Gowin RiscV_AE350_SOC DSP 软件编程

用户手册

MUG1032-1.1, 2023-12-29

免责声明

本文档并未授予任何知识产权的许可，并未以明示或暗示，或以禁止反言或其它方式授予任何知识产权许可。除高云半导体在其产品的销售条款和条件中声明的责任之外，高云半导体概不承担任何法律或非法律责任。高云半导体对高云半导体产品的销售和／或使用不作任何明示或暗示的担保，包括对产品的特定用途适用性、适销性或对任何专利权、版权或其它知识产权的侵权责任等，均不作担保。高云半导体对文档中包含的文字、图片及其它内容的准确性和完整性不承担任何法律或非法律责任，高云半导体保留修改文档中任何内容的权利，恕不另行通知。高云半导体不承诺对这些文档进行适时的更新。

版本信息

日期	版本	描述
2023/09/12	1.0	初始版本。
2023/12/29	1.1	更新各个函数的参数和返回值的描述。

目录

目录 i

图目录 v

表目录 vi

1 关于本手册 1

1.1 手册内容 1

1.2 术语、缩略语 1

1.3 技术支持与反馈 1

2 概述 2

2.1 Newlib 和 MCUlib 工具链的链接选项 3

2.2 Glibc 工具链的链接选项 3

2.3 DSP 软件编程函数库的数据对齐 4

3 函数接口描述 6

3.1 基本函数 6

3.1.1 绝对值函数 6

3.1.2 求和函数 8

3.1.3 求商函数 10

3.1.4 点积函数 11

3.1.5 乘法函数 14

3.1.6 取反函数 16

3.1.7 偏移函数 18

3.1.8 缩放函数 20

3.1.9 移位函数 23

3.1.10 求差函数 25

3.2 复函数 27

3.2.1 共轭复函数 27

3.2.2 复点积函数 29
3.2.3 复幅值函数 33
3.2.4 复平方函数 34
3.2.5 复数乘法函数 35
3.2.6 复数实数乘法函数 37
3.3 控制器函数 39
3.3.1 Clarke 变换函数 39
3.3.2 Clarke 反变换函数 40
3.3.3 Park 变换函数 41
3.3.4 Park 反变换函数 42
3.3.5. PID 函数 43
3.4 滤波函数 50
3.4.1 双二阶滤波器（直接 1 型）函数 50
3.4.2 双二阶滤波器（转置直接 2 型）函数 57
3.4.3 卷积函数 60
3.4.4 部分卷积函数 62
3.4.5 相关函数 64
3.4.6 FIR 滤波器函数 67
3.4.7 FIR 抽取滤波器函数 72
3.4.8 FIR 格型滤波器函数 77
3.4.9 IIR 格型滤波器函数 80
3.4.10 LMS 滤波器函数 85
3.4.11 NLMS 滤波器函数 89
3.4.12 稀疏 FIR 滤波器函数 93
3.4.13 上采样 FIR 滤波器函数 98
3.5 矩阵函数 101
3.5.1 矩阵加法函数 101
3.5.2 矩阵逆函数 103
3.5.3 矩阵乘法函数 105
3.5.4 矩阵缩放函数 110
3.5.5 矩阵减法函数 112
3.5.6 矩阵转置函数 114

3.5.7 矩阵 2 次幂函数 116
3.5.8 矩阵外积函数 116
3.5.9 复矩阵乘法函数 118
3.6 统计函数 120
3.6.1 最大值函数 120
3.6.2 均值函数 122
3.6.3 最小值函数 124
3.6.4 RMS 函数 126
3.6.5 幂函数 127
3.6.6 标准差函数 129
3.6.7 方差函数 130
3.6.8 熵函数 131
3.6.9 相对熵函数 132
3.6.10 LSE 函数 132
3.6.11 LSE 点积函数 133
3.6.12 朴素高斯贝叶斯预测函数 133
3.7 转换函数 134
3.7.1 Radix-2 CFFT 函数 135
3.7.2 Radix-4 CFFT 函数 139
3.7.3 CFFT 函数 144
3.7.4 DCT Type II 函数 148
3.7.5 DCT Type IV 函数 152
3.7.6 RFFT 函数 156
3.8 工具函数 161
3.8.1 arctan 函数 161
3.8.2 arctan2 函数 162
3.8.3 cos 和 sin 函数 163
3.8.4 转换函数 166
3.8.5 复制函数 171
3.8.6 置位函数 173
3.8.7 平方根函数 174
3.8.8 Barycenter 函数 175

3.8.9 加权和函数 176
3.9 SVM 预测函数 176
3.9.1 SVM 线性预测 176
3.9.2 SVM 径向基（RBF）预测 178
3.9.3 SVM 多项式预测 179
3.9.4. SVM Sigmoid 预测 181
3.10 距离函数 182
3.10.1 浮点型距离 182
3.10.2 布尔型距离 186
3.11 排序函数 190
3.11.1 泛型排序函数 190
3.11.2 归并排序函数 193
4 应用程序 196
4.1 程序描述 196
4.2 应用程序 196
4.3 程序运行 196

图目录

图 3-1 Clarke Transform Diagram 39
图 3-2 Park Transform Diagram 41
图 3-3 State of PID Controller 44

表目录

表 1-1 术语、缩略语 1
表 2-1 链接选项 4
表 3-1 The Input and Output Formats 137
表 3-2 The Input and Output Formats 138
表 3-3 The Input and Output Formats 138
表 3-4 The Input and Output Formats 139
表 3-5 Input and Output Formats 141
表 3-6 Input and Output Formats 142
表 3-7 Input and Output Formats 143
表 3-8 Input and Output Formats 143
表 3-9 Input and Output Formats 146
表 3-10 Input and Output Formats 146
表 3-11 Input and Output Formats 147
表 3-12 Input and Output Formats 148
表 3-13 Input and Output Formats 150
表 3-14 Input and Output Formats 151
表 3-15 Input and Output Formats 151
表 3-16 Input and Output Formats 152
表 3-17 Input and Output Formats 154
表 3-18 Input and Output Formats 155
表 3-19 Input and Output Formats 155
表 3-20 Input and Output Formats 156
表 3-21 Input and Output Formats 158
表 3-22 Input and Output Formats 159
表 3-23 Input and Output Formats 160

表 3-24 Input and Output Formats 160

1 关于本手册

1.1 手册内容

本手册主要描述 Gowin® RiscV_AE350_SOC DSP 软件编程函数库的功能规范，如何使用 DSP 函数接口软件编程，应用程序演示等。

1.2 术语、缩略语

本手册中的相关术语、缩略语及相关释义，如表 1-1 所示。

表 1-1 术语、缩略语

术语、缩略语	全称	含义
CSR	Control and Status Register	控制和状态寄存器
DSP	Digital Signal Processor	数字信号处理器
ISA	Instruction Set Architecture	指令集架构
RISC-V	Reduced Instruction Set Computer V	第五代精简指令集计算机
SOC	System on Chip	片上系统

1.3 技术支持与反馈

高云®半导体提供全方位技术支持，在使用过程中如有疑问或建议，可直接与公司联系：

网址：www.gowinsemi.com.cn

E-mail：support@gowinsemi.com

Tel：+86 755 8262 0391

2 概述

Gowin RiscV_AE350_SOC DSP 软件编程函数库，提供全面的函数接口，代码高效精简，方便用户简便快捷地开发数字信号处理系统。这些函数接口包括基础向量、矩阵和复向量等基本运算，也包括滤波和函数变换等复杂算法运算。DSP 软件编程函数库将复杂的信号处理封装为简单易用的应用函数接口，使用 RDS 软件工具链进行设计与开发，有利于用户节省开发资源，使用户更多得专注于系统设计，降低开发时间。

DSP 软件编程函数库的函数接口可以分为以下几类：基本函数、复函数、控制器函数、滤波函数、矩阵函数、统计函数、变换函数、工具函数、SVM 函数、距离函数和排序函数，第 3 章函数接口描述将详细介绍每个函数接口。

当调用 DSP 函数接口时，只需包含 DSP 软件编程函数库的头文件，这些头文件定义了函数接口的原型和实例化结构体，以功能类别作为命名方式，如下所示：

- riscv_dsp_basic_math.h
- riscv_dsp_complex_math.h
- riscv_dsp_controller_math.h
- riscv_dsp_filtering_math.h
- riscv_dsp_matrix_math.h
- riscv_dsp_statistics_math.h
- riscv_dsp_transform_math.h
- riscv_dsp_utils_math.h
- riscv_dsp_svm_math.h
- riscv_dsp_distance_math.h
- riscv_dsp_sort_math.h

例如如果需要使用基本函数对向量进行某些操作，则可以在 C 文件中包含头文件 riscv_dsp_basic_math.h。

注！

为获得更好的性能，控制器函数直接在头文件 riscv_dsp_controller_math.h 中实现。如果

需要使用 DSP ISA 优化这些函数接口，则必须使用编译器标志-mext-dsp 来开启工具链相关功能。

DSP 软件编程函数库定义了几种数据类型来保存信号数据，以便于后续处理。这些数据类型定义于 riscv_dsp_math_types.h，如下所示，q7_t 用于 8 位整数，q15_t 用于 16 位整数，q31_t 用于 32 位整数，q63_t 用于 64 位整数，float32_t 用于 32 位单精度浮点，float64_t 用于 64 位双精度浮点，如下所示：

```
typedef int8_t q7_t;            /* q7 type */
typedef int16_t q15_t;          /* q15 type */
typedef int32_t q31_t;          /* q31 type */
typedef int64_t q63_t;          /* q63 type */
typedef float float32_t;        /* single-precision floating-point type */
typedef double float64_t;       /* double-precision floating-point type */
```

2.1 Newlib 和 MCUlib 工具链的链接选项

除上述的头文件外，不同的工具链关联不同的链接选项，本节详细介绍用于 Newlib 和 MCUlib 的选项包括：

- -ldsp 链接 DSP 软件编程函数库（libdsp.a）
- -mext-dsp 开启 DSP ISA 扩展
- -lm 链接 libm math 函数库

