必须缩小 log2(numTaps) 位. 参考信号不应该缩小。在所有的乘累加处理后，2.62 格式累加器右移，然后饱和生成 1.31 最后的结果输出信号和错误信号是 1.31 格式.

这个滤波器中，滤波器系数会根据样本更新，并且系数更新是饱和的。

4.10.3.2 csky_vdsp2_lms_q15

```
void csky_vdsp2_lms_q15 (const csky_vdsp2_lms_instance_q15 *S, q15_t *pSrc, q15_t *pRef, q15_t
↪*pOut, q15_t *pErr, uint32_t blockSize)
```

参数:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

*S：指向 LMS 滤波器结构体实例

*pSrc：指向输入数据

*pRef：指向参考数据

*pOut：指向输出数据

*pErr：指向误差数据

blockSize：处理的样本的数量

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。系数和状态变量都表示为 1.15 格式。中间乘法生成 2.30 格式的结果，结果在 34.30 格式的 64 位累加器累加。因为有 33 位保护位，所以不会有溢出的风险。同时还可以保存所有的中间乘法结果的精度。最后，34.30 格式的丢弃低 15 位截断为 34.15，然后饱和生成 1.15 格式的结果。

这个滤波器中，滤波器系数会根据样本更新，并且系数更新是饱和的。

4.10.3.3 csky_vdsp2_lms_init_q31

```
void csky_vdsp2_lms_init_q31 (csky_vdsp2_lms_instance_q31 *S, uint16_t numTaps, q31_t *pCoeffs,␣
↪q31_t *pState, q31_t mu, uint32_t blockSize, uint32_t postShift)
```

参数:

*S：指向 LMS 滤波器结构体实例

numTaps：滤波器系数的数量

*pCoeffs：指向系数缓存

*pState：指向状态缓存

mu：控制滤波器系数更新的步进大小

blockSize：处理的样本的数量

postShift：系数的移位数

返回值:

无

简要说明:

pCoeffs 指向滤波器系数的数组，保存的顺序如下：

```
{b[numTaps-1], b[numTaps-2], b[N-2], ..., b[1], b[0]}
```

初始化滤波器系数作为自适应滤波器的起始点 pState 指向一个长度为 numTaps+blockSize-1 样本数组, 其中 blockSize 是处理的输入样本数量，传入函数*csky_vdsp2_lms_q31()* 。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.10.3.4 csky_vdsp2_lms_init_q15

```
void csky_vdsp2_lms_init_q15 (csky_vdsp2_lms_instance_q15 *S, uint16_t numTaps, q15_t *pCoeffs,
↪q15_t *pState, q15_t mu, uint32_t blockSize, uint32_t postShift)
```

参数:

`*S`：指向 LMS 滤波器结构体实例

`numTaps`：滤波器系数的数量

`*pCoeffs`：指向系数缓存

`*pState`：指向状态缓存

`mu`：控制滤波器系数更新的步进大小

`blockSize`：处理的样本的数量

`postShift`：系数的移位数

返回值:

无

简要说明:

pCoeffs 指向滤波器系数的数组，保存的顺序如下：

```
{b[numTaps-1], b[numTaps-2], b[N-2], ..., b[1], b[0]}
```

初始化滤波器系数作为自适应滤波器的起始点 pState 指向一个长度为 numTaps+blockSize-1 样本数组, 其中 blockSize 是处理的输入样本数量，传入函数*csky_vdsp2_lms_q15()* 。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

4.11 归一化 LMS 滤波器

4.11.1 函数

- *csky_vdsp2_lms_norm_q31*：Q31 归一化 LMS 滤波器处理函数
- *csky_vdsp2_lms_norm_q15*：Q15 归一化 LMS 滤波器处理函数
- *csky_vdsp2_lms_norm_init_q31*：Q31 归一化 LMS 滤波器初始化函数
- *csky_vdsp2_lms_norm_init_q15*：Q15 归一化 LMS 滤波器初始化函数

4.11.2 简要说明

这组函数实现了常用的自适应滤波器。归一化 LMS 在最小均方 (LMS) 自适应滤波器的基础上，附加了额外的归一化因子，提高了滤波器的自适应速率。CSI DSP 库内的归一化 LMS 滤波器函数支持 Q15, Q31 数据类型。

一个归一化最小均方 (NLMS) 滤波器包括以下两部分。

- 第一部分是一个标准横向 FIR 滤波器。
- 第二部分是系数更新机制。

归一化 LMS 滤波器有两个输入信号。一个是接受的输入信号，另一个是参考的输入信号，输出两个信号，一个是 FIR 滤波器的输出信号，另一个是与参考输入相比的误差信号。滤波器根据输出和参考输入之间的差值更新系数，直到 FIR 滤波器的 FIR 滤波器的输出跟参考输入相符。误差通过滤波器的调解倾向于 0。

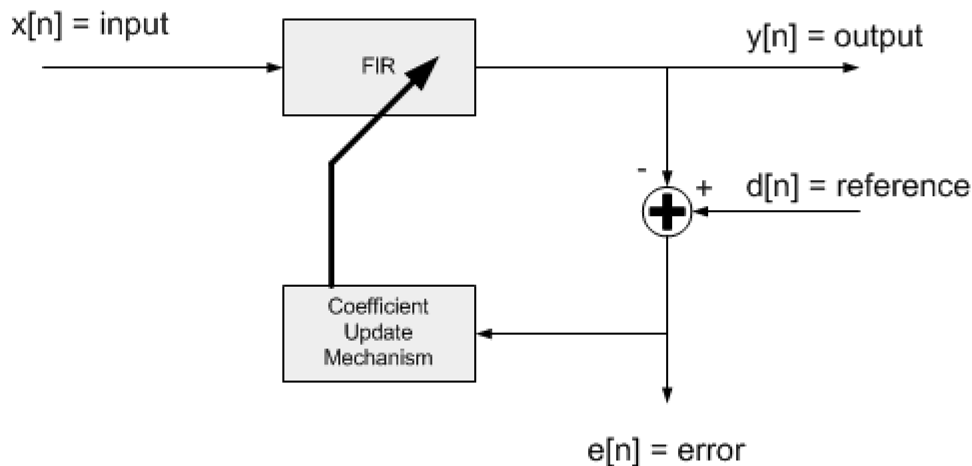

图 4.8: 归一化 LMS 滤波器中间结构

函数以块为单位处理数据，每次调用滤波器函数处理 blockSize 个样本。pSrc 指向输入信号，pRef 指向参考信号，pOut 指向输出信号和 pErr 指向误差信号。所有的数组都包括 blockSize 个值。

函数以块为单位操作。滤波器内部系数 b[n] 以样本为单位更新。

算法:

输出信号 y[n] 通过标准 FIR 滤波器计算：

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

```
y[n] = b[0] * x[n] + b[1] * x[n-1] + b[2] * x[n-2] + ...+ b[numTaps-1] * x[n-numTaps+1]
```

误差信号等于参考信号 d[n] 和滤波器输出的差值:

```
e[n] = d[n] - y[n].
```

计算每个误差信号的每个样本后，滤波器状态变量的瞬时能量：

```
E = x[n]^2 + x[n-1]^2 + ... + x[n-numTaps+1]^2.
```

滤波器系数 b[k] 以样本为单位更新:

```
b[k] = b[k] + e[n] * (mu/E) * x[n-k],  for k=0, 1, ..., numTaps-1
```

其中 mu 是步进大小，控制系数收敛速率。

接口中, pCoeffs 指向系数数组，大小是 numTaps. 系数保存的顺序如下：

```
{b[numTaps-1], b[numTaps-2], b[N-2], ..., b[1], b[0]}
```

pState 指向状态数组，数组大小是 numTaps + blockSize - 1. 样本在状态缓存中的保存顺序是：

```
{x[n-numTaps+1], x[n-numTaps], x[n-numTaps-1], x[n-numTaps-2]....x[0], x[1], ...,␣
↪x[blockSize-1]}
```

注意：状态缓存的长度超过了系数数组 blockSize-1 个样本. 增长的状态缓存长度可以取代传统 FIR 滤波器使用的循环寻址，显著提高速度。状态变量在每块数据处理后更新。

结构体实例

滤波器的系数和状态变量都保存在数据结构的实例中。每个滤波器都必须有一个单独的结构体实例。系数数组可能可以在几个实例之间共享，但是状态变量数组不能共享。为支持的 3 种数据类型分别提供了不同的结构体实例声明。

初始化函数

为每种支持的数据类型都提供了一个相应的初始化函数。初始化函数处理以下操作：

- 设置内部结构体字段的值
- 清零状态缓存中的值如果手动初始化，而不调用初始化函数，需要指定结构体实例的以下字段：numTaps, pCoeffs, mu, energy, x0, pState. pState 中的所有值置 0. 对 Q7, Q15, 和 Q31 类型，下列字段必须被初始化：recipTable, postShift

结构体实例不能被放入常量数据段，推荐使用初始化函数初始化它。

定点行为:

使用 Q15 和 Q31 版本的归一化 LMS 滤波器函数需要注意。下列问题必须考虑:

- 系数缩放
- 溢出和饱和

系数缩放:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

滤波器系数表示为一个小数值，被限制在范围 [-1 +1) 之间。定点函数有一个附加的缩放参数 postShift。滤波器的输出累加器是一个可移位的寄存器，结果移动 postShift 位。基本上就是将滤波器系数缩放 2^postShift ，允许将滤波器的系数扩展到超过范围 [+1 -1)。postShift 的值根据用户系统模型期望的增益设定

溢出和饱和:

Q15 和 Q31 版本的溢出和饱和分别描述在各个函数各自的文档部分。

4.11.3 函数说明

4.11.3.1 csky_vdsp2_lms_norm_q31

```
void csky_vdsp2_lms_norm_q31 (csky_vdsp2_lms_norm_instance_q31 *S, q31_t *pSrc, q31_t *pRef, q31_t␣
↪*pOut, q31_t *pErr, uint32_t blockSize)
```

参数:

*S：指向 LMS 滤波器结构体实例

*pSrc：指向输入数据

*pRef：指向参考数据

*pOut：指向输出数据

*pErr：指向误差数据

blockSize：处理的样本的数量

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。累加器是 2.62 格式，并且维持了中间乘法结果的所有精度，但是只有一个保护位。为了防止溢出，输入信号必须缩小 log2(numTaps) 位. 参考信号不应该缩小。在所有的乘累加处理后，2.62 格式累加器右移，然后饱和生成 1.31 最后的结果输出信号和错误信号是 1.31 格式。