如果目标系统不支持 DSP ISA 扩展来加速 DSP 软件编程函数库，则使用-ldsp 构建一个链接 DSP 软件编程函数库（libdsp.a）的应用程序。如果目标系统支持 DSP ISA 扩展，则使用-ldsp -mext-dsp 选项开启扩展，以达到更好的性能。

注！

使用-ldsp -mext-dsp 选项构建的应用程序，不能在不支持 DSP ISA 扩展的系统上运行，否则加载时会出现问题。

如果应用程序需要使用包含 libm math 函数库函数接口的 DSP 软件编程函数接口（即，riscv_dsp_std_f32，riscv_dsp_rms_f32，riscv_dsp_cmag_f32，riscv_dsp_sqrt_f32 和 riscv_dsp_atan_f32），则使用-lm 选项。

2.2 Glibc 工具链的链接选项

对于 glibc 工具链可用的链接选项包括：

- -static 使用静态链接
- -ldsp 链接无 DSP ISA 扩展的 DSP 软件编程函数库

-ldsp_p 链接具有 DSP ISA 扩展的 DSP 软件编程函数库（为向后兼容，之前的-ldsp_hw 选项，作为一个链接指向-ldsp_p）

- -lm 链接 libm math 函数库

静态链接和动态链接需要不同的选项，表 2-1 总结了不同链接类型以及是否需要 DSP ISA 扩展支持的链接选项。

表 2-1 链接选项

链接类型	DSP ISA 扩展	链接选项
Static Linking	Without	-static -ldsp
	With	-static -ldsp_p
Dynamic Linking (default)	Without	-ldsp
	With	-ldsp_p

所需选项中，具有 DSP ISA 扩展（即-static -ldsp_p 和-ldsp_p）的选项，用于支持扩展的目标系统构建应用程序，而其他选项（例如，-static -ldsp 和-ldsp），仅用于不支持扩展的目标系统。请用户确保指定的选项对应于所用的目标系统。

如果应用程序是静态链接，并且需要使用包含 libm math 函数库函数接口的 DSP 软件编程函数接口（即 riscv_dsp_std_f32，riscv_dsp_rms_f32，riscv_dsp_cmag_f32，riscv_dsp_sqrt_f32 和 riscv_dsp_atan_f32），则确保使用-lm 选项。

2.3 DSP 软件编程函数库的数据对齐

DSP 软件编程函数库是为 32 位 RiscV_AE350_SOC MCU 平台而设计，使用 4 字节数据对齐方式，即假设输入数组（包括输入源数组和输出目的数组）的首地址，是与 4 字节对齐的。不管输入指针的类型，这些函数接口实现加载/保存一个字，然后使用 SIMD DSP 指令处理，以提高性能。从非对齐的内存地址访问一个字，称为非对齐的内存访问，在不支持非对齐访问的 MCU 会导致地址非对齐异常。对于支持非对齐访问的 MCU，非对齐的内存访问不会导致异常，但会导致系统性能下降。RiscV_AE350_SOC MCU 如何处理非对齐访问的详细描述，请参考 RiscV_AE350_SOC 架构和 CSR 定义。

DSP 软件编程函数库支持的数据类型中，Q31 和 F32 都是 4 字节数据类型，因此都以 4 字节地址对齐方式声明。Q15 和 Q7 分别是 2 字节和 1 字节的数据类型，默认情况下分别以 2 字节和 1 字节地址对齐。为避免 32 位 RiscV_AE350_SOC MCU 内存访问不一致，需要显式声明 Q15 和 Q7 的输入数组是在 4 字节对齐的地址上访问的，如下所示：

```
q7_t inputA[blocksize] __attribute__ ((aligned(4)));
q15_t outputvec[blocksize] __attribute__ ((aligned(4)));
```

以下函数，不需要显式声明对齐访问：

- 控制器函数
- 转换函数
- Arc tangent 函数
- Arc tangent 2 函数
- Cosine 函数
- Sine 函数
- 平方根函数

以下函数，也不需要显式声明对齐访问：

- 使用 F32 数据类型输入数组的函数
- 使用 Q31 数据类型输入数组的函数

3 函数接口描述

以下各节按功能分类，分别详细描述 DSP 软件编程函数库的函数接口。

3.1 基本函数

3.1.1 绝对值函数

绝对值函数，用于从源向量中获取元素进行取绝对值，结果依次写入目的向量，实现过程如下所示：

```
for (n = 0; n < size; n++)
  dst[n] = (src[n] < 0) ? -src[n] : src[n];
```

DSP 软件编程函数库支持以下不同数据类型的绝对值函数：浮点型、Q31、Q15 和 Q7，以下各节详细描述各个绝对值函数。

riscv_dsp_abs_f32

原型：

```
void riscv_dsp_abs_f32 (float32_t *src, float32_t *dst, uint32_t size)
```

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

riscv_dsp_abs_q31

原型：

```
void riscv_dsp_abs_q31 (q31_t *src, q31_t *dst, uint32_t size)
```

参数：

- [in] *src 指向输入向量的指针

- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 输出结果的范围为[0, 0x7FFFFFFF]；
- 如果输入值为 INT32_MIN (0x80000000)，则输出值为 INT32_MAX (0x7FFFFFFF)。

riscv_dsp_abs_q15

原型：

void riscv_dsp_abs_q15 (q15_t *src, q15_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 输出结果的范围为[0, 0x7FFF]；
- 如果输入值为 INT16_MIN (0x8000)，则输出值为 INT16_MAX (0x7FFF)。

riscv_dsp_abs_q7

原型：

void riscv_dsp_abs_q7 (q7_t *src, q7_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 输出结果的范围为[0, 0x7F]；
- 如果输入值为 INT8_MIN (0x80)，则输出值为 INT8_MAX (0x7F)。

3.1.2 求和函数

求和函数，分别从两个不同的源向量中获取两个元素进行相加，结果依次写入目的向量，实现过程如下所示：

```
for (n = 0; n < size; n++)
  dst[n] = src1[n] + src2[n];
```

DSP 软件编程函数库支持以下不同数据类型的求和函数：浮点型、Q31、Q15、Q7 和其他数据类型，以下各节详细描述各个求和函数。

riscv_dsp_add_f32

原型：

```
void riscv_dsp_add_f32 (float32_t *src1, float32_t *src2, float32_t *dst, uint32_t size)
```

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [out] *dst 指向目的向量的指针
- [in] size 向量的元素数量

返回值：

无。

riscv_dsp_add_q31

原型：

```
void riscv_dsp_add_q31 (q31_t *src1, q31_t *src2, q31_t *dst, uint32_t size)
```

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [out] *dst 指向目的向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

求和结果饱和为 Q31 类型的范围[0x80000000, 0x7FFFFFFF]。

riscv_dsp_add_q15

原型：

void riscv_dsp_add_q15 (q15_t *src1, q15_t *src2, q15_t *dst, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [out] *dst 指向目的向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

求和结果饱和为 Q15 类型的范围[0x8000, 0x7FFF]。

riscv_dsp_add_q7

原型：

void riscv_dsp_add_q7 (q7_t *src1, q7_t *src2, q7_t *dst, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [out] *dst 指向目的向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

求和结果饱和为 Q7 类型的范围[0x80, 0x7F]。

riscv_dsp_add_u8_u16

原型：

void riscv_dsp_add_u8_u16(uint8_t *src1, uint8_t *src2, uint16_t *dst, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针

- [out] *dst 指向目的向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

两个 uint8_t 数据的求和结果保存为 uint16_t 类型。

3.1.3 求商函数

求商函数，分别从两个不同的向量/标量中获取两个元素进行相除，结果依次写入目的向量/标量，实现过程如下所示：

```
for (n = 0; n < size; n++)
  dst[n] = src1[n] / src2[n];
```

DSP 软件编程函数库支持以下不同数据类型的求商函数：浮点型、Q31 和其他数据类型，以下各节详细描述各个求商函数。

riscv_dsp_div_f32

原型：

void riscv_dsp_div_f32 (float32_t *src1, float32_t *src2, float32_t *dst, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针（被除数向量）
- [in] *src2 指向第二个输入向量的指针（除数向量）
- [out] *dst 指向目的向量的指针（商数向量）
- [in] size 向量的元素数量

返回值：

无。

riscv_dsp_div_q31

原型：

q31_t riscv_dsp_div_q31 (q31_t src1, q31_t src2)

参数：

- [in] src1 被除数的值
- [in] src2 除数的值

返回值：

商数的值。

注！

- src1 的绝对值应小于 src2 的绝对值；
- 此函数处理的是标量，而不是向量。

riscv_dsp_div_u64_u32

原型：

q31_t riscv_dsp_div_u64_u32 (uint64_t src1, uint32_t src2)

参数：

- [in] src1 被除数的值
- [in] src2 除数的值

返回值：

商数的值。

注！

- src1 右移 32 位的值应小于 src2 右移 32 位的值，否则结果无效；
- 此函数处理的是标量，而不是向量。

riscv_dsp_div_s64_u32

原型：

q31_t riscv_dsp_div_s64_u32 (q63_t src1, uint32_t src2)

参数：

- [in] src1 被除数的值
- [in] src2 除数的值

返回值：

商数的值。

注！

- src1 右移 31 位的绝对值应小于 src2 右移 31 位的绝对值；
- 此函数处理的是标量，而不是向量。

3.1.4 点积函数

点积函数用于计算两个向量的点积，返回结果值，实现过程如下所示：

```
for (output = 0, n = 0; n < size; n++)
  output += src1[n] * src2[n];
```

DSP 软件编程函数库支持以下不同数据类型的点积函数：浮点型、Q31、Q15、Q7 和其他数据类型，以下各节详细描述各个点积函数。

riscv_dsp_dprod_f32

原型：

float32_t riscv_dsp_dprod_f32 (float32_t *src1, float32_t *src2, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] size 向量的元素数量

返回值：

两个输入向量的点积。

riscv_dsp_dprod_q31

原型：

q63_t riscv_dsp_dprod_q31 (q31_t *src1, q31_t *src2, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] size 向量的元素数量

返回值：

两个输入向量的点积。

注！

返回值的格式为 Q48，类型为 q63_t。

riscv_dsp_dprod_q15

原型：

q63_t riscv_dsp_dprod_q15 (q15_t *src1, q15_t *src2, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] size 向量的元素数量

返回值：

两个输入向量的点积。

注！

返回值的格式为 Q30，类型为 q63_t。

riscv_dsp_dprod_q7

原型：

q31_t riscv_dsp_dprod_q7 (q7_t *src1, q7_t *src2, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] size 向量的元素数量

返回值：

两个输入向量的点积。

注！

返回值的格式为 Q14，类型为 q31_t。

riscv_dsp_dprod_u8

原型：

uint32_t riscv_dsp_dprod_u8 (uint8_t *src1, uint8_t *src2, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] size 向量的元素数量

返回值：

两个输入向量的点积。

注！

此函数，用于相乘一个 uint8_t 数据和另一个 uint8_t 数据，然后累加结果到一个 uint32_t 累加器。

riscv_dsp_dprod_u8xq15

原型：

q31_t riscv_dsp_dprod_u8xq15 (uint8_t *src1, q15_t *src2, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] size 向量的元素数量

返回值：

两个输入向量的点积。

注！

此函数，用于相乘一个 uint8_t 数据和另一个 q15_t 数据，然后累加结果到一个 q31_t 累加器。

riscv_dsp_dprod_q7xq15

原型：

q31_t riscv_dsp_dprod_q7xq15 (q7_t *src1, q15_t *src2, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] size 向量的元素数量

返回值：

两个输入向量的点积。

注！

此函数用于相乘一个 q7_t 数据和另一个 q15_t 数据，然后累加结果到一个 q31_t 累加器。

3.1.5 乘法函数

乘法函数，分别从两个不同的源向量获取两个元素进行相乘，结果依次写入目的向量，实现过程如下所示：

```
for (n = 0; n < size; n++)
  dst[n] = src1[n] * src2[n];
```

DSP 软件编程函数库支持以下不同数据类型的乘法函数：浮点型、Q31、Q15、Q7 和其他数据类型，以下各节详细描述各个乘法函数。

riscv_dsp_mul_f32

原型：

void riscv_dsp_mul_f32 (float32_t *src1, float32_t *src2, float32_t *dst, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [out] *dst 指向目的向量的指针
- [in] size 向量的元素数量

返回值：

无。

riscv_dsp_mul_q31

原型：

void riscv_dsp_mul_q31 (q31_t *src1, q31_t *src2, q31_t *dst, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [out] *dst 指向目的向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

乘法结果饱和为 Q31 类型的范围[0x80000000, 0x7FFFFFFF]。

riscv_dsp_mul_q15

原型：

void riscv_dsp_mul_q15 (q15_t *src1, q15_t *src2, q15_t *dst, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [out] *dst 指向目的向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

乘法结果饱和为 Q15 类型的范围[0x8000, 0x7FFF]。

riscv_dsp_mul_q7

原型：

void riscv_dsp_mul_q7 (q7_t *src1, q7_t *src2, q7_t *dst, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针

- [in] *src2 指向第二个输入向量的指针
- [out] *dst 指向目的向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

乘法结果饱和为 Q7 类型的范围[0x80, 0x7F]。

riscv_dsp_mul_u8_u16

原型：

void riscv_dsp_mul_u8_u16 (uint8_t *src1, uint8_t *src2, uint16_t *dst, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [out] *dst 指向目的向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

乘法结果保存为 uint16_t *dst。

3.1.6 取反函数

取反函数，用于对源向量中的元素求反，结果依次写入目的向量，实现过程如下所示：

```
for (n = 0; n < size; n++)
  dst[n] = -src[n];
```

DSP 软件编程函数库支持以下不同数据类型的取反函数：浮点型、Q31、Q15 和 Q7，以下各节详细描述各个取反函数。

riscv_dsp_neg_f32

原型：

void riscv_dsp_neg_f32 (float32_t *src, float32_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针

- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

riscv_dsp_neg_q31

原型：

void riscv_dsp_neg_q31 (q31_t *src, q31_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 输出结果的范围为[0x80000001, 0x7FFFFFFF]；
- 如果输入值为 INT32_MIN (0x80000000)，则输出值为 INT32_MAX (0x7FFFFFFF)。

riscv_dsp_neg_q15

原型：

void riscv_dsp_neg_q15 (q15_t *src, q15_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 输出结果的范围为[0x8001, 0x7FFF]；
- 如果输入值为 INT16_MIN (0x8000)，则输出值为 INT16_MAX (0x7FFF)。

riscv_dsp_neg_q7

原型：

void riscv_dsp_neg_q7 (q7_t *src, q7_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针

- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 输出结果的范围为[0x81, 0x7F]；
- 如果输入值为 INT8_MIN (0x80)，则输出值为 INT8_MAX (0x7F)。

3.1.7 偏移函数

偏移函数，用于将源向量中的每个元素偏移一个常数，结果依次写入目的向量，实现过程如下所示：

```
for (n = 0; n < size; n++)
  dst[n] = src[n] + offset;
```

DSP 软件编程函数库支持以下不同数据类型的偏移函数：浮点型、Q31、Q15、Q7 和其他数据类型，以下各节详细描述各个偏移函数。

riscv_dsp_offset_f32

原型：

void riscv_dsp_offset_f32 (float32_t *src, float32_t offset, float32_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] offset 恒定的偏移值
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

riscv_dsp_offset_q31

原型：

void riscv_dsp_offset_q31 (q31_t *src, q31_t offset, q31_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] offset 恒定的偏移值
- [out] *dst 指向输出向量的指针

- [in] size 向量的元素数量

返回值：

无。

注！

结果值饱和为 Q31 类型的范围[0x80000000, 0x7FFFFFFF]。

riscv_dsp_offset_q15

原型：

void riscv_dsp_offset_q15 (q15_t *src, q15_t offset, q15_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] offset 恒定的偏移值
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

结果值饱和为 Q15 类型的范围[0x8000, 0x7FFF]。

riscv_dsp_offset_q7

原型：

void riscv_dsp_offset_q7 (q7_t *src, q7_t offset, q7_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] offset 恒定的偏移值
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

结果值饱和为 Q7 类型的范围[0x80, 0x7F]。

riscv_dsp_offset_u8

原型：

void riscv_dsp_offset_u8 (uint8_t *src, q7_t offset, uint8_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] offset 恒定的偏移值
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

结果值饱和为 uint8_t 类型的范围[0x00, 0xFF]。

3.1.8 缩放函数

缩放函数，用于将源向量中的每个元素乘以一个常数缩放值，结果依次写入目的向量，实现过程如下所示：

```
for (n = 0; n < size; n++)
  dst[n] = src[n] * scaling_value;
```

在用分数表示的情况下，引入参数 shift 来调整结果值的范围，关于该参数的详细介绍，请参考分数数据类型函数（第 3.1.8.2 ~ 3.1.8.5 节）。

DSP 软件编程函数库支持以下不同数据类型的缩放函数：浮点型、Q31、Q15、Q7 和其他数据类型，以下各节详细描述各个缩放函数。

riscv_dsp_scale_f32

原型：

void riscv_dsp_scale_f32 (float32_t *src, float32_t scale, float32_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] scale 恒定的缩放值
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

示例：

```
#define size 2
```

```
float32_t src[size] = {0.8167, 0.5};
float32_t dst[size] = {0};
float32_t scale = 0.4;
riscv_dsp_scale_f32(src, scale, dst, size);
```

本示例同样适用于 Q31、Q15 或 Q7 类型的缩放函数。

riscv_dsp_scale_q31

原型：

```
void riscv_dsp_scale_q31 (q31_t *src, q31_t scalefract, int8_t shift, q31_t *dst, uint32_t size)
```

参数：

- [in] *src 指向输入向量的指针
- [in] scalefract 恒定的分数缩放值
- [in] shift 用于结果调整的位（参照以下注释说明）
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 由于分数表示，结果应在 Q31 类型范围内，可以使用输入参数 shift 来调整结果值，此函数中 shift 的范围为 0~31；
- 此函数大致的实现过程，如下所示：

```
dst[n] = (src[n] * scalefract) >> (31 - shift)
```

Where 0 <= n < size and 0 <= shift <= 31.

riscv_dsp_scale_q15

原型：

```
void riscv_dsp_scale_q15 (q15_t *src, q15_t scalefract, int8_t shift, q15_t *dst, uint32_t size)
```

参数：

- [in] *src 指向输入向量的指针
- [in] scalefract 恒定的分数缩放值
- [in] shift 用于结果调整的位（参照以下注释说明）
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 由于分数表示，结果应在 Q15 类型范围内，可以使用输入参数 shift 来调整结果值，此函数中 shift 的范围为 0~15；
- 此函数大致的实现过程，如下所示：

dst[n] = (src[n] * scalefract) >> (15 - shift)

Where 0 <= n < size and 0 <= shift <= 15.

riscv_dsp_scale_q7

原型：

void riscv_dsp_scale_q7 (q7_t *src, q7_t scalefract, int8_t shift, q7_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] scalefract 恒定的分数缩放值
- [in] shift 用于结果调整的位（参照以下注释说明）
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 由于分数表示，结果应在 Q7 类型范围内，可以使用输入参数 shift 来调整结果值，此函数中 shift 的范围为 0~7；
- 此函数大致的实现过程，如下所示：

dst[n] = (src[n] * scalefract) >> (7 - shift)

Where 0 <= n < size and 0 <= shift <= 7.

riscv_dsp_scale_u8

原型：

void riscv_dsp_scale_u8 (uint8_t *src, q7_t scalefract, int8_t shift, uint8_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] scalefract 恒定的分数缩放值
- [in] shift 用于结果调整的位（参照以下注释说明）

- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 由于目的向量的数据类型为 uint8_t，所以缩放结果将饱和为 uint8_t 类型的范围[0x00, 0xFF]，可以使用输入参数 shift 来调整结果值，此函数中 shift 的范围为 0~7；
- 此函数大致的实现过程，如下所示：

dst[n] = (src[n] * scalefract) >> (7 - shift)

Where 0 <= n < size and 0 <= shift <= 7.