这个滤波器中，滤波器系数会根据样本更新，并且系数更新是饱和的。

4.11.3.2 csky_vdsp2_lms_norm_q15

```
void csky_vdsp2_lms_norm_q15 (csky_vdsp2_lms_norm_instance_q15 *S, q15_t *pSrc, q15_t *pRef, q15_t␣
↪*pOut, q15_t *pErr, uint32_t blockSize)
```

参数:

*S：指向 LMS 滤波器结构体实例

*pSrc：指向输入数据

*pRef：指向参考数据

*pOut：指向输出数据

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

*pErr：指向误差数据

blockSize：处理的样本的数量

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。系数和状态变量都表示为 1.15 格式。中间乘法生成 2.30 格式的结果，结果在 34.30 格式的 64 位累加器累加。因为有 33 位保护位，所以不会有溢出的风险。同时还可以保存所有的中间乘法结果的精度。最后，34.30 格式的丢弃低 15 位截断为 34.15，然后饱和生成 1.15 格式的结果。

这个滤波器中，滤波器系数会根据样本更新，并且系数更新是饱和的。

4.11.3.3 csky_vdsp2_lms_norm_init_q31

```
void csky_vdsp2_lms_norm_init_q31 (csky_vdsp2_lms_norm_instance_q31 *S, uint16_t numTaps, q31_t␣
↪*pCoeffs, q31_t *pState, q31_t mu, uint32_t blockSize, uint8_t postShift)
```

参数:

*S：指向 LMS 滤波器结构体实例

numTaps：滤波器系数的数量

*pCoeffs：指向系数缓存

*pState：指向状态缓存

mu：控制滤波器系数更新的步进大小

blockSize：处理的样本的数量

postShift：系数的移位数

返回值:

无

缩放和溢出行为:

pCoeffs 指向滤波器系数的数组，保存的顺序如下：

```
{b[numTaps-1], b[numTaps-2], b[N-2], ..., b[1], b[0]}
```

初始化滤波器系数作为自适应滤波器的起始点。pState 指向一个长度为 numTaps+blockSize-1 样本数组，其中 blockSize 是处理的输入样本数量，传入函数*csky_vdsp2_lms_norm_q31()* 。

4.11.3.4 csky_vdsp2_lms_norm_init_q15

```
void csky_vdsp2_lms_norm_init_q15 (csky_vdsp2_lms_norm_instance_q15 *S, uint16_t numTaps, q15_t␣
↪*pCoeffs, q15_t *pState, q15_t mu, uint32_t blockSize, uint8_t postShift)
```

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

参数:

`*S`: 指向 LMS 滤波器结构体实例

`numTaps`: 滤波器系数的数量

`*pCoeffs`: 指向系数缓存

`*pState`: 指向状态缓存

`mu`: 控制滤波器系数更新的步进大小

`blockSize`: 处理的样本的数量

`postShift`: 系数的移位数

返回值:

无

缩放和溢出行为:

pCoeffs 指向滤波器系数的数组，保存的顺序如下：

```
{b[numTaps-1], b[numTaps-2], b[N-2], ..., b[1], b[0]}
```

初始化滤波器系数作为自适应滤波器的起始点。pState 指向一个长度为 numTaps+blockSize-1 样本数组，其中 blockSize 是处理的输入样本数量，传入函数*csky_vdsp2_lms_norm_q15()* 。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

第五章　矩阵函数

简要说明

这组函数提供了基本的矩阵操作。函数以矩阵数据结构为操作描述矩阵。比如，矩阵数据结构体的类型定义如下：

```
typedef struct
{
  uint16_t numRows;     // number of rows of the matrix.
  uint16_t numCols;     // number of columns of the matrix.
  q31_t *pData;     // points to the data of the matrix.
} csky_vdsp2_matrix_instance_q31;
```

Q15 矩阵的数据类型也类似。

矩阵结构体指定了矩阵的大小，也指定了矩阵的数据数组。数组的大小是 numRows X numCols ，数据按行顺序排列。就是说，矩阵的元素 (i, j) 保存在:

```
pData[i*numCols + j]
```

初始化函数

每种矩阵数据结构体类型都有一个相应的初始化函数。初始化函数设置数据结构内部的数值。函数 csky_vdsp2_mat_init_q31(), csky_vdsp2_mat_init_q31() 和 csky_vdsp2_mat_init_q15() 分别对应 Q31 和 Q15 类型。

是否使用初始化函数是可选的。但是如果使用了初始化函数，则结构体实例不能放在常量数据段。想要将结构体实例放在常量数据段，则需要手动初始化数据结构。比如：

```
csky_vdsp2_matrix_instance_q31 S = {nRows, nColumns, pData};
csky_vdsp2_matrix_instance_q15 S = {nRows, nColumns, pData};
```

其中 nRows 指定行数, nColumns 指定列数, pData 指向数据数组。

5.1 矩阵初始化

5.1.1 函数

- *csky_vdsp2_mat_init_q31*：Q31 矩阵初始化
- *csky_vdsp2_mat_init_q15*：Q15 矩阵初始化

5.1.2 简要说明

初始化矩阵数据结构. 函数设置矩阵结构体的 numRows, numCols, 和 pData 字段

5.1.3 函数说明

5.1.3.1 csky_vdsp2_mat_init_q31

```
void csky_vdsp2_mat_init_q31 (csky_vdsp2_matrix_instance_q31 *S, uint16_t nRows, uint16_t nColumns,
↪ q31_t *pData)
```

参数:

`*S`: 指向一个矩阵结构体实例

`nRows`: 矩阵的行数

`nColumns`: 矩阵的列数

`*pData`: 指向矩阵数据数组

返回值:

无

5.1.3.2 csky_vdsp2_mat_init_q15

```
void csky_vdsp2_mat_init_q15 (csky_vdsp2_matrix_instance_q15 *S, uint16_t nRows, uint16_t nColumns,
↪ q15_t *pData)
```

参数:

`*S`: 指向一个矩阵结构体实例

`nRows`: 矩阵的行数

`nColumns`: 矩阵的列数

`*pData`: 指向矩阵数据数组

返回值:

无

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

5.2 矩阵加法

5.2.1 函数

- *csky_vdsp2_mat_add_q31*：Q31 矩阵加法
- *csky_vdsp2_mat_add_q15*：Q15 矩阵加法

5.2.2 简要说明

两个矩阵相加.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{23}+b_{23} \\ a_{31}+b_{31} & a_{32}+b_{32} & a_{33}+b_{33} \end{bmatrix}$$

图 5.1: 两个 3 x 3 矩阵相加

5.2.3 函数说明

5.2.3.1 csky_vdsp2_mat_add_q31

```
csky_vdsp2_status csky_vdsp2_mat_add_q31 (const csky_vdsp2_matrix_instance_q31 *pSrcA, const csky_
↪vdsp2_matrix_instance_q31 *pSrcB, csky_vdsp2_matrix_instance_q31 *pDst)
```

参数:

*pSrcA: 指向第一个输入矩阵结构

*pSrcB: 指向第二个输入矩阵结构

*pDst: 指向输出矩阵结构

返回值:

无

缩放和溢出行为:

函数使用饱和计算。超过 Q31 最大范围 [0x80000000 0x7FFFFFFF] 的结果会被饱和。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

5.2.3.2 csky_vdsp2_mat_add_q15

```
csky_vdsp2_status csky_vdsp2_mat_add_q15 (const csky_vdsp2_matrix_instance_q15 *pSrcA, const csky_
↪vdsp2_matrix_instance_q15 *pSrcB, csky_vdsp2_matrix_instance_q15 *pDst)
```

参数:

*pSrcA: 指向第一个输入矩阵结构

*pSrcB: 指向第二个输入矩阵结构

*pDst: 指向输出矩阵结构

返回值:

无

缩放和溢出行为:

函数使用饱和计算。超过 Q15 最大范围 [0x8000 0x7FFF] 的结果会被饱和。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

5.3 矩阵减法

5.3.1 函数

- *csky_vdsp2_mat_sub_q31*：Q31 矩阵相减
- *csky_vdsp2_mat_sub_q15*：Q15 矩阵相减

5.3.2 简要说明

两个矩阵相减

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} - \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}-b_{11} & a_{12}-b_{12} & a_{13}-b_{13} \\ a_{21}-b_{21} & a_{22}-b_{22} & a_{23}-b_{23} \\ a_{31}-b_{31} & a_{32}-b_{32} & a_{33}-b_{33} \end{bmatrix}$$

图 5.2: 两个 3 x 3 矩阵相减

5.3.3 函数说明

5.3.3.1 csky_vdsp2_mat_sub_q31

```
csky_vdsp2_status csky_vdsp2_mat_sub_q31 (const csky_vdsp2_matrix_instance_q31 *pSrcA, const csky_
↪vdsp2_matrix_instance_q31 *pSrcB, csky_vdsp2_matrix_instance_q31 *pDst)
```

参数:

`*pSrcA`: 指向第一个输入矩阵结构体

`*pSrcB`: 指向第二个输入矩阵结构体

`*pDst`: 指向输出矩阵结构体

返回值:

无

缩放和溢出行为:

函数使用饱和计算. 结果如果超出 Q31 的最大范围 [0x80000000 0x7FFFFFFF] 会被饱和.