3.1.9 移位函数

移位函数，用于将源向量中的每个元素以一个常数进行移位，结果依次写入目的向量，正/负移位的值决定元素移位方向左/右，实现过程如下所示：

```
for (n = 0; n < size; n++)
{
  if (shift < 0)
    dst[n] = src[n] >> (-shift);
  else
    dst[n] = src[n] << (shift);
}
```

DSP 软件编程函数库支持以下不同数据类型的移位函数：浮点型、Q31、Q15、Q7 和其他数据类型，以下各节详细描述各个移位函数。

riscv_dsp_shift_q31

原型：

void riscv_dsp_shift_q31 (q31_t *src, int8_t shift, q31_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] shift 有符号的移位值
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 考虑到性能，移位的值须不大于 32，否则结果可能不正确；
- 结果值饱和为 Q31 类型的范围[0x80000000, 0x7FFFFFFF]。

示例：

```
#define size 1024
q31_t src[size] = {…};
q31_t dst[size];
q31_t shift = 1;
riscv_dsp_shift_q31(src, shift, dst, size);
```

本示例同样适用于 Q15 或 Q7 类型的移位函数。

riscv_dsp_shift_q15

原型：

```
void riscv_dsp_shift_q15 (q15_t *src, int8_t shift, q15_t *dst, uint32_t size)
```

参数：

- [in] *src 指向输入向量的指针
- [in] shift 有符号的移位值
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 考虑到性能，移位的值须不大于 16，否则结果可能不正确；
- 结果值饱和为 Q15 类型的范围[0x8000, 0x7FFF]。

riscv_dsp_shift_q7

原型：

```
void riscv_dsp_shift_q7 (q7_t *src, int8_t shift, q7_t *dst, uint32_t size)
```

参数：

- [in] *src 指向输入向量的指针
- [in] shift 有符号的移位值
- [out] *dst 指向输出向量的指针

- [in] size 向量的元素数量

返回值：

无。

注！

- 考虑到性能，移位的值须不大于 24，否则结果可能不正确；
- 结果值饱和为 Q7 类型的范围[0x80, 0x7F]。

riscv_dsp_shift_u8

原型：

void riscv_dsp_shift_u8 (uint8_t *src, int8_t shift, uint8_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] shift 有符号的移位值
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 考虑到性能，移位的值须不大于 24，否则结果可能不正确；
- 结果值饱和为 uint8_t 类型的范围[0x00, 0xFF]。

3.1.10 求差函数

求差函数，分别从两个不同的源向量中获取两个元素进行相减，结果依次写入目的向量，实现过程如下所示：

```
for (n = 0; n < size; n++)
  dst[n] = src1[n] - src2[n];
```

DSP 软件编程函数库支持以下不同数据类型的求差函数：浮点型、Q31、Q15、Q7 和其他数据类型，以下各节详细描述各个求差函数。

riscv_dsp_sub_f32

原型：

void riscv_dsp_sub_f32 (float32_t *src1, float32_t *src2, float32_t *dst, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针

- [in] *src2 指向第二个输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

riscv_dsp_sub_q31

原型：

void riscv_dsp_sub_q31 (q31_t *src1, q31_t *src2, q31_t *dst, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

结果饱和为 Q31 类型的范围[0x80000000, 0x7FFFFFFF]。

riscv_dsp_sub_q15

原型：

void riscv_dsp_sub_q15 (q15_t *src1, q15_t *src2, q15_t *dst, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

结果饱和为 Q15 类型的范围[0x8000, 0x7FFF]。

riscv_dsp_sub_q7

原型：

void riscv_dsp_sub_q7 (q7_t *src1, q7_t *src2, q7_t *dst, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

结果饱和为 Q7 类型的范围[0x80, 0x7F]。

riscv_dsp_sub_u8_q7

原型：

void riscv_dsp_sub_u8_q7 (uint8_t *src1, uint8_t *src2, q7_t *dst, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

结果饱和为 Q7 类型的范围[0x80, 0x7F]。

3.2 复函数

复函数，用于为复向量提供计算。DSP 软件编程函数库中，复向量元素应按实数部与虚数部交错排列，即内存布局看起来为[实，虚，实，虚，…]。

3.2.1 共轭复函数

共轭复函数，用于计算源向量复数的共轭数，结果写入目的向量，实现过程如下所示：

for (n = 0; n < size; n++)

```
dst[2 * n] = src[2 * n];
dst[2 * n + 1] = -src[2 * n + 1];
```

DSP 软件编程函数库支持以下不同数据类型的共轭复函数：浮点型、Q31 和 Q15，以下各节详细描述各个共轭复函数。

riscv_dsp_cconj_f32

原型：

```
void riscv_dsp_cconj_f32 (const float32_t *src, float32_t *dst, uint32_t size)
```

参数：

- [in] *src 指向输入复向量的指针
- [out] *dst 指向输出复向量的指针
- [in] size 向量的复元素数量

返回值：

无。

示例：

一组包含 3 个复元素的复向量{3 + 4i, 4 - 7i, -3 + 5i}，复共轭如下所示：

```
//Complex vector is arranged with real and imaginary parts interleaved.
#define size 3
float32_t src[2 * size] = {3.0, 4.0, 4.0, -7.0, -3.0, 5.0};
float32_t dst[2 * size];
riscv_dsp_cconj_f32(src, dst, size);
```

本示例同样适用于 Q31 或 Q15 类型的共轭复函数。

riscv_dsp_cconj_q31

原型：

```
void riscv_dsp_cconj_q31 (const q31_t *src, q31_t *dst, uint32_t size)
```

参数：

- [in] *src 指向输入复向量的指针
- [out] *dst 指向输出复向量的指针
- [in] size 向量的复元素数量

返回值：

无。

注！

- 复向量的实部和虚部都在 Q31 类型的范围；
- 如果值为 INT32_MIN (0x80000000)的输入虚部取负，则复共轭后输出虚部为 INT32_MAX (0x7FFFFFFF)。

riscv_dsp_cconj_q15

原型：

void riscv_dsp_cconj_q15 (const q15_t *src, q15_t *dst, uint32_t size)

参数：

- [in] *src 指向输入复向量的指针
- [out] *dst 指向输出复向量的指针
- [in] size 向量的复元素数量

返回值：

无。

注！

- 复向量的实部和虚部都在 Q15 类型的范围；
- 如果值为 INT16_MIN (0x8000)的输入虚部取负，则复共轭后输出虚部为 INT16_MAX (0x7FFF)。

3.2.2 复点积函数

复点积函数，用于计算两个复向量的点积，结果写入目标向量。

DSP 软件编程函数库包括两种复点积函数：

第一种复点积函数，包括 riscv_dsp_cdprod_f32、riscv_dsp_cdprod_q31 和 riscv_dsp_cdprod_q15，实现过程如下所示：

```
for (n = 0; n < size; n++)
  a = src1[2 * n];
  b = src1[2 * n + 1];
  c = src2[2 * n];
  d = src2[2 * n + 1];
  dst[2 * n] = a * c + b * d;
  dst[2 * n + 1] = a * d - b * c;
```

第二种复点积函数，称为复点积类型 2 函数，包括 riscv_dsp_cdprod_typ2_f32、riscv_dsp_cdprod_typ2_q31 和 riscv_dsp_cdprod_typ2_q15，实现过程如下所示：

```
for (n = 0; n < size; n++)
```

```
a = src1[2 * n];
b = src1[2 * n + 1];
c = src2[2 * n];
d = src2[2 * n + 1];
real_sum += a * c - b * d;
image_sum += a * d + b * c;
```

以下各节详细描述各个复点积函数。

riscv_dsp_cdprod_f32

原型：

```
void riscv_dsp_cdprod_f32 (const float32_t *src1, const float32_t *src2, uint32_t size, float32_t *dst)
```

参数：

- [in] *src1 指向第一个输入复向量的指针
- [in] *src2 指向第二个输入复向量的指针
- [in] size 向量的复元素数量
- [out] *dst 指向输出复向量的指针

返回值：

无。

示例：

两组复向量，每组包含 3 个复元素{3 + 4i, 4 - 7i, -3 + 5i}和{1 - 2i, 5 - 1i, -4 + 3i}， 复点积运算如下所示：

```
#define size 3
float32_t src1[2*size] = {3, 4, 4, -7, -3, 5};
float32_t src2[2*size] = {1, -2, 5, -1, -4, 3};
float32_t dst[2*size];
riscv_dsp_cdprod_f32(src1, src2, size, dst);
```

本示例同样适用于 Q31 或 Q15 类型的复点积函数。

riscv_dsp_cdprod_q31

原型：

```
void riscv_dsp_cdprod_q31 (const q31_t *src1, const q31_t *src2, uint32_t size, q31_t *dst)
```

参数：

- [in] *src1 指向第一个输入复向量的指针

- [in] *src2 指向第二个输入复向量的指针
- [in] size 向量的复元素数量
- [out] *dst 指向输出复向量的指针

返回值：

无。

注！

值写入 Q29 类型的目的向量。

riscv_dsp_cdprod_q15

原型：

void riscv_dsp_cdprod_q15 (const q15_t *src1, const q15_t *src2, uint32_t size, q15_t *dst)

参数：

- [in] *src1 指向第一个输入复向量的指针
- [in] *src2 指向第二个输入复向量的指针
- [in] size 向量的复元素数量
- [out] *dst 指向输出复向量的指针

返回值：

无。

注！

值写入 Q13 类型的目的向量。

riscv_dsp_cdprod_typ2_f32

原型：

void riscv_dsp_cdprod_typ2_f32 (const float32_t *src1, const float32_t *src2, uint32_t size, float32_t *rout, float32_t *iout)

参数：

- [in] *src1 指向第一个输入复向量的指针
- [in] *src2 指向第二个输入复向量的指针
- [in] size 向量的复元素数量
- [out] *rout 指向实数输出的指针
- [out] *iout 指向像点输出的指针