5.3.3.2 csky_vdsp2_mat_sub_q15

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

```
csky_vdsp2_status csky_vdsp2_mat_sub_q15 (const csky_vdsp2_matrix_instance_q15 *pSrcA, const csky_
↪vdsp2_matrix_instance_q15 *pSrcB, csky_vdsp2_matrix_instance_q15 *pDst)
```

参数:

`*pSrcA`: 指向第一个输入矩阵结构体

`*pSrcB`: 指向第二个输入矩阵结构体

`*pDst`: 指向输出矩阵结构体

返回值:

无

缩放和溢出行为:

函数使用饱和计算. 结果如果超出 Q15 的最大范围 [0x8000 0x7FFF] 会被饱和.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

5.4 复数矩阵乘法

5.4.1 函数

- *csky_vdsp2_mat_cmplx_mult_q31*：Q31 复数矩阵乘法
- *csky_vdsp2_mat_cmplx_mult_q15*：Q15 复数矩阵乘法

5.4.2 简要说明

只有当第一个矩阵的列数和第二个矩阵的行数相等的时候，才可以做复数矩阵乘法。M x N 矩阵和 N x P 矩阵相乘的结果是一个 M x P 矩阵。当使能矩阵大小检查，函数会检查:

1. pSrcA 和 pSrcB 的内部尺寸是否相等
2. 输出向量的尺寸是否跟 pSrcA 和 pSrcB 的计算结果相等

5.4.3 函数说明

5.4.3.1 csky_vdsp2_mat_cmplx_mult_q31

```
csky_vdsp2_status csky_vdsp2_mat_cmplx_mult_q31 (const csky_vdsp2_matrix_instance_q31 *pSrcA,
const csky_vdsp2_matrix_instance_q31 *pSrcB, csky_vdsp2_matrix_instance_q31 *pDst)
```

参数:

*pSrcA: 指向第一个输入矩阵结构

*pSrcB: 指向第二个输入矩阵结构

*pDst: 指向输出矩阵结构

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。累加器用 2.62 格式维持了中间结果的所有精度，但是只有一个保护位。累加时候没有饱和计算，因此累加器溢出会扭曲结果，需要缩小输入信号来防止中间结果溢出。因为最多会有 numColsA 个加法进位，所以需要缩小 log2(numColsA) 位来防止溢出。2.62 格式的累加器右移 31 位，然后饱和生成 1.31 格式的最终结果。

5.4.3.2 csky_vdsp2_mat_cmplx_mult_q15

```
csky_vdsp2_status csky_vdsp2_mat_cmplx_mult_q15 (const csky_vdsp2_matrix_instance_q15 *pSrcA,
const csky_vdsp2_matrix_instance_q15 *pSrcB, csky_vdsp2_matrix_instance_q15 *pDst)
```

参数:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

*pSrcA: 指向第一个输入矩阵结构

*pSrcB: 指向第二个输入矩阵结构

*pDst: 指向输出矩阵结构

返回值:

无

最佳性能的条件:

输入，输出，和缓存都需要 32 位对齐

限制:

如果芯片不支持分对齐访问，则定义宏 UNALIGNED_SUPPORT_DISABLE 同时，输入，输出，临时 buffer，都应该是 32 位对齐。

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。输入都是 1.15 格式的，乘法生成 2.30 结果。2.30 格式的中间结果在 34.30 格式的 64 位累加器累加。由于有 33 个保护位，所以不会有溢出风险。34.30 的结果丢弃低 15 位截断为 34.15 格式，然后饱和成为 1.15 格式的结果。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

5.5 矩阵乘法

5.5.1 函数

- *csky_vdsp2_mat_mult_q31*：Q31 矩阵乘法
- *csky_vdsp2_mat_mult_q15*：Q15 矩阵乘法
- *csky_vdsp2_mat_mult_trans_q31*：Q31 矩阵乘法
- *csky_vdsp2_mat_mult_trans_q15*：Q15 矩阵乘法

5.5.2 简要说明

两个矩阵相乘

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \text{x} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}\text{x}b_{11}+a_{12}\text{x}b_{21}+a_{13}\text{x}b_{31} & a_{11}\text{x}b_{12}+a_{12}\text{x}b_{22}+a_{13}\text{x}b_{32} & a_{11}\text{x}b_{13}+a_{12}\text{x}b_{23}+a_{13}\text{x}b_{33} \\ a_{21}\text{x}b_{11}+a_{22}\text{x}b_{21}+a_{23}\text{x}b_{31} & a_{21}\text{x}b_{12}+a_{22}\text{x}b_{22}+a_{23}\text{x}b_{32} & a_{21}\text{x}b_{13}+a_{22}\text{x}b_{23}+a_{23}\text{x}b_{33} \\ a_{31}\text{x}b_{11}+a_{32}\text{x}b_{21}+a_{33}\text{x}b_{31} & a_{31}\text{x}b_{12}+a_{32}\text{x}b_{22}+a_{33}\text{x}b_{32} & a_{31}\text{x}b_{13}+a_{32}\text{x}b_{23}+a_{33}\text{x}b_{33} \end{bmatrix}$$

图 5.3: 两个 3 x 3 矩阵相乘

只有第一个矩阵的列数和第二个矩阵的行数相等的时候，才可以做矩阵乘法。M x N 矩阵和 N x P 矩阵相乘的结果是一个 M x P 矩阵。

5.5.3 函数说明

5.5.3.1 csky_vdsp2_mat_mult_q31

```
csky_vdsp2_status csky_vdsp2_mat_mult_q31 (const csky_vdsp2_matrix_instance_q31 *pSrcA, const csky_
↪vdsp2_matrix_instance_q31 *pSrcB, csky_vdsp2_matrix_instance_q31 *pDst)
```

参数:

`*pSrcA`: 指向第一个输入矩阵结构

`*pSrcB`: 指向第二个输入矩阵结构

`*pDst`: 指向输出矩阵结构

返回值:

无

缩放和溢出行为:

函数实现使用了一个 64 位的内部累加器。累加器使用 2.62 格式，维持了中间乘法结果的所有精度，但是只提供了 1 个保护位。累加过程中没有饱和处理，累加器溢出会导致结果扭曲，因此必须缩小输入信号，防止溢出。因为最多会有 numColsA 个加法进位，所以输入矩阵需要缩小 log2(numColsA) 个位，来防止溢出。2.62 格式的累加器右移 31 位，饱和生成 1.31 的最后结果。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

5.5.3.2 csky_vdsp2_mat_mult_q15

```
csky_vdsp2_status csky_vdsp2_mat_mult_q15 (const csky_vdsp2_matrix_instance_q15 *pSrcA, const csky_
↪vdsp2_matrix_instance_q15 *pSrcB, csky_vdsp2_matrix_instance_q15 *pDst)
```

参数:

*pSrcA: 指向第一个输入矩阵结构

*pSrcB: 指向第二个输入矩阵结构

*pDst: 指向输出矩阵结构

返回值:

无

缩放和溢出行为:

函数实现使用了一个 64 位的内部累加器。输入都是 1.15 格式，相乘产生的结果是 2.30 格式。2.30 的中间结果在 34.30 格式的累加器累加。因为提供了 33 个保护位，所有不会有溢出风险。34.30 的结果丢弃低 15 位，截断为 34.15 格式，然后饱和成 1.15 格式的最后结果

5.5.3.3 csky_vdsp2_mat_mult_trans_q31

```
csky_vdsp2_status csky_vdsp2_mat_mult_trans_q31 (const csky_vdsp2_matrix_instance_q31 *pSrcA,␣
↪const csky_vdsp2_matrix_instance_q31 *pSrcB, csky_vdsp2_matrix_instance_q31 *pDst)
```

参数:

*pSrcA: 指向第一个输入矩阵结构

*pSrcB: 指向第二个输入矩阵结构

*pDst: 指向输出矩阵结构

返回值:

无

缩放和溢出行为:

这个版本和函数*csky_vdsp2_mat_mult_q31()* 的区别在于，它的第二个输入矩阵 pSrcB 已经经过转置，在数据读取和乘累加的计算上都有很大的便利。其他处理没有细节没有区别。

5.5.3.4 csky_vdsp2_mat_mult_trans_q15

```
csky_vdsp2_status csky_vdsp2_mat_mult_trans_q15 (const csky_vdsp2_matrix_instance_q15 *pSrcA,␣
↪const csky_vdsp2_matrix_instance_q15 *pSrcB, csky_vdsp2_matrix_instance_q15 *pDst)
```

参数:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

`*pSrcA`: 指向第一个输入矩阵结构

`*pSrcB`: 指向第二个输入矩阵结构

`*pDst`: 指向输出矩阵结构

返回值:

无

缩放和溢出行为:

这个版本和函数*csky_vdsp2_mat_mult_q15()* 的区别在于，它的第二个输入矩阵 pSrcB 已经经过转置，在数据读取和乘累加的计算上都有很大的便利。其他处理没有细节没有区别。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

5.6 矩阵缩放

5.6.1 函数

- *csky_vdsp2_mat_scale_q31* : Q31 矩阵缩放
- *csky_vdsp2_mat_scale_q15* : Q15 矩阵缩放

5.6.2 简要说明

矩阵与标量相乘，矩阵中的每个元素都与一个标量相乘比如:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times K = \begin{bmatrix} a_{11} \times K & a_{12} \times K & a_{13} \times K \\ a_{21} \times K & a_{22} \times K & a_{23} \times K \\ a_{31} \times K & a_{32} \times K & a_{33} \times K \end{bmatrix}$$

图 5.4: 3x3 矩阵缩放

Q15 和 Q31 的定点函数, scale 表示为一个小数乘法 scaleFract 和一个移位 shift. 移位让缩放操作的增益可以超过 1.0。定点数据的总的缩放因子是:

```
scale = scaleFract * 2^shift.
```

5.6.3 函数说明

5.6.3.1 csky_vdsp2_mat_scale_q31

```
csky_vdsp2_status csky_vdsp2_mat_scale_q31 (const csky_vdsp2_matrix_instance_q31 *pSrc, q31_t␣
↪scaleFract, int32_t shift, csky_vdsp2_matrix_instance_q31 *pDst)
```

参数:

`*pSrc`: 指向输入矩阵结构体

`scaleFract`: 缩放因子

`shift`: 结果的移位数量

`*pDst`: 指向输出矩阵结构体

返回值:

无

缩放和溢出行为:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

输入数据 pSrc 和 scaleFract 都是 1.31 格式. 它们相乘的生成 2.62 的中间结果，然后移位和饱和成 1.31 格式

5.6.3.2 csky_vdsp2_mat_scale_q15

```
csky_vdsp2_status csky_vdsp2_mat_scale_q15 (const csky_vdsp2_matrix_instance_q15 *pSrc, q15_t␣
↪scaleFract, int32_t shift, csky_vdsp2_matrix_instance_q15 *pDst)
```

参数:

`*pSrc`: 指向输入矩阵结构体

`scaleFract`: 缩放因子

`shift`: 结果的移位数量

`*pDst`: 指向输出矩阵结构体

返回值:

无

缩放和溢出行为:

输入数据 pSrc 和 scaleFract 都是 1.15 格式. 它们相乘的生成 2.30 的中间结果，然后移位和饱和成 1.15 格式

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

5.7 矩阵转置

5.7.1 函数

- *csky_vdsp2_mat_trans_q31*：Q31 矩阵转置
- *csky_vdsp2_mat_trans_q15*：Q15 矩阵转置

5.7.2 简要说明

转置一个矩阵. 转置一个 M x N 就是让它按中心对角线翻转出一个 N x M 矩阵

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}$$

转置一个 3 x 3 矩阵

5.7.3 函数说明

5.7.3.1 csky_vdsp2_mat_trans_q31

```
csky_vdsp2_status csky_vdsp2_mat_trans_q31 (const csky_vdsp2_matrix_instance_q31 *pSrc, csky_vdsp2_
↪matrix_instance_q31 *pDst)
```

参数:

*pSrc: 指向输入矩阵

*pDst: 指向输出矩阵

返回值:

无

5.7.3.2 csky_vdsp2_mat_trans_q15

```
csky_vdsp2_status csky_vdsp2_mat_trans_q15 (const csky_vdsp2_matrix_instance_q15 *pSrc, csky_vdsp2_
↪matrix_instance_q15 *pDst)
```

参数:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

`*pSrc`：指向输入矩阵
`*pDst`：指向输出矩阵

返回值：

无

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

第六章　统计函数

6.1　最大值

6.1.1　函数

- *csky_vdsp2_max_q31*：Q31 数组中的最大值
- *csky_vdsp2_max_q15*：Q15 数组中的最大值
- *csky_vdsp2_max_q7*：Q7 数组中的最大值

6.1.2　简要说明

取一个数组中的最大值。函数返回最大值和最大值在数组中的位置。为 Q31, Q15 和 Q7 数据类型分别提供不同的函数。

6.1.3　函数说明

6.1.3.1　csky_vdsp2_max_q31

```
void csky_vdsp2_max_q31 (q31_t *pSrc, uint32_t blockSize, q31_t *pResult, uint32_t *pIndex)
```

参数:

`*pSrc`: 指向输入数组

`blockSize`: 输入数组的长度

`*pResult`: 返回的最大值

`*pIndex`: 返回的最大值的索引

返回值:

无

6.1.3.2　csky_vdsp2_max_q15

```
void csky_vdsp2_max_q15 (q15_t *pSrc, uint32_t blockSize, q15_t *pResult, uint32_t *pIndex)
```

参数:

*pSrc: 指向输入数组

blockSize: 输入数组的长度

*pResult: 返回的最大值

*pIndex: 返回的最大值的索引

返回值:

无

6.1.3.3 csky__vdsp2__max__q7

```
void csky_vdsp2_max_q7 (q7_t *pSrc, uint32_t blockSize, q7_t *pResult, uint32_t *pIndex)
```

参数:

*pSrc: 指向输入数组

blockSize: 输入数组的长度

*pResult: 返回的最大值

*pIndex: 返回的最大值的索引

返回值:

无

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

6.2 最小值

6.2.1 函数

- *csky_vdsp2_min_q31*：Q31 数组的最小值
- *csky_vdsp2_min_q15*：Q15 数组的最小值
- *csky_vdsp2_min_q7*：Q7 数组的最小值

6.2.2 简要说明

计算数组数据中的最小值. 函数返回最小值和最小值在数组中的位置。为 Q31，Q15，Q7 分别提供了不同的函数。

6.2.3 函数说明

6.2.3.1 csky_vdsp2_min_q31

```
void csky_vdsp2_min_q31 (q31_t *pSrc, uint32_t blockSize, q31_t *pResult, uint32_t *pIndex)
```

参数:

`*pSrc`: 指向输入数组

`blockSize`: 输入数组的长度

`*pResult`: 返回的最大值

`*pIndex`: 返回的最大值的索引

返回值:

无

6.2.3.2 csky_vdsp2_min_q15

```
void csky_vdsp2_min_q15 (q15_t *pSrc, uint32_t blockSize, q15_t *pResult, uint32_t *pIndex)
```

参数:

`*pSrc`: 指向输入数组

`blockSize`: 输入数组的长度

`*pResult`: 返回的最大值

`*pIndex`: 返回的最大值的索引

返回值:

无

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

6.2.3.3 csky__vdsp2__min__q7

```
void csky_vdsp2_min_q7 (q7_t *pSrc, uint32_t blockSize, q7_t *pResult, uint32_t *pIndex)
```

参数:

*pSrc: 指向输入数组

blockSize: 输入数组的长度

*pResult: 返回的最大值

*pIndex: 返回的最大值的索引

返回值:

无

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

6.3 平均值

6.3.1 函数

- *csky_vdsp2_mean_q31*：Q31 数组的平均值
- *csky_vdsp2_mean_q15*：Q15 数组的平均值
- *csky_vdsp2_mean_q7*：Q7 数组的平均值

6.3.2 简要说明

计算输入数组的平均值。数组中的所有元素的平均值。使用下面的算法:

```
Result = (pSrc[0] + pSrc[1] + pSrc[2] + ... + pSrc[blockSize-1]) / blockSize;
```

为 Q31，Q15 和 Q7 数据类型分别提供了不同的函数

6.3.3 函数说明

6.3.3.1 csky_vdsp2_mean_q31

```
void csky_vdsp2_mean_q31 (q31_t *pSrc, uint32_t blockSize, q31_t *pResult)
```

参数:

`*pSrc`: 指向输入数组

`blockSize`: 输入数组的长度

`*pResult`: 返回的平均值

返回值:

无

缩放和溢出行为:

函数实现使用了一个 64 位内部累加器。输入表示为 1.31 格式，累加器的格式是 33.31。因此可以保留所有的中间结果精度，不会有溢出风险。最后，结果截断为 1.31 格式。

6.3.3.2 csky_vdsp2_mean_q15

```
void csky_vdsp2_mean_q15 (q15_t *pSrc, uint32_t blockSize, q15_t *pResult)
```

参数:

`*pSrc`: 指向输入数组

`blockSize`: 输入数组的长度

`*pResult`: 返回的平均值

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

返回值:

无

缩放和溢出行为:

函数实现使用一个内部 32 位累加器。输入表示为 1.15 格式，32 位累加器用 17.15 格式。因此可以保留所有的精度，不会有溢出风险。最后，累加器饱和截断生成 1.15 格式的结果。

6.3.3.3 csky__vdsp2__mean__q7

```
void csky_vdsp2_mean_q7 (q7_t *pSrc, uint32_t blockSize, q7_t *pResult)
```

参数:

*pSrc: 指向输入数组

blockSize: 输入数组的长度

*pResult: 返回的平均值

返回值:

无

缩放和溢出行为:

函数实现使用了一个 32 位内部累加器。输入表示为 1.7 格式，累加器的格式是 25.7。因此可以保留所有的中间结果精度，不会有溢出风险。最后，结果截断为 1.7 格式。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

6.4 平方和

6.4.1 函数

- *csky_vdsp2_power_int32*：32 位整数所有元素的平方之和
- *csky_vdsp2_power_q31*：Q31 数组中所有元素的平方和
- *csky_vdsp2_power_q15*：Q15 数组中所有元素的平方和
- *csky_vdsp2_power_q7*：Q7 数组中所有元素的平方和

6.4.2 简要说明

计算输入数组中元素的平方和. 算法如下:

```
Result = pSrc[0] * pSrc[0] + pSrc[1] * pSrc[1] + pSrc[2] * pSrc[2] + ... +
pSrc[blockSize-1] * pSrc[blockSize-1];
```

为 Q31，Q15，Q7 分别提供了不同的函数。

6.4.3 函数说明

6.4.3.1 csky_vdsp2_power_int32

```
void csky_vdsp2_power_int32 (int32 *pSrc, uint32_t blockSize, q63_t *pResult)
```

参数:

*pSrc: 指向输入数组

blockSize: 输入数组的长度

*pResult: 返回的平方和

返回值:

无

缩放和溢出行为:

这个函数的输入为 32 位整数，元素平方后扩展为 64 位整数，相加使用了 64 位累加器。相加结果会被饱和，最后的结果会在 [0x0, 0x7fffffffffffffff] 之间。因此，在使用时，需要注意溢出。

6.4.3.2 csky_vdsp2_power_q31

```
void csky_vdsp2_power_q31 (q31_t *pSrc, uint32_t blockSize, q31_t *pResult)
```

参数:

*pSrc：指向输入数组

blockSize：输入数组的长度

*pResult：返回的平方和

返回值：

无

缩放和溢出行为：

函数实现使用了一个内部 64 位累加器。输入数据表示为 1.31 格式。中间乘法生成 2.62 格式，然后截断为 2.48 格式的结果，结果在 16.48 格式的 64 位累加器累加。因为有 15 位保护位，所以不会有溢出的风险。同时还可以保存所有的中间乘法结果的精度. 最后，返回结果是 16.48 格式.

6.4.3.3 csky_vdsp2_power_q15

```
void csky_vdsp2_power_q15 (q15_t *pSrc, uint32_t blockSize, q15_t *pResult)
```

参数：

*pSrc：指向输入数组

blockSize：输入数组的长度

*pResult：返回的平方和

返回值：

无

缩放和溢出行为：

函数实现使用了一个内部 64 位累加器。输入数据表示为 1.15 格式。中间乘法生成 2.30 格式的结果，结果在 34.30 格式的 64 位累加器累加。因为有 33 位保护位，所以不会有溢出的风险。同时还可以保存所有的中间乘法结果的精度。最后，返回结果是 34.30 格式.

6.4.3.4 csky_vdsp2_power_q7

```
void csky_vdsp2_power_q7 (q7_t *pSrc, uint32_t blockSize, q7_t *pResult)
```

参数：

*pSrc：指向输入数组

blockSize：输入数组的长度

*pResult：返回的平方和

返回值：

无

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

缩放和溢出行为:

函数实现使用了一个内部 32 位累加器。输入数据表示为 1.7 格式。中间乘法生成 2.14 格式的结果，结果在 18.14 格式的 64 位累加器累加。因为有 17 位保护位，所以不会有溢出的风险。同时还可以保存所有的中间乘法结果的精度最后，返回结果是 18.14 格式.