返回值：

无。

示例：

两组复向量，每组包含 3 个复元素{3 + 4i, 4 - 7i, -3 + 5i}和{1 - 2i, 5 - 1i, -4 + 3i}，复点积运算如下所示：

```
#define size 3
float32_t src1[2*size] = {3, 4, 4, -7, -3, 5};
float32_t src2[2*size] = {1, -2, 5, -1, -4, 3};
float32_t real_out, image_out;
riscv_dsp_cdprod_typ2_f32(src1, src2, size, &real_out, &image_out);
```

本示例同样适用于 Q31 或 Q15 类型的复点积类型 2 函数。

riscv_dsp_cdprod_typ2_q31

原型：

```
void riscv_dsp_cprod_typ2_q31 (const q31_t *src1, const q31_t *src2, uint32_t size, q63_t *rout, q63_t *iout)
```

参数：

- [in] *src1 指向第一个输入复向量的指针
- [in] *src2 指向第二个输入复向量的指针
- [in] size 向量的复元素数量
- [out] *rout 指向实数输出的指针
- [out] *iout 指向像点输出的指针

返回值：

无。

注！

值写入两个 Q48 类型的输出变量。

riscv_dsp_cdprod_typ2_q15

原型：

```
void riscv_dsp_cprod_typ2_q15 (const q15_t *src1, const q15_t *src2, uint32_t size, q31_t *rout, q31_t *iout)
```

参数：

- [in] *src1 指向第一个输入复向量的指针
- [in] *src2 指向第二个输入复向量的指针
- [in] size 向量的复元素数量
- [out] *rout 指向实数输出的指针
- [out] *iout 指向像点输出的指针

返回值：

无。

注！

值写入两个 Q24 类型的输出变量。

3.2.3 复幅值函数

复幅值函数，用于计算复向量的幅值，实现过程如下所示：

```
for (n = 0; n < size; n++)
  dst[n] = sqrt(src[2 * n]2 + src[2 * n + 1]2);
```

DSP 软件编程函数库支持以下不同数据类型的复幅值函数：浮点型、Q31 和 Q15，以下各节详细描述各个复幅值函数。

riscv_dsp_cmag_f32

原型：

```
void riscv_dsp_cmag_f32 (const float32_t *src, float32_t *dst, uint32_t size)
```

参数：

- [in] *src 指向输入复向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的复元素数量

返回值：

无。

riscv_dsp_cmag_q31

原型：

```
void riscv_dsp_cmag_q31 (const q31_t *src, q31_t *dst, uint32_t size)
```

参数：

- [in] *src 指向输入复向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的复元素数量

返回值：

无。

注！

值写入 Q29 类型的目的向量。

riscv_dsp_cmag_q15

原型：

void riscv_dsp_cmag_q15 (const q15_t *src, q15_t *dst, uint32_t size)

参数：

- [in] *src 指向输入复向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的复元素数量

返回值：

无。

注！

值写入 Q13 类型的目的向量。

3.2.4 复平方函数

复平方函数，用于计算复向量中复数的平方，实现过程如下所示：

```
for (n = 0; n < size; n++)
  dst[n] = src[2 * n]2 + src[2 * n + 1]2;
```

DSP 软件编程函数库支持以下不同数据类型的复平方函数：浮点型、Q31 和 Q15，以下各节详细描述各个复平方函数。

riscv_dsp_cmag_sqr_f32

原型：

void riscv_dsp_cmag_sqr_f32 (const float32_t *src, float32_t *dst, uint32_t size)

参数：

- [in] *src 指向输入复向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的复元素数量

返回值：

无。

riscv_dsp_cmag_sqr_q31

原型：

void riscv_dsp_cmag_sqr_q31 (const q31_t *src, q31_t *dst, uint32_t size)

参数：

- [in] *src 指向输入复向量的指针
- [out] *dst 指向输出向量的指针

- [in] size 向量的复元素数量

返回值：

无。

注！

值写入 Q29 类型的目的向量。

riscv_dsp_cmag_sqr_q15

原型：

void riscv_dsp_cmag_sqr_q15 (const q15_t *src, q15_t *dst, uint32_t size)

参数：

- [in] *src 指向输入复向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的复元素数量

返回值：

无。

注！

值写入 Q13 类型的目的向量。

3.2.5 复数乘法函数

复数乘法函数，用于计算两个复向量的相乘，结果写入目的向量，实现过程如下所示：

```
for (n = 0; n < size; n++)
  a = src1[2 * n];
  b = src1[2 * n + 1];
  c = src2[2 * n];
  d = src2[2 * n + 1];
  dst[2 * n] = a * c - b * d;
  dst[2 * n + 1] = a * d + b * c;
```

DSP 软件编程函数库支持以下不同数据类型的复数乘法函数：浮点型、Q31 和 Q15，以下各节详细描述各个复数乘法函数。

riscv_dsp_cmul_f32

原型：

void riscv_dsp_cmul_f32 (const float32_t *src1, const float32_t *src2,

float32_t *dst, uint32_t size)

参数：

- [in] *src1 指向第一个输入复向量的指针
- [in] *src2 指向第二个输入复向量的指针
- [out] *dst 指向输出复向量的指针
- [in] size 向量的复元素数量

返回值：

无。

示例：

两组复向量，每组包含 3 个复元素[3 + 4i, 4 - 7i, -3 + 5i]和[1 - 2i, 5 - 1i, -4 + 3i]，两个复向量的乘法如下所示：

```
#define size 3
float32_t src1[2*size] = {3, 4, 4, -7, -3, 5};
float32_t src2[2*size] = {1, -2, 5, -1, -4, 3};
float32_t dst[2*size];
riscv_dsp_cmul_f32(src1, src2, dst, size);
```

本示例同样适用于 Q31 或 Q15 类型的复数乘法函数。

riscv_dsp_cmul_q31

原型：

void riscv_dsp_cmul_q31 (const q31_t *src1, const q31_t *src2, q31_t *dst, uint32_t size)

参数：

- [in] *src1 指向第一个输入复向量的指针
- [in] *src2 指向第二个输入复向量的指针
- [out] *dst 指向输出复向量的指针
- [in] size 向量的复元素数量

返回值：

无。

注！

值写入 Q29 类型的目的向量。

riscv_dsp_cmul_q15

原型：

void riscv_dsp_cmul_q15 (const q15_t *src1, const q15_t *src2, q15_t

*dst, uint32_t size)

参数：

- [in] *src1 指向第一个输入复向量的指针
- [in] *src2 指向第二个输入复向量的指针
- [out] *dst 指向输出复向量的指针
- [in] size 向量的复元素数量

返回值：

无。

注！

值写入 Q13 类型的目的向量。

3.2.6 复数实数乘法函数

复数实数乘法函数，用于计算复向量与实向量的相乘，结果写入目的向量，实现过程如下所示：

```
for (n = 0; n < size; n++)
  r = real[n];
  dst[2 * n] = src[2 * n] * r;
  dst[2 * n + 1] = src[2 * n + 1] * r;
```

DSP 软件编程函数库支持以下不同数据类型的复数实数乘法函数：浮点型、Q31 和 Q15，以下各节详细描述各个复数实数乘法函数。

riscv_dsp_cmul_real_f32

原型：

void riscv_dsp_cmul_real_f32 (const float32_t *src, const float32_t *real, float32_t *dst, uint32_t size)

参数：

- [in] *src 指向输入复向量的指针
- [in] *real 指向输入实数向量的指针
- [out] *dst 指向输出复向量的指针
- [in] size 向量的元素数量

返回值：

无。

示例：

一个复向量包含 3 个元素[3 + 4i, 4 - 7i, -3 + 5i]，一个实向量包含 3 个

元素[1, -3, 2]，两个向量的乘法如下所示：

```
#define size 3
float32_t src[2*size] = {3, 4, 4, -7, -3, 5};
float32_t real[size] = {1, -3, 2};
float32_t dst[2*size];
riscv_dsp_cmul_real_f32(src, real, dst, size);
```

本示例同样适用于 Q31 或 Q15 类型的复数实数乘法函数。

riscv_dsp_cmul_real_q31

原型：

```
void riscv_dsp_cmul_real_q31 (const q31_t *src, const q31_t *real, q31_t *dst, uint32_t size)
```

参数：

- [in] *src 指向输入复向量的指针
- [in] *real 指向输入实数向量的指针
- [out] *dst 指向输出复向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

结果值饱和为 Q31 类型的范围[0x80000000, 0x7FFFFFFF]。

riscv_dsp_cmul_real_q15

原型：

```
void riscv_dsp_cmul_real_q15 (const q15_t *src, const q15_t *real, q15_t *dst, uint32_t size)
```

参数：

- [in] *src 指向输入复向量的指针
- [in] *real 指向输入实数向量的指针
- [out] *dst 指向输出复向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

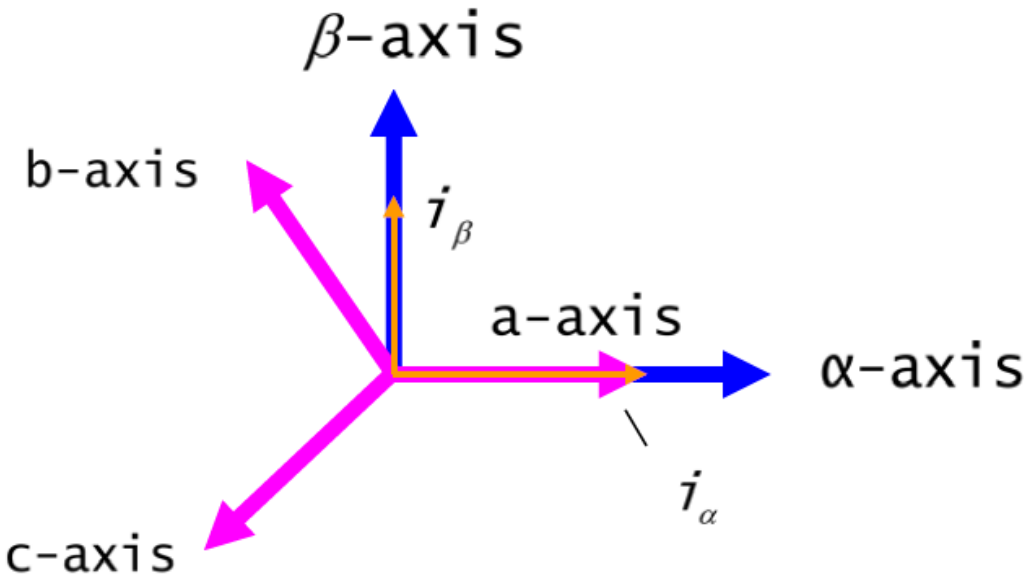
结果值饱和为 Q15 类型的范围[0x8000, 0x7FFF]。

3.3 控制器函数

3.3.1 Clarke 变换函数

Clarke 变换函数，用于将三相量从三相系统转换到两轴参考系（α-轴和β-轴），如图 3-1 所示。

图 3-1 Clarke Transform Diagram



Clarke 变换实现过程如下所示：

let $I_a + I_b + I_c = 0$

such that,

isqrt3 = 1 / sqrt(3);

$i_\alpha = I_a$;

i_β = isqrt3 * I_a + 2 * isqrt3 * I_b

DSP 软件编程函数库支持以下不同数据类型的 Clarke 变换函数：浮点型和 Q31，以下各节详细描述各个 Clarke 变换函数。

riscv_dsp_clarke_f32

原型：

void riscv_dsp_clarke_f32 (float32_t a, float32_t b, float32_t *alpha, float32_t *beta)

参数：

- [in] a 三相系统的三相量 a

- [in] b 三相系统的三相量 b
- [out] *alpha 两轴参考系的 α-轴
- [out] *beta 两轴参考系的 β-轴

返回值：

无。

riscv_dsp_clarke_q31

原型：

void riscv_dsp_clarke_q31 (q31_t a, q31_t b, q31_t *alpha, q31_t *beta)

参数：

- [in] a 三相系统的三相量 a
- [in] b 三相系统的三相量 b
- [out] *alpha 两轴参考系的 α-轴
- [out] *beta 两轴参考系的 β-轴

返回值：

无。

3.3.2 Clarke 反变换函数

Clarke 反变换函数，用于将两轴参考系（α-轴和 β-轴）转换回三相系统 a、b 和 c 三相量，实现过程如下所示：

$I_a = i_\alpha$;

$I_b = -0.5 * i_\alpha + 0.5 * sqrt(3) * i_\beta$;

DSP 软件编程函数库支持以下不同数据类型的 Clarke 反变换函数：浮点型和 Q31，以下各节详细描述各个 Clarke 反变换函数。

riscv_dsp_inv_clarke_f32

原型：

void riscv_dsp_inv_clarke_f32 (float32_t alpha, float32_t beta, float32_t *a, float32_t *b)

参数：

- [in] alpha 两轴参考系的 α-轴
- [in] beta 两轴参考系的 β-轴
- [out] *a 三相系统的三相量 a
- [out] *b 三相系统的三相量 b

返回值：

无。

riscv_dsp_inv_clarke_q31

原型：

void riscv_dsp_inv_clarke_q31 (q31_t alpha, q31_t beta, q31_t *a, q31_t *b)

参数：

- [in] alpha 两轴参考系的 α-轴
- [in] beta 两轴参考系的 β-轴
- [out] *a 三相系统的三相量 a
- [out] *b 三相系统的三相量 b

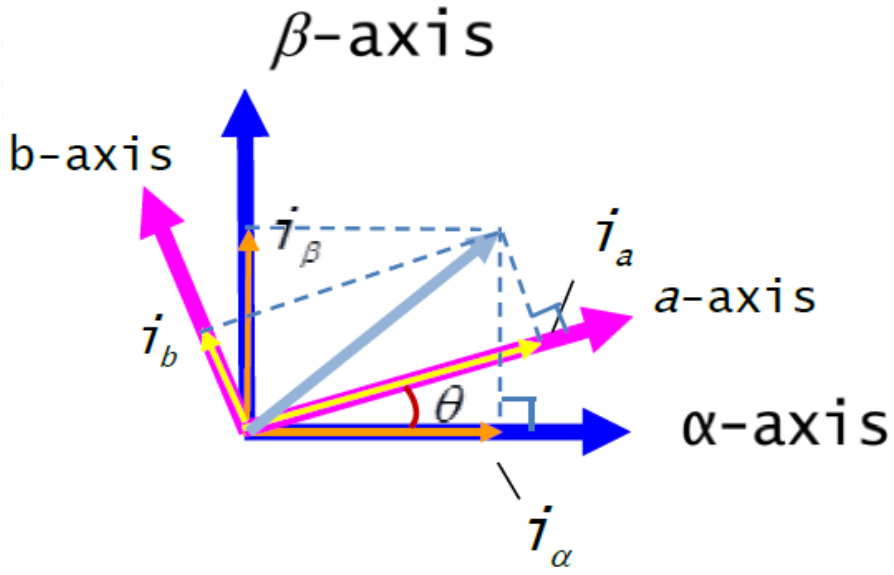
返回值：

无。

3.3.3 Park 变换函数

Park 变换函数，通过旋转 θ 角将输入的两相坐标（α-和 β-轴）转换为转子框架(a-和 b-轴)，如图 3-2 所示。

图 3-2 Park Transform Diagram



Park 变换实现过程如下所示：

Cosval = cos(θ); Sinval = sin(θ);

$i_a = i_\alpha$ * Cosval + i_β * Sinval;

$i_b = -i_\alpha * Sinval + i_\beta * Cosval$

DSP 软件编程函数库支持以下不同数据类型的 Park 变换函数：浮点型和 Q31，以下各节详细描述各个 Park 变换函数。

riscv_dsp_park_f32

原型：

void riscv_dsp_park_f32 (float32_t alpha, float32_t beta, float32_t *a, float32_t *b, float32_t sin, float32_t cos)

参数：

- [in] alpha 两相坐标的 α-轴
- [in] beta 两相坐标的 β-轴
- [out] *a 一个输出转子框架的 a-轴
- [out] *b 一个输出转子框架的 b-轴
- [in] sin 旋转角 θ 的正弦值
- [in] cos 旋转角 θ 的余弦值

返回值：

无。

riscv_dsp_park_q31

原型：

void riscv_dsp_park_q31 (q31_t alpha, q31_t beta, q31_t *a, q31_t *b, q31_t cos, q31_t *sin)

参数：

- [in] alpha 两相坐标的 α-轴
- [in] beta 两相坐标的 β-轴
- [out] *a 一个输出转子框架的 a-轴
- [out] *b 一个输出转子框架的 b-轴
- [in] sin 旋转角 θ 的正弦值
- [in] cos 旋转角 θ 的余弦值

返回值：

无。

3.3.4 Park 反变换函数

Park 反变换函数，通过旋转 θ 角将转子框架（a-和 b-轴）转换回两相坐标（α-和 β-轴），实现过程如下所示：

$i_\alpha = i_a * Cosval - ib * Sinval;$

$i\beta = i\alpha * Sinval + ib * Cosval$

DSP 软件编程函数库支持以下不同数据类型的 Park 反变换函数：浮点型和 Q31，以下各节详细描述各个 Park 反变换函数。

riscv_dsp_inv_park_f32

原型：

void riscv_dsp_inv_park_f32 (float32_t a, float32_t b, float32_t *alpha, float32_t *beta, float32_t sin, float32_t cos)

参数：

- [in] a 一个输入转子框架的 a-轴
- [in] b 一个输入转子框架的 b-轴
- [out] *alpha 一个输出两相坐标的 α-轴
- [out] *beta 一个输出两相坐标的 β-轴
- [in] sin 旋转角 θ 的正弦值
- [in] cos 旋转角 θ 的余弦值

返回值：

无。

riscv_dsp_inv_park_q31

原型：

void riscv_dsp_inv_park_f32 (q31_t a, float32_t b, q31_t *alpha, q31_t *beta, q31_t sin, q31_t cos)

参数：

- [in] a 一个输入转子框架的 a-轴
- [in] b 一个输入转子框架的 b-轴
- [out] *alpha 一个输出两相坐标的 α-轴
- [out] *beta 一个输出两相坐标的 β-轴
- [in] sin 旋转角 θ 的正弦值
- [in] cos 旋转角 θ 的余弦值

返回值：

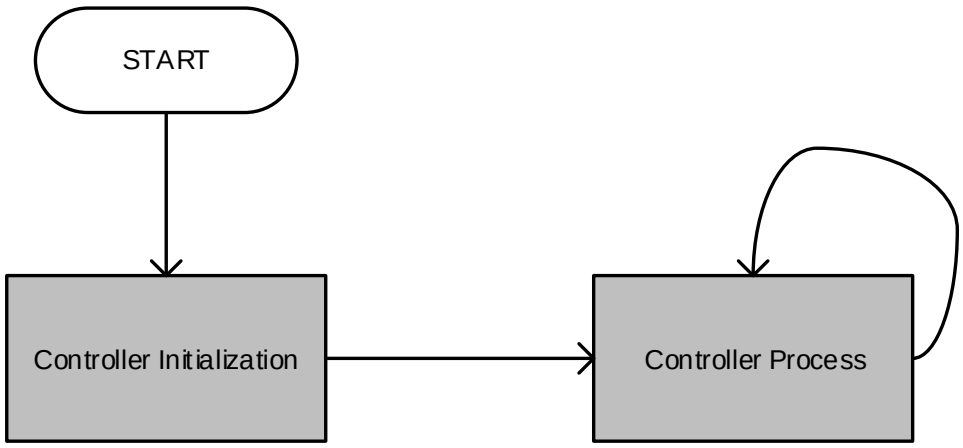
无。

3.3.5. PID 函数

PID（Proportional-Integral-Differential，比例-积分-微分）控制器函

数，提供了一种连续循环反馈机制，根据周期性输入控制输出误差，PID 控制器状态如图 3-3 所示。

图 3-3 State of PID Controller



每次当前数据输入控制器程序时，PID 实现过程如下所示：

```
output = Kp * current_data + Ki * state0 + Kd * state1 + state2
state2 = output;
state1 = state0;
state0 = current_data;
```