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

6.5 均方根 (RMS)

6.5.1 函数

- *csky_vdsp2_rms_q31*：Q31 数组元素的均方根
- *csky_vdsp2_rms_q15*：Q15 数组元素的均方根

6.5.2 简要说明

计算输入数组中所有元素的均方根. 算法如下:

```
Result = sqrt(((pSrc[0] * pSrc[0] + pSrc[1] * pSrc[1] + ... + pSrc[blockSize-1] *␣
↪pSrc[blockSize-1]) / blockSize));
```

为 Q31，Q15 分别提供了不同的函数。

6.5.3 函数说明

6.5.3.1 csky_vdsp2_rms_q31

```
void csky_vdsp2_rms_q31 (float32_t *pSrc, uint32_t blockSize, float32_t *pResult)
```

参数:

`*pSrc`: 指向输入数组

`blockSize`: 输入数组的长度

`*pResult`: 返回的均方根结果

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。输入数据表示为 1.31 格式。中间乘法生成 2.62 格式的结果，结果在 2.62 格式的 64 位累加器累加。因为只有一个保护位，所以需要缩小输入信号来防止溢出。因为最多会有 blockSize 个加法进位，所以需要缩小 log2(blockSize) 才可以保证没有溢出。最后，2.62 格式右移 31 位，然后饱和生成 1.15 格式的结果。

6.5.3.2 csky_vdsp2_rms_q15

```
void csky_vdsp2_rms_q15 (float32_t *pSrc, uint32_t blockSize, float32_t *pResult)
```

参数:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

`*pSrc`: 指向输入数组

`blockSize`: 输入数组的长度

`*pResult`: 返回的均方根结果

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。输入数据表示为 1.15 格式。中间乘法生成 2.30 格式的结果，结果在 34.30 格式的 64 位累加器累加。因为有 33 位保护位，所以不会有溢出的风险。同时还可以保存所有的中间乘法结果的精度。最后，34.30 格式的丢弃低 15 位截断为 34.15，然后饱和生成 1.15 格式的结果。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

6.6 标准偏差

6.6.1 函数

- *csky_vdsp2_std_q31*：Q31 数组元素的标准偏差
- *csky_vdsp2_std_q15*：Q15 数组元素的标准偏差

6.6.2 简要说明

计算输入数组元素的标准偏差使用的算法如下:

```
Result = sqrt((sumOfSquares - sum2 / blockSize) / (blockSize - 1))

where, sumOfSquares = pSrc[0] * pSrc[0] + pSrc[1] * pSrc[1] + ... + pSrc[blockSize-1] *␣
↪pSrc[blockSize-1]
                    sum = pSrc[0] + pSrc[1] + pSrc[2] + ... + pSrc[blockSize-1]
```

为 Q31，Q15 数据类型分别提供不同的函数。

6.6.3 函数说明

6.6.3.1 csky_vdsp2_std_q31

```
void csky_vdsp2_std_q31 (float32_t *pSrc, uint32_t blockSize, float32_t *pResult)
```

参数:

*pSrc: 指向输入数组

blockSize: 输入数组的长度

*pResult: 返回的标准偏差

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。输入数据表示为 1.31 格式，然后移位 8 位，生成 1.23 格式。中间乘法生成 2.46 格式的结果，结果在 18.46 格式的 64 位累加器累加。除法之后，中间结果应该是 18.46 格式。最后，18.46 格式右移 15 位，然后饱和生成 1.31 格式的结果。

6.6.3.2 csky_vdsp2_std_q15

```
void csky_vdsp2_std_q15 (float32_t *pSrc, uint32_t blockSize, float32_t *pResult)
```

参数:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

*pSrc: 指向输入数组

blockSize: 输入数组的长度

*pResult: 返回的标准偏差

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。输入数据表示为 1.15 格式。中间乘法生成 2.30 格式的结果，结果在 34.30 格式的 64 位累加器累加。因为有 33 位保护位，所以不会有溢出的风险。同时还可以保存所有的中间乘法结果的精度。最后，34.30 格式的丢弃低 15 位截断为 34.15，然后饱和生成 1.15 格式的结果。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

6.7 方差

6.7.1 函数

- *csky_vdsp2_var_q31*：Q31 数组元素的方差
- *csky_vdsp2_var_q15*：Q15 数组元素的方差

6.7.2 简要说明

计算输入数组元素的方差。使用的算法如下：

```
Result = (sumOfSquares - sum2 / blockSize) / (blockSize - 1)

where, sumOfSquares = pSrc[0] * pSrc[0] + pSrc[1] * pSrc[1] + ... + pSrc[blockSize-1] *␣
↪pSrc[blockSize-1]
                    sum = pSrc[0] + pSrc[1] + pSrc[2] + ... + pSrc[blockSize-1]
```

为 Q31 和 Q15 分别提供了不同的函数。

6.7.3 函数说明

6.7.3.1 csky_vdsp2_var_q31

```
void csky_vdsp2_var_q31 (float32_t *pSrc, uint32_t blockSize, float32_t *pResult)
```

参数:

*pSrc: 指向输入数组

blockSize: 输入数组的长度

*pResult: 返回的方差

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。输入数据表示为 1.31 格式，然后移位 8 位，生成 1.23 格式。中间乘法生成 2.46 格式的结果，结果在 18.46 格式的 64 位累加器累加。除法之后，中间结果应该是 18.46 格式。最后，18.46 格式右移 15 位，然后饱和生成 1.31 格式的结果。

6.7.3.2 csky_vdsp2_var_q15

```
void csky_vdsp2_var_q15 (float32_t *pSrc, uint32_t blockSize, float32_t *pResult)
```

参数:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

`*pSrc`: 指向输入数组

`blockSize`: 输入数组的长度

`*pResult`: 返回的方差

返回值:

无

缩放和溢出行为:

函数实现使用了一个内部 64 位累加器。输入数据表示为 1.15 格式。中间乘法生成 2.30 格式的结果，结果在 34.30 格式的 64 位累加器累加。因为有 33 位保护位，所以不会有溢出的风险。同时还可以保存所有的中间乘法结果的精度。最后，34.30 格式的丢弃低 15 位截断为 34.15，然后饱和生成 1.15 格式的结果。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

第七章　辅助函数

7.1　向量复制

7.1.1　函数

- *csky_vdsp2_copy_q31*：复制 Q31 向量元素
- *csky_vdsp2_copy_q15*：复制 Q15 向量元素
- *csky_vdsp2_copy_q7*：复制 Q7 向量元素

7.1.2　简要说明

把样本一个个从源向量复制到目的向量。

```
pDst[n] = pSrc[n];   0 <= n < blockSize.
```

为 Q31，Q15 和 Q7 数据类型分别提供不同的函数。

7.1.3　函数说明

7.1.3.1　csky_vdsp2_copy_q31

```
void csky_vdsp2_copy_q31 (q31_t *pSrc, q31_t *pDst, uint32_t blockSize)
```

参数:

`*pSrc`: 指向输入向量

`*pDst`: 指向输出向量

`blockSize`: 输入向量的元素数量

返回值:

无

7.1.3.2 csky_vdsp2_copy_q15

```
void csky_vdsp2_copy_q15 (q15_t *pSrc, q15_t *pDst, uint32_t blockSize)
```

参数:

*pSrc: 指向输入向量

*pDst: 指向输出向量

blockSize: 输入向量的元素数量

返回值:

无

7.1.3.3 csky_vdsp2_copy_q7

```
void csky_vdsp2_copy_q7 (q7_t *pSrc, q7_t *pDst, uint32_t blockSize)
```

参数:

*pSrc: 指向输入向量

*pDst: 指向输出向量

blockSize: 输入向量的元素数量

返回值:

无

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

7.2 向量填充

7.2.1 函数

- *csky_vdsp2_fill_q31*：用常量填充 Q31 向量
- *csky_vdsp2_fill_q15*：用常量填充 Q31 向量
- *csky_vdsp2_fill_q7*：用常量填充 Q7 向量 v

7.2.2 简要说明

用一个常量值填充目的向量.

```
pDst[n] = value;   0 <= n < blockSize.
```

为 Q31，Q15 和 Q7 数据类型分别提供不同的函数。

7.2.3 函数说明

7.2.3.1 csky_vdsp2_fill_q31

```
void csky_vdsp2_fill_q31 (q31_t value, q31_t *pDst, uint32_t blockSize)
```

参数:

`value`: 用来填充的值

`*pDst`: 指向输出向量

`blockSize`: 输出向量的元素数量

返回值:

无

7.2.3.2 csky_vdsp2_fill_q15

```
void csky_vdsp2_fill_q15 (q15_t value, q15_t *pDst, uint32_t blockSize)
```

参数:

`value`: 用来填充的值

`*pDst`: 指向输出向量

`blockSize`: 输出向量的元素数量

返回值:

无

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

7.2.3.3 csky_vdsp2_fill_q7

```
void csky_vdsp2_fill_q7 (q7_t value, q7_t *pDst, uint32_t blockSize)
```

参数:

value: 用来填充的值

*pDst: 指向输出向量

blockSize: 输出向量的元素数量

返回值:

无

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

7.3 转换 Q15 的值

7.3.1 函数

- *csky_vdsp2_q15_to_q31*：转换 Q15 向量元素到 Q31 向量
- *csky_vdsp2_q15_to_q7*：转换 Q15 向量元素到 Q7 向量

7.3.2 函数说明

7.3.2.1 csky_vdsp2_q15_to_q31

```
void csky_vdsp2_q15_to_q31 (q15_t *pSrc, q31_t *pDst, uint32_t blockSize)
```

参数:

*pSrc: 指向输入的向量

*pDst: 指向输出的向量

blockSize: 输入向量的元素数量

返回值:

无

简要说明:

转换过程的公式如下:

```
pDst[n] = (q31_t) pSrc[n] << 16;   0 <= n < blockSize.
```

7.3.2.2 csky_vdsp2_q15_to_q7

```
void csky_vdsp2_q15_to_q7 (q15_t *pSrc, q7_t *pDst, uint32_t blockSize)
```

参数:

*pSrc: 指向输入的向量

*pDst: 指向输出的向量

blockSize: 输入向量的元素数量

返回值:

无

简要说明:

转换过程的公式如下:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

```
pDst[n] = (q7_t) pSrc[n] >> 8;   0 <= n < blockSize.
```