DSP 软件编程函数库支持以下不同数据类型的 PID 函数：浮点型和 Q31，以下各节详细描述各个 PID 函数。

riscv_dsp_init_pid_f32

原型：

```
void riscv_dsp_init_pid_f32 (riscv_dsp_pid_f32_t *instance, int32_t set)
```

参数：

- riscv_dsp_pid_f32_t 浮点 PID 函数的实例化结构体，定义如下：

```
typedef struct
{
  float32_t gain1;
  float32_t gain2;
  float32_t gain3;
  float32_t state[3];
  float32_t Kp;
  float32_t Ki;
```

```
    float32_t Kd;
} riscv_dsp_pid_f32_t;
```

其中，

- gain1 等于 Kp + Ki + Kd
- gain2 等于-Kp - 2Kd
- gain3 等于 Kd
- state[3] 状态缓存区
- Kp 比例值
- Ki 积分值
- Kd 导数值

- [in] *instance 指向实例化结构体的指针
- [in] set 如果该值为 0，不做任何事情，否则清除状态缓存区

返回值：

无。

注！

- 执行 riscv_dsp_pid_f32 函数前，须调用此函数来初始化控制器；
- 当参数 gain1、gain2、gain3 和 state[3]被控制器管理时，应输入 Kp、Ki 和 Kd。

示例：

```
riscv_dsp_pid_f32_t instance;
instance.Kp = 0.02F;
instance.Ki = 0.03F;
instance.Kd = 0.014F;
//initialize controller
riscv_dsp_init_pid_f32(&instance, 1);
//Begin controller
while (1)
{
    f32_t output;
    f32_t input = … read data from input port …
    output = riscv_dsp_pid_f32(&instance, input);
    …
}
```

本示例同样适用于 Q31 或 Q15 类型的 PID 函数。

riscv_dsp_pid_f32

原型：

float32_t riscv_dsp_pid_f32 (riscv_dsp_pid_f32_t *instance, float32_t src)

参数：

- riscv_dsp_pid_f32_t 浮点 PID 函数的实例化结构体，定义如下：

```
typedef struct
{
  float32_t gain1;
  float32_t gain2;
  float32_t gain3;
  float32_t state[3];
  float32_t Kp;
  float32_t Ki;
  float32_t Kd;
} riscv_dsp_pid_f32_t;
```

其中，

  - gain1 等于 Kp + Ki + Kd
  - gain2 等于-Kp - 2Kd
  - gain3 等于 Kd
  - state[3] 状态缓存区
  - Kp 比例值
  - Ki 积分值
  - Kd 导数值

- [in] *instance 指向实例化结构体的指针
- [in] src 当前数据

返回值：

控制器的输出。

注！

首先须调用 riscv_dsp_init_pid_f32 函数来初始化控制器；然后使用当前数据循环执行 riscv_dsp_pid_f32 函数来调整控制器的输出，具体参考 riscv_dsp_init_pid_f32 示例。

riscv_dsp_init_pid_q31

原型：

void riscv_dsp_init_pid_q31 (riscv_dsp_pid_q31_t *instance, int32_t set)

参数：

- riscv_dsp_pid_q31_t Q31 PID 函数的实例化结构体，定义如下：

```
typedef struct
{
  q31_t gain1;
  q31_t gain2;
  q31_t gain3;
  q31_t state[3];
  q31_t Kp;
  q31_t Ki;
  q31_t Kd;
} riscv_dsp_pid_q31_t;
```

其中，

  - gain1 等于 Kp + Ki + Kd
  - gain2 等于-Kp - 2Kd
  - gain3 等于 Kd
  - state[3] 状态缓存区
  - Kp 比例值
  - Ki 积分值
  - Kd 导数值

- [in] *instance 指向实例化结构体的指针
- [in] set 如果该值为 0，不做任何事情，否则清除状态缓存区

返回值：

无。

注！

- 执行 riscv_dsp_pid_ q31 函数前，须调用此函数来初始化控制器；
- 当参数 gain1、gain2、gain3 和 state[3]被控制器管理时，应输入 Kp、Ki 和 Kd。

riscv_dsp_pid_q31

原型：

q31_t riscv_dsp_pid_q31 (riscv_dsp_pid_q31_t *instance, q31_t src)

参数：

- riscv_dsp_pid_q31_t Q31 PID 函数的实例化结构体，定义如下：

```
typedef struct
{
  q31_t gain1;
  q31_t gain2;
  q31_t gain3;
  q31_t state[3];
  q31_t Kp;
  q31_t Ki;
  q31_t Kd;
} riscv_dsp_pid_q31_t;
```

其中，

  - gain1 等于 Kp + Ki + Kd
  - gain2 等于-Kp - 2Kd
  - gain3 等于 Kd
  - state[3] 状态缓存区
  - Kp 比例值
  - Ki 积分值
  - Kd 导数值

- [in] *instance 指向实例化结构体的指针
- [in] src 当前数据

返回值：

控制器的输出。

注！

首先须调用 riscv_dsp_init_pid_ q31 函数来初始化控制器；然后使用当前数据循环执行 riscv_dsp_pid_ q31 函数来调整控制器的输出，具体参考 riscv_dsp_init_pid_f32 示例。

riscv_dsp_init_pid_q15

原型：

void riscv_dsp_init_pid_q15 (riscv_dsp_pid_q15_t *instance, int32_t set)

参数：

- riscv_dsp_pid_ q15_t Q15 PID 函数的实例化结构体，定义如下：

```
typedef struct
{
  q15_t gain1;
  q15_t gain2;
  q15_t gain3;
  q15_t state[3];
  q15_t Kp;
  q15_t Ki;
  q15_t Kd;
} riscv_dsp_pid_ q15_t;
```

其中，

  - gain1 等于 Kp + Ki + Kd
  - gain2 等于-Kp - 2Kd
  - gain3 等于 Kd
  - state[3] 状态缓存区
  - Kp 比例值
  - Ki 积分值
  - Kd 导数值

- [in] *instance 指向实例化结构体的指针
- [in] set 如果该值为 0，不做任何事情，否则清除状态缓存区

返回值：

无。

注！

- 执行 riscv_dsp_pid_ q15 函数前，须调用此函数来初始化控制器；
- 当参数 gain1、gain2、gain3 和 state[3]被控制器管理时，应输入 Kp、Ki 和 Kd。

riscv_dsp_pid_q15

原型：

```
q15_t riscv_dsp_pid_q15 (riscv_dsp_pid_q15_t *instance, q15_t src)
```

参数：

- riscv_dsp_pid_ q15_t Q15 PID 函数的实例化结构体，定义如下：

```
typedef struct
{
  q15_t gain1;
  q15_t gain2;
  q15_t gain3;
  q15_t state[3];
  q15_t Kp;
  q15_t Ki;
  q15_t Kd;
} riscv_dsp_pid_ q15_t;
```

  其中，

  - gain1 等于 Kp + Ki + Kd
  - gain2 等于-Kp - 2Kd
  - gain3 等于 Kd
  - state[3] 状态缓存区
  - Kp 比例值
  - Ki 积分值
  - Kd 导数值

- [in] *instance 指向实例化结构体的指针
- [in] src 当前数据

返回值：

控制器的输出。

注！

首先须调用 riscv_dsp_init_pid_ q15 函数来初始化控制器；然后使用当前数据循环执行 riscv_dsp_pid_ q15 函数来调整控制器的输出，具体参考 riscv_dsp_init_pid_f32 示例。

3.4 滤波函数

3.4.1 双二阶滤波器（直接 1 型）函数

双二阶滤波器函数提供双二阶滤波器，DSP 软件编程函数库支持如下的直接 1 型双二阶滤波器等式：

dst[n] = b0 * src[n] + b1 * src[n-1] + b2 * src[n-2] + a1 * dst[n-1] + a2 * dst[n-2];

在等式中，每阶段使用 5 个系数和 4 个状态变量，系数 b0、b1 和 b2 确定零点位置，系数 a1 和 a2 确定极点位置。

DSP 软件编程函数库支持以下不同数据类型的直接 1 型双二阶滤波器函数和实例化结构体：浮点型、Q31、Q15 和 32x64 Q31。实例化结构体，用于保存阶段和系统等信息，以及每个阶段的状态。在使用双二阶滤波器函数之前，确保已初始化相应的实例化结构体。

以下各节详细描述各个双二阶滤波器函数。

riscv_dsp_bq_df1_f32

原型：

void riscv_dsp_bq_df1_f32 (const riscv_dsp_bq_df1_f32_t *instance, float32_t *src, float32_t *dst, uint32_t size)

参数：

- riscv_dsp_bq_df1_f32_t 浮点双二次级联滤波器的实例化结构体，定义如下：

  typedef struct

  {

  uint32_t nstage;

  float32_t *state;

  float32_t *coeff;

  } riscv_dsp_bq_df1_f32_t;

  其中，

  - nstage 过滤器的各个阶段
  - *state 指向状态向量的指针，其大小为 4*nstage
  - *coeff 指向系数向量的指针，其大小为 5*nstage
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 每个阶段的样本数量

返回值：

无。

示例：

#define nstage 3

```
#define size 6
float32_t state[4 * nstage] = {0.0};
float32_t coeff[5 * nstage] = {0.40, 0.10, 0.24, -0.40, -0.34, 0.20, 0.06, 0.28, -0.04, -0.20, 0.08, 0.40, 0.60, -1.00, -0.14};
float32_t src[size] = {1.0, 0.5, 0.4, -0.1, -0.1, 0.3};
float32_t dst[size];
riscv_dsp_bq_df1_f32_t instance = {nstage, state, coeff};
riscv_dsp_bq_df1_f32(&instance, src, dst, size);
```

riscv_dsp_bq_df1_q31

原型：

```
void riscv_dsp_bq_df1_q31 (const riscv_dsp_bq_df1_q31_t *instance, q31_t *src, q31_t *dst, uint32_t size)
```