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

7.4 转换 Q31 的值

7.4.1 函数

- *csky_vdsp2_q31_to_q15*：转换 Q31 向量元素到 Q15 向量
- *csky_vdsp2_q31_to_q7*：转换 Q31 向量元素到 Q7 向量
- *csky_vdsp2_q31_to_q7_rs*：转换 32 位定点数向量元素到 8 位定点数向量, 结果进行了舍入及饱和处理
- *csky_vdsp2_q63_to_q31_rs*：转换 64 位定点数向量元素到 32 位定点数向量，结果进行了舍入及饱和处理

7.4.2 函数说明

7.4.2.1 csky_vdsp2_q31_to_q15

```
void csky_vdsp2_q31_to_q15 (q31_t *pSrc, q15_t *pDst, uint32_t blockSize)
```

参数:

*pSrc: 指向输入的向量
*pDst: 指向输出的向量
blockSize: 输入向量的元素数量

返回值:

无

简要说明:

转换过程的公式如下:

```
pDst[n] = (q15_t) pSrc[n] >> 16;   0 <= n < blockSize.
```

7.4.2.2 csky_vdsp2_q31_to_q7

```
void csky_vdsp2_q31_to_q7 (q31_t *pSrc, q7_t *pDst, uint32_t blockSize)
```

参数:

*pSrc: 指向输入的向量
*pDst: 指向输出的向量
blockSize: 输入向量的元素数量

返回值:

无

简要说明:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

转换过程的公式如下:

```
pDst[n] = (q7_t) pSrc[n] >> 24;   0 <= n < blockSize.
```

7.4.2.3 csky_vdsp2_q31_to_q7_rs

```
void csky_vdsp2_q31_to_q7_rs (q31_t *pSrc, q7_t *pDst, uint32_t shiftValue, uint32_t blockSize)
```

参数:

*pSrc: 指向输入的向量
*pDst: 指向输出的向量
shiftValue: 右移位数
blockSize: 输入向量的元素数量

返回值:

无

简要说明:

转换过程的公式如下:

```
pDst[n] = (q7_t) (pSrc[n] >> shiftValue + round);   0 <= n < blockSize.
```

缩放和溢出时的行为:

函数使用饱和算法。输出结果为 Q7，范围是 [0x80, 0x7F].

7.4.2.4 csky_vdsp2_q63_to_q31_rs

```
void csky_vdsp2_q63_to_q31_rs (q63_t *pSrc, q31_t *pDst, uint32_t shiftValue, uint32_t blockSize)
```

参数:

*pSrc: 指向输入的向量
*pDst: 指向输出的向量
shiftValue: 右移位数
blockSize: 输入向量的元素数量

返回值:

无

简要说明:

转换过程的公式如下:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

```
pDst[n] = (q31_t) (pSrc[n] >> shiftValue + round);   0 <= n < blockSize.
```

缩放和溢出时的行为:

函数使用饱和算法。输出结果为 Q31，范围是 [0x80000000, 0x7FFFFFFF].

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

7.5 转换 Q7 的值

7.5.1 函数

- *csky_vdsp2_q7_to_q15*：转换 Q7 向量元素到 Q15 向量
- *csky_vdsp2_q7_to_q31*：转换 Q7 向量元素到 Q31 向量

7.5.2 函数说明

7.5.2.1 csky_vdsp2_q7_to_q15

```
void csky_vdsp2_q7_to_q15 (q7_t *pSrc, q15_t *pDst, uint32_t blockSize)
```

参数:

*pSrc: 指向输入的向量

*pDst: 指向输出的向量

blockSize: 输入向量的元素数量

返回值:

无

简要说明:

转换过程的公式如下:

```
pDst[n] = (q15_t) pSrc[n] << 8;   0 <= n < blockSize.
```

7.5.2.2 csky_vdsp2_q7_to_q31

```
void csky_vdsp2_q7_to_q31 (q7_t *pSrc, q31_t *pDst, uint32_t blockSize)
```

参数:

*pSrc: 指向输入的向量

*pDst: 指向输出的向量

blockSize: 输入向量的元素数量

返回值:

无

简要说明:

转换过程的公式如下:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

```
pDst[n] = (q31_t) pSrc[n] << 24;   0 <= n < blockSize.
```

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

第八章　变换函数

8.1　复数 FFT 函数

8.1.1　函数

- *csky_vdsp2_cfft_q15*：Q15 复数 FFT 处理函数
- *csky_vdsp2_cfft_q31*：Q31 复数 FFT 处理函数
- *csky_vdsp2_cfft_fast_q15*：Q15 复数 FFT 处理函数快速版本
- *csky_vdsp2_cfft_fast_q31*：Q31 复数 FFT 处理函数快速版本

8.1.2　简要说明

快速傅里叶变换 (FFT) 是一个离散傅里叶变换（DFT）的快速算法。FFT 比 DFT 快了几个数量级，特别是处理较长的序列。这节描述的算法是处理复数数据的，另外有一组函数专门用于处理实数序列。

为 Q15 和 Q31 分别提供了不同的函数接口。

FFT 函数在原地操作。也就是说，保存输入数据的数组也会被用作保存对应的结果。输入数据是复数的，并且包括 2*fftLen 个交错的数据，如下所示：

```
{real[0], imag[0], real[1], imag[1],..}
```

FFT 结果也会包含在相同的数组，并且频域的值也会有一样的交错方式。

Q15 和 Q31

定点复数 FFT 使用了一种混合基算法。算法可以分解为多个 radix-4 步骤和单个 radix-2 步骤。

算法支持长度 [16, 32, 64, ···, 4096]，并且分别为不同的长度提供旋转因子表。其中快速版本重做了旋转因子表，提供了可连续读取旋转因子，加快了整个 FFT 的计算速度。

函数使用了标准 FFT 定义，当计算正向变换的时候，输出值可能会增长 fftLen 。逆变换包括了一个缩放 1/fftLen 作为计算的一部分，这跟教科书中的逆 FFT 定义相符。

使用方式

头文件 csky_vdsp2_const_structs.h 中定义了一些预设的数据结构，数据结构中已经初始化好了旋转因子和位翻转表。可以在源文件中里面包含这个头文件，然后将其作为参数传给 csky_vdsp2_cfft_q31 函数。比如：

```
csky_vdsp2_cfft_q31(&csky_vdsp2_cfft_sR_q31_len64, pSrc, 1, 1)
```

是一次包括了位翻转的 64 点的复数逆 FFT。

其中，数据结构 csky_vdsp2_cfft_sR_q31_len64 被视作常量，不会在计算过程中改变。相同的数据结构可以在多个正逆变换中重复使用。

8.1.3 函数说明

8.1.3.1 csky_vdsp2_cfft_q15

```
void csky_vdsp2_cfft_q15 (const csky_vdsp2_cfft_instance_q15 *S, q15_t *p1, uint8_t ifftFlag,␣
↪uint8_t bitReverseFlag)
```

参数:

*S: 指向 CFFT 结构体实例

*p1: 指向复数数据缓存，大小是 2*fftLen 。原地处理

ifftFlag: 设置是正向 (ifftFlag=0) 还是逆向 (ifftFlag=1) 变换的标志位

bitReverseFlag: 设置输出位翻转 (bitReverseFlag=1) 或者不翻转 (bitReverseFlag=0) 的标志位

返回值:

无

8.1.3.2 csky_vdsp2_cfft_q31

```
void csky_vdsp2_cfft_q31 (const csky_vdsp2_cfft_instance_q31 *S, q31_t *p1, uint8_t ifftFlag,␣
↪uint8_t bitReverseFlag)
```

参数:

*S: 指向 CFFT 结构体实例

*p1: 指向复数数据缓存，大小是 2*fftLen 。原地处理

ifftFlag: 设置是正向 (ifftFlag=0) 还是逆向 (ifftFlag=1) 变换的标志位

bitReverseFlag: 设置输出位翻转 (bitReverseFlag=1) 或者不翻转 (bitReverseFlag=0) 的标志位

返回值:

无

8.1.3.3 csky_vdsp2_cfft_fast_q15

```
void csky_vdsp2_cfft_fast_q15 (const csky_vdsp2_cfft_instance_q15 *S, q15_t *p1, uint8_t ifftFlag,␣
↪uint8_t bitReverseFlag)
```

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

参数:

*S: 指向 CFFT 结构体实例, 其中的旋转因子表为快速版本对应的表格

*p1: 指向复数数据缓存，大小是 2*fftLen 。原地处理

ifftFlag: 设置是正向 (ifftFlag=0) 还是逆向 (ifftFlag=1) 变换的标志位

bitReverseFlag: 设置输出位翻转 (bitReverseFlag=1) 或者不翻转 (bitReverseFlag=0) 的标志位

返回值:

无

8.1.3.4 csky_vdsp2_cfft_fast_q31

```
void csky_vdsp2_cfft_fast_q31 (const csky_vdsp2_cfft_instance_q31 *S, q31_t *p1, uint8_t ifftFlag,␣
↪uint8_t bitReverseFlag)
```

参数:

*S: 指向 CFFT 结构体实例, 其中的旋转因子表为快速版本对应的表格

*p1: 指向复数数据缓存，大小是 2*fftLen 。原地处理

ifftFlag: 设置是正向 (ifftFlag=0) 还是逆向 (ifftFlag=1) 变换的标志位

bitReverseFlag: 设置输出位翻转 (bitReverseFlag=1) 或者不翻转 (bitReverseFlag=0) 的标志位

返回值:

无

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

8.2 实数 FFT 函数

8.2.1 函数

- *csky_vdsp2_rfft_q15*：Q15 实数 FFT 处理函数
- *csky_vdsp2_rfft_q31*：Q31 实数 FFT 处理函数
- *csky_vdsp2_rfft_fast_q15*：Q15 实数 FFT 处理函数快速版本
- *csky_vdsp2_rfft_fast_q31*：Q31 实数 FFT 处理函数快速版本

8.2.2 简要说明

CSI DSP 库为计算实数数据序列的 FFT 实现了特别的算法。FFT 的定义包括了复数数据，但是很多应用的输入只是实数。实数 FFT 算法有对称性的优点，相同长度下比复数算法有速度优势。

快速 RFFT 算法继承自混合基的 CFFT，但是减少了计算量。其中快速版本计算 CFFT 时，使用了对应的复数 FFT 处理函数快速版本。

长度为 N 的正向实数 FFT 序列计算使用的步骤如下：

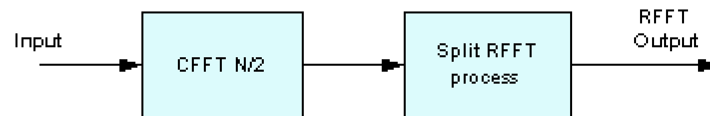

图 8.1: 实数快速傅里叶变换

实数序列初始化的时候，跟 CFFT 的复数处理一样. 之后，一个处理步骤重新整理数据，获取复数频谱的一半。除了第一个复数值包括了两个实数值 X[0] 和 X[N/2]，其他所有的数据都是复数. 换句话说，第一个复数样本包装了两个实数值。

逆 RFFT 的输入应该保持和正向 RFFT 的结果相同的格式。第一步处理步骤为逆 CFFT 预处理数据。其中快速版本计算逆 CFFT 时，使用了对应的复数 FFT 处理函数快速版本。

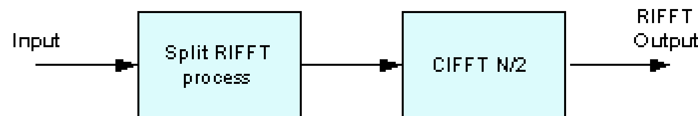

图 8.2: 实数逆向快速傅里叶变换

Q15 和 Q31

主要的函数是 csky_vdsp2_rfft_q31().

一个 N-点序列 FFT 在频域是偶对称的。数据的第二半等于在频率上被翻转的第一半的共轭：

```
*X[0] - real data
*X[1] - complex data
*X[2] - complex data
```

(continues on next page)

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

(续上页)

```
*...
*X[fftLen/2-1] - complex data
*X[fftLen/2] - real data
*X[fftLen/2+1] - conjugate of X[fftLen/2-1]
*X[fftLen/2+2] - conjugate of X[fftLen/2-2]
*...
*X[fftLen-1] - conjugate of X[1]
```

这些数据，我们可以统一的表示 FFT 为

```
*N/2+1 samples:
*X[0] - real data
*X[1] - complex data
*X[2] - complex data
*...
*X[fftLen/2-1] - complex data
*X[fftLen/2] - real data
```

更近一步，第一个和最后一个样本的虚部，可以用零填充。因此，我们可以将 N 点实数序列表示为 (N/2+1) 复数值：

```
*X[0] - complex data, its image part is zero.
*X[1] - complex data
*X[2] - complex data
*...
*X[fftLen/2-1] - complex data
*X[fftLen/2] - complex data, its image part is zero.
```

实数 FFT 函数将频域数据包装成这种方式。正向变换以这种方式输出数据，逆向变换期望接收的输入也是这种方式。函数总是根据需要处理位翻转，所有输入和输出总是正常顺序。函数支持 [32, 64, 128, ···,8192] 长度的样本。

正向和逆向实数 FFT 函数应用标准 FFT; 正向变换不需要缩放，逆向变换缩放 1/fftLen 。

使用方式

头文件 csky_vdsp2_const_structs.h 中定义了一些预设的数据结构，数据结构是已经初始好了的 RFFT 函数第一个参数。可以在源文件中里面包含这个头文件，然后将其作为参数传给 RFFT 函数。

比如：

```
csky_vdsp2_rfft_q31(&csky_vdsp2_rfft_sR_q31_len64, pSrc, pDst)
```

是一次包括了 64 点的 RFFT。

其中，数据结构 csky_vdsp2_rfft_sR_q31_len64 被视作常量，不会在计算过程中改变。相同的数据结构可以在多次变换中重复使用。

数据结构也可以如下手动初始化：

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

```
*csky_vdsp2_rfft_instance_q31 S = {fftLenReal, fftLenBy2, ifftFlagR, bitReverseFlagR,␣
↪twidCoefRModifier, pTwiddleAReal, pCfft};
*csky_vdsp2_rfft_instance_q15 S = {fftLenReal, fftLenBy2, ifftFlagR, bitReverseFlagR,␣
↪twidCoefRModifier, pTwiddleAReal, pCfft};
```

其中 fftLenReal 是实数变换的长度; fftLenBy2 是中间复数变换的长度. ifftFlagR 选择正向 (=0) 或逆向 (=1) 变换. bitReverseFlagR 选择输出位翻转 (=0) 或者输出正常顺序 (=1). twidCoefRModifier 旋转因子表的步幅调节。这个值基于 FFT 长度；pTwiddleAReal 指向旋转系数 A 数组；pCfft 指向 CFFT 结构体实例. CFFT 结构体也必须被初始化.

8.2.3 函数说明

8.2.3.1 csky_vdsp2_rfft_q15

```
void csky_vdsp2_rfft_q15 (const csky_vdsp2_rfft_instance_q15 *S, q15_t *pSrc, q15_t *pDst)
```

参数:

*S: 指向一个 RFFT 结构体

*p: 指向输入数据

*pOut: 指向输出数据

返回值:

无

推荐使用方式:

针对不同的长度，用法如下:

```
csky_vdsp2_rfft_q15(&csky_vdsp2_rfft_sR_q15_len32, pSrc, pDst);
csky_vdsp2_rfft_q15(&csky_vdsp2_rfft_sR_q15_len64, pSrc, pDst);
....
csky_vdsp2_rfft_q15(&csky_vdsp2_rfft_sR_q15_len4096, pSrc, pDst);
csky_vdsp2_rfft_q15(&csky_vdsp2_rfft_sR_q15_len8192, pSrc, pDst);
```

输入和输出格式:

每个步骤的内部输入缩小 2，以防止 CFFT/CIFFT 处理中饱和。因此，不同的 RFFT 大小的输出格式不同。不同大小的 RFFT 输入和输出格式，下表罗列了 RFFT 和 RIFFT 的缩放位数:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

RFFT 大小	输入格式	输出格式	向上缩放的位数
32	1.15	5.11	4
64	1.15	6.10	5
128	1.15	7.9	6
256	1.15	8.8	7
512	1.15	9.7	8
1024	1.15	10.6	9
2048	1.15	11.5	10
4096	1.15	12.4	11
8192	1.15	13.3	12

RIFFT 大小	输入格式	输出格式	向上缩放的位数
32	1.15	5.11	0
64	1.15	6.10	0
128	1.15	7.9	0
256	1.15	8.8	0
512	1.15	9.7	0
1024	1.15	10.6	0
2048	1.15	11.5	0
4096	1.15	12.4	0
8192	1.15	13.3	0

8.2.3.2 csky_vdsp2_rfft_q31

```
void csky_vdsp2_rfft_q31 (const csky_vdsp2_rfft_instance_q31 *S, q31_t *pSrc, q31_t *pDst)
```

参数:

*S: 指向一个 RFFT 结构体

*p: 指向输入数据

*pOut: 指向输出数据

返回值:

无

推荐使用方式:

针对不同的长度，用法如下:

```
csky_vdsp2_rfft_q31(&csky_vdsp2_rfft_sR_q31_len32, pSrc, pDst);
csky_vdsp2_rfft_q31(&csky_vdsp2_rfft_sR_q31_len64, pSrc, pDst);
....
csky_vdsp2_rfft_q31(&csky_vdsp2_rfft_sR_q31_len4096, pSrc, pDst);
csky_vdsp2_rfft_q31(&csky_vdsp2_rfft_sR_q31_len8192, pSrc, pDst);
```

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

输入和输出格式:

每个步骤的内部输入缩小 2，以防止 CFFT/CIFFT 处理中饱和。因此，不同的 RFFT 大小的输出格式不同。不同大小的 RFFT 输入和输出格式，下表罗列了 RFFT 和 RIFFT 的缩放位数:

RFFT 大小	输入格式	输出格式	向上缩放的位数
32	1.31	5.27	4
64	1.31	6.26	5
128	1.31	7.25	6
256	1.31	8.24	7
512	1.31	9.23	8
1024	1.31	10.22	9
2048	1.31	11.21	10
4096	1.31	12.20	11
8192	1.31	13.19	12

RIFFT 大小	输入格式	输出格式	向上缩放的位数
32	1.31	5.27	0
64	1.31	6.26	0
128	1.31	7.25	0
256	1.31	8.24	0
512	1.31	9.23	0
1024	1.31	10.22	0
2048	1.31	11.21	0
4096	1.31	12.20	0
8192	1.31	13.19	0

8.2.3.3 csky_vdsp2_rfft_fast_q15

```
void csky_vdsp2_rfft_fast_q15 (const csky_vdsp2_rfft_fast_instance_q15 *S, q15_t *pSrc, q15_t␣
↪*pDst)
```

参数:

*S: 指向一个 RFFT 结构体

*p: 指向输入数据

*pOut: 指向输出数据

返回值:

无

推荐使用方式:

针对不同的长度，用法如下:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

```
csky_vdsp2_rfft_fast_q15(&csky_vdsp2_rfft_fast_sR_q15_len32, pSrc, pDst);
csky_vdsp2_rfft_fast_q15(&csky_vdsp2_rfft_fast_sR_q15_len64, pSrc, pDst);
....
csky_vdsp2_rfft_fast_q15(&csky_vdsp2_rfft_fast_sR_q15_len4096, pSrc, pDst);
csky_vdsp2_rfft_fast_q15(&csky_vdsp2_rfft_fast_sR_q15_len8192, pSrc, pDst);
```

输入和输出格式:

保护操作及输出格式和普通版本保持一致。

8.2.3.4 csky_vdsp2_rfft_fast_q31

```
void csky_vdsp2_rfft_fast_q31 (const csky_vdsp2_rfft_fast_instance_q31 *S, q31_t *pSrc, q31_t␣
↪*pDst)
```

参数:

*S: 指向一个 RFFT 结构体

*p: 指向输入数据

*pOut: 指向输出数据

返回值:

无

推荐使用方式:

针对不同的长度，用法如下:

```
csky_vdsp2_rfft_fast_q31(&csky_vdsp2_rfft_fast_sR_q31_len32, pSrc, pDst);
csky_vdsp2_rfft_fast_q31(&csky_vdsp2_rfft_fast_sR_q31_len64, pSrc, pDst);
....
csky_vdsp2_rfft_fast_q31(&csky_vdsp2_rfft_fast_sR_q31_len4096, pSrc, pDst);
csky_vdsp2_rfft_fast_q31(&csky_vdsp2_rfft_fast_sR_q31_len8192, pSrc, pDst);
```

输入和输出格式:

保护操作及输出格式和普通版本保持一致。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

8.3 DCT IV 型函数

8.3.1 函数

- *csky_vdsp2_dct4_q15*：Q15 DCT4/IDCT4 处理函数
- *csky_vdsp2_dct4_q31*：Q31 DCT4/IDCT4 处理函数
- *csky_vdsp2_dct4_fast_q15*：Q15 DCT4/IDCT4 处理函数快速版本
- *csky_vdsp2_dct4_fast_q31*：Q31 DCT4/IDCT4 处理函数快速版本

8.3.2 简要说明

离散余弦变换 (DCT) 可以用来构建出能量集中在光谱较低部分的输出。意味着可以用最小的数值来表示信号，在存储和传输中都很有用。因此广泛应用在信号和图片编码程序中。DCT 系列 (DCT 类型- 1,2,3,4) 是均匀边界条件的不同组合结果。DCT 系列有出色的能量包装特性，特别是在数据压缩方面。

DCT 本质上是一个实数信号偶扩展的离散傅里叶变换 (DFT)。输入数据的重新排序可以让计算 DCT 变成计算一个实数信号的 DFT，以及少量的附加操作。因此特定硬件和软件就可以按照 DFT 实现一个简单有效的 DCT 算法。

DCT2 型的内部可以用快速傅里叶变换（FFT）实现，由于变换应用在实数数值，因此可以使用实数 FFT 算法。DCT4 的实现则使用了 DCT2，因为他们的实现很相似，只差了一些预处理和后期处理。DCT2 的实现可以描述为以下步骤：

- 输入重排序
- 计算实数 FFT，快速版本使用了对应的实数 FFT 处理函数快速版本
- 权重乘法和实数 FFT 输出，并从结果中获得实部

综上 DCT4 处理可以用以下框图描述：

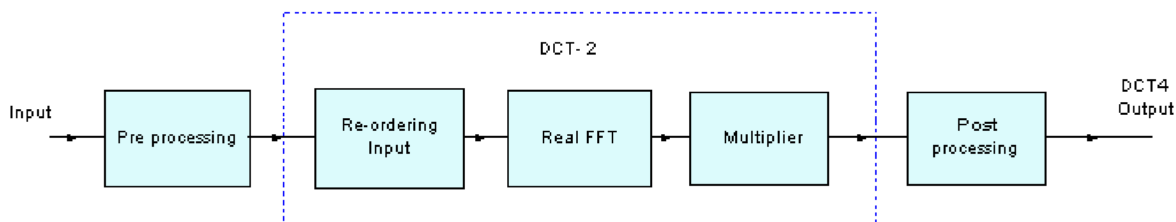

图 8.3: 离散余弦变换 - IV 型

算法:

N-点 IV 型 DCT 公式定义为实数线性转换:

$$X_c(k) = \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} x(n) \cos\left[\left(n+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\frac{\pi}{N}\right]$$

其中 k = 0,1,2,⋯..N-1

它的逆运算定义为:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

$$x(n) = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} X_c(k) \cos\left[\left(n+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\frac{\pi}{N}\right]$$

其中 n = 0,1,2,⋯..N-1

DCT4 矩阵通过乘以 sqrt（2 / N）的总比例因子而变得对合（即它们是自逆的）。变换矩阵的对称性说明用于正向和反向变换计算的快速算法是相同的。注意，DCT4 和逆 DCT4 的实现是相同的，因此，它们可以使用相同的处理函数。

结构体实例:

实数 FFT 和 FFT 的实例，余弦值表和旋转因子表都保存在一个数据结构的实例中。每个转换都必须定义一个单独的数据结构体实例。为支持的 3 种数据类型分别提供了不同的结构体实例。

使用方式:

头文件 csky_vdsp2_const_structs.h 中定义了一些预设的数据结构，数据结构是已经初始好了的 RFFT 函数第一个参数。可以在源文件中里面包含这个头文件，然后将其作为参数传给 RFFT 函数。

比如：

```
csky_vdsp2_dct4_q31(&csky_vdsp2_dct4_sR_q31_len128, pState, pSrc)
```

是一次包括了 128 点的 RFFT。

其中，数据结构 csky_vdsp2_dct4_sR_q31_len128 被视作常量，不会在计算过程中改变。相同的数据结构可以在多次变换中重复使用。

数据结构也可以如下手动初始化：

```
*csky_vdsp2_dct4_instance_q31 S = {N, Nby2, normalize, pTwiddle, pCosFactor, pRfft,␣
↪pCfft};
*csky_vdsp2_dct4_instance_q15 S = {N, Nby2, normalize, pTwiddle, pCosFactor, pRfft,␣
↪pCfft};
```

其中 N 是 DCT4 的长度; Nby2 是 DCT4 长度的一半; normalize 是使用的归一化因子，并且相等于 sqrt(2/N); pTwiddle 指向旋转因子表; pCosFactor 指向余弦因子表; pRfft 指向实数 FFT 实例; pCfft 指向复数 FFT 实例; CFFT 和 RFFT 结构体也需要被初始化.

定点行为:

使用定点 DCT4 转换函数需要注意。特别是要考虑，在每个函数内使用的累加器的溢出和饱和行为。具体参考每个函数各自的文档和使用说明。

8.3.3 函数说明

8.3.3.1 csky__vdsp2__dct4__q15

```
void csky_vdsp2_dct4_q15 (const csky_vdsp2_dct4_instance_q15 *S, q15_t *pState, q15_t␣
↪*pInlineBuffer)
```

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

参数:

`*S`: 指向 DCT/IDCT4 结构体实例

`*pState`: 指向状态数组

`*pInlineBuffer`: 指向输入和输出数据

返回值:

无

推荐使用方式:

针对不同的长度，用法如下:

```
csky_vdsp2_dct4_q15(&csky_vdsp2_dct4_sR_q15_len128, pState, pSrc);
csky_vdsp2_dct4_q15(&csky_vdsp2_dct4_sR_q15_len512, pState, pSrc);
csky_vdsp2_dct4_q15(&csky_vdsp2_dct4_sR_q15_len2048, pState, pSrc);
csky_vdsp2_dct4_q15(&csky_vdsp2_dct4_sR_q15_len8192, pState, pSrc);
```

输入和输出格式:

输入样本需要缩小 1 个位来防止在 Q15 DCT 处理中饱和。RFFT 处理函数的内部输入会被缩放以防止溢出。缩放的位数，依赖于变换的大小。不同 DCT 大小的输入和输出的格式和位的数量，缩放位的数量在下表中:

DCT 大小	输入格式	输出格式	向上缩放的位数
8192	1.15	13.3	12
2048	1.15	11.5	10
512	1.15	9.7	8
128	1.15	7.9	6

8.3.3.2 csky_vdsp2_dct4_q31

```
void csky_vdsp2_dct4_q31 (const csky_vdsp2_dct4_instance_q31 *S, q31_t *pState, q31_t␣
↪*pInlineBuffer)
```

参数:

`*S`: 指向 DCT/IDCT4 结构体实例

`*pState`: 指向状态数组

`*pInlineBuffer`: 指向输入和输出数据

返回值:

无

推荐使用方式:

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

针对不同的长度，用法如下:

```
csky_vdsp2_dct4_q31(&csky_vdsp2_dct4_sR_q31_len128, pState, pSrc);
csky_vdsp2_dct4_q31(&csky_vdsp2_dct4_sR_q31_len512, pState, pSrc);
csky_vdsp2_dct4_q31(&csky_vdsp2_dct4_sR_q31_len2048, pState, pSrc);
csky_vdsp2_dct4_q31(&csky_vdsp2_dct4_sR_q31_len8192, pState, pSrc);
```

输入和输出格式:

输入样本需要缩小 1 个位来防止在 Q31 DCT 处理中饱和。因为从 DCT2 转换到 DCT4 涉及到一个减法。RFFT 处理函数的内部输入会被缩放以防止溢出。缩放的位数，依赖于变换的大小。不同 DCT 大小的输入输出格式和位的数量，缩放位的数量在下表中:

DCT 大小	输入格式	输出格式	向上缩放的位数
8192	2.30	14.18	13
2048	2.30	12.20	11
512	2.30	10.22	9
128	2.30	8.24	7

8.3.3.3 csky_vdsp2_dct4_fast_q15

```
void csky_vdsp2_dct4_fast_q15 (const csky_vdsp2_dct4_fast_instance_q15 *S, q15_t *pState, q15_t
↪*pInlineBuffer)
```

参数:

*S: 指向 DCT/IDCT4 结构体实例

*pState: 指向状态数组

*pInlineBuffer: 指向输入和输出数据

返回值:

无

推荐使用方式:

针对不同的长度，用法如下:

```
csky_vdsp2_dct4_fast_q15(&csky_vdsp2_dct4_fast_sR_q15_len128, pState, pSrc);
csky_vdsp2_dct4_fast_q15(&csky_vdsp2_dct4_fast_sR_q15_len512, pState, pSrc);
csky_vdsp2_dct4_fast_q15(&csky_vdsp2_dct4_fast_sR_q15_len2048, pState, pSrc);
csky_vdsp2_dct4_fast_q15(&csky_vdsp2_dct4_fast_sR_q15_len8192, pState, pSrc);
```

输入和输出格式:

保护操作及输出格式和普通版本保持一致。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

8.3.3.4 csky_vdsp2_dct4_fast_q31

```
void csky_vdsp2_dct4_fast_q31 (const csky_vdsp2_dct4_fast_instance_q31 *S, q31_t *pState, q31_t
↪*pInlineBuffer)
```

参数:

*S: 指向 DCT/IDCT4 结构体实例

*pState: 指向状态数组

*pInlineBuffer: 指向输入和输出数据

返回值:

无

推荐使用方式:

针对不同的长度，用法如下:

```
csky_vdsp2_dct4_fast_q31(&csky_vdsp2_dct4_fast_sR_q31_len128, pState, pSrc);
csky_vdsp2_dct4_fast_q31(&csky_vdsp2_dct4_fast_sR_q31_len512, pState, pSrc);
csky_vdsp2_dct4_fast_q31(&csky_vdsp2_dct4_fast_sR_q31_len2048, pState, pSrc);
csky_vdsp2_dct4_fast_q31(&csky_vdsp2_dct4_fast_sR_q31_len8192, pState, pSrc);
```

输入和输出格式:

保护操作及输出格式和普通版本保持一致。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.