参数：

- riscv_dsp_bq_df1_q31_t Q31 双二次级联滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t nstage;
  q31_t *state;
  q31_t *coeff;
  int8_t shift;
} riscv_dsp_bq_df1_q31_t;
```

其中，

  - nstage 过滤器的各个阶段
  - *state 指向状态向量的指针，其大小为 4*nstage
  - *coeff 指向系数向量的指针，其大小为 5*nstage
  - shift 移位位数

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 每个阶段的样本数量

返回值：

无。

示例：

```
#define shift 0
#define nstage 3
#define size 6
q31_t state_q31[4 * nstage] = {0};
q31_t coeff_q31[5 * nstage], src_q31[size], dst_q31[size];
riscv_dsp_bq_df1_q31_t instance_q31 = {nstage, state_q31, coeff_q31, shift};
float32_t coeff_f32[5 * nstage] = {0.40, 0.10, 0.24, -0.40, -0.34, 0.20, 0.06, 0.28, -0.04, -0.20, 0.08, 0.40, 0.60, -1.00, -0.14};
float32_t src_f32[size] = {1.0, 0.5, 0.4, -0.1, -0.1, 0.3};
riscv_dsp_convert_f32_q31(coeff_f32, coeff_q31, 5 * nstage);
riscv_dsp_convert_f32_q31(src_f32, src_q31, size);
riscv_dsp_bq_df1_q31(&instance_q31, src_q31, dst_q31, size);
```

此 Q31 类型的双二阶滤波器函数，使用 riscv_dsp_convert_f32_q31 函数转为 F32 向量为 Q31 类型。本示例同样适用于 Q15 类型的双二阶滤波器函数。

riscv_dsp_bq_df1_fast_q31

原型：

```
void riscv_dsp_bq_df1_fast_q31 (const riscv_dsp_bq_df1_q31_t *instance, q31_t *src, q31_t *dst, uint32_t size)
```

参数：

- riscv_dsp_bq_df1_q31_t Q31 双二次级联滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t nstage;
  q31_t *state;
  q31_t *coeff;
  int8_t shift;
} riscv_dsp_bq_df1_q31_t;
```

其中，

- nstage 过滤器的各个阶段
- *state 指向状态向量的指针，其大小为 4*nstage
- *coeff 指向系数向量的指针，其大小为 5*nstage
- shift 移位位数

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 每个阶段的样本数量

返回值：

无。

riscv_dsp_bq_df1_q15

原型：

void riscv_dsp_bq_df1_q15 (const riscv_dsp_bq_df1_q15_t *instance, q15_t *src, q15_t *dst, uint32_t size)

参数：

- riscv_dsp_bq_df1_q15_t Q15 双二次级联滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t nstage;
  q15_t *state;
  q15_t *coeff;
  int8_t shift;
} riscv_dsp_bq_df1_q15_t;
```

其中，

- nstage 过滤器的各个阶段
- *state 指向状态向量的指针，其大小为 4*nstage
- *coeff 指向系数向量的指针，其大小为 5*nstage 或 6*nstage（当使用 DSP 扩展时），请参照以下注释说明。
- shift 移位位数

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针

- [in] size 每个阶段的样本数量

返回值：

无。

注！

系数向量以如下顺序保存系数{b0, b1, b2, a1, a2, ...}，前 5 个元素用于第一阶段，接下来 5 个元素用于接下来的阶段，以此类推。如果使用 DSP 扩展，则顺序变为{b0, 0, b1, b2, a1, a2, ...}，其中，zero (0)元素插入到 b0 和 b1 之间，用于 SIMD 加载/保存，这种情况下，前 6 个元素用于第一阶段，接下来 6 个元素用于接下来的阶段，以此类推。

riscv_dsp_bq_df1_fast_q15

原型：

void riscv_dsp_bq_df1_fast_q15 (const riscv_dsp_bq_df1_q15_t *instance, q15_t *src, q15_t *dst, uint32_t size)

参数：

- riscv_dsp_bq_df1_q15_t Q15 双二次级联滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t nstage;
  q15_t *state;
  q15_t *coeff;
  int8_t shift;
} riscv_dsp_bq_df1_q15_t;
```

其中，

  - nstage 过滤器的各个阶段
  - *state 指向状态向量的指针，其大小为 4*nstage
  - *coeff 指向系数向量的指针，其大小为 5*nstage 或 6*nstage（当使用 DSP 扩展时），请参照以下注释说明。
  - shift 移位位数

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 每个阶段的样本数量

返回值：

无。

注！

系数向量以如下顺序保存系数{b0, b1, b2, a1, a2, …}，前 5 个元素用于第一阶段，接下来 5 个元素用于接下来的阶段，以此类推。如果使用 DSP 扩展，则顺序变为{b0, 0, b1, b2, a1, a2, …}，其中，zero (0)元素插入到 b0 和 b1 之间，用于 SIMD 加载/保存，这种情况下，前 6 个元素用于第一阶段，接下来 6 个元素用于接下来的阶段，以此类推。

riscv_dsp_bq_df1_32x64_q31

原型：

void riscv_dsp_bq_df1_32x64_q31 (const riscv_dsp_bq_df1_32x64_q31_t *instance, q31_t *src, q31_t *dst, uint32_t size)

参数：

- riscv_dsp_bq_df1_32x64_q31_t 32x64 Q31 双二次级联滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t nstage;
  q63_t *state;
  q31_t *coeff;
  int8_t shift;
} riscv_dsp_bq_df1_32x64_q31_t;
```

其中，

  - nstage 过滤器的各个阶段
  - *state 指向状态向量的指针，其大小为 4*nstage
  - *coeff 指向系数向量的指针，其大小为 5*nstage
  - shift 移位位数
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 每个阶段的样本数量

返回值：

无。

注！

使用一个 64 位状态缓存区来计算累加值，此函数用此缓存区保存与更新中间值。

3.4.2 双二阶滤波器（转置直接 2 型）函数

双二阶滤波器函数，提供双二阶滤波器，DSP 软件编程函数库支持如下的转置直接 2 型双二阶滤波器等式：

```
for (n = 0; n < size; n++)
  out[n] = b0 * in[n] + d0;
  d0 = b1 * in[n] + a1 * out[n] + d1;
  d1 = b2 * in[n] + a2 * out[n];
```

在等式中，每阶段使用 5 个系数（b0、b1、b2、a1 和 a2）和 2 个状态变量（d0 和 d1）。在每个阶段完成后，将两个状态变量 d0 和 d1 保存为一个状态向量。

DSP 软件编程函数库支持转置直接 2 型双二阶滤波器用于立体声（2 通道）信号，其等式如下：

```
for (n = 0; n < size; n+=2)
  out[n] = b0 * in[n] + d00;
  out[n + 1] = b0 * in[n + 1] + d01;
  d00 = b1 * in[n] + a1 * out[n] + d10;
  d01 = b1 * in[n + 1] + a1 * out[n + 1] + d11;
  d10 = b2 * in[n] + a2 * out[n];
  d11 = b2 * in[n + 1] + a2 * out[n + 1];
```

在等式中，每阶段使用 5 个系数（b0、b1、b2、a1 和 a2）和 4 个状态变量（d00、d01、d10 和 d11）。在每个阶段完成后，将四个状态变量保存为一个状态向量。

DSP 软件编程函数库支持以下不同数据类型的直接 2 型双二阶滤波器函数和实例化结构体：浮点型、F32 和 F64。实例化结构体，用于保存阶段和系统等信息，以及每个阶段的状态。在使用双二阶滤波器函数之前，确保已初始化相应的实例化结构体。

以下各节详细描述各个双二阶滤波器函数。

riscv_dsp_bq_df2T_f32

原型：

```
void riscv_dsp_bq_df2T_f32 (const riscv_dsp_bq_df2T_f32_t *instance, float32_t *src, float32_t *dst, uint32_t size)
```

参数：

- riscv_dsp_bq_df2T_f32_t F32 双二次级联滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t nstage;
  float32_t *state;
  float32_t *coeff;
} riscv_dsp_bq_df2T_f32_t;
```

其中，

- nstage 过滤器的各个阶段
- *state 指向状态向量的指针，其大小为 2*nstage
- *coeff 指向系数向量的指针，其大小为 5*nstage

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 每个阶段的样本数量

返回值：

无。

示例：

```
#define nstage 3
#define size 6
float32_t state[2 * nstage] = {0.0};
float32_t coeff[5 * nstage] = {0.40, 0.10, 0.24, -0.40, -0.34, 0.20, 0.06, 0.28, -0.04, -0.20, 0.08, 0.40, 0.60, -1.00, -0.14};
float32_t src[size] = {1, 0.5, 0.4, -0.1, -0.1, 0.3};
float32_t dst[size];
riscv_dsp_bq_df2T_f32_t inst = {nstage, state, coeff};
riscv_dsp_bq_df2T_f32(&inst, src, dst, size);
```

本示例同样适用于 F64 类型的转置直接 2 型双二阶滤波器。

riscv_dsp_bq_df2T_f64

原型：

```
void riscv_dsp_bq_df2T_f64 (const riscv_dsp_bq_df2T_f64_t *instance, float64_t *src, float64_t *dst, uint32_t size)
```

参数：

- riscv_dsp_bq_df2T_f64_t F64 双二次级联滤波器的实例化结构体，定

义如下：

```
typedef struct
{
  uint32_t nstage;
  float64_t *state;
  float64_t *coeff;
} riscv_dsp_bq_df2T_f64_t;
```

其中，

- nstage 过滤器的各个阶段
- *state 指向状态向量的指针，其大小为 2*nstage
- *coeff 指向系数向量的指针，其大小为 5*nstage

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 每个阶段的样本数量

返回值：

riscv_dsp_bq_stereo_df2T_f32

原型：

void riscv_dsp_bq_stereo_df2T_f32(const riscv_dsp_bq_stereo_df2T_f32_t *instance, float32_t *src, float32_t *dst, uint32_t size);

参数：

- riscv_dsp_bq_stereo_df2T_f32_t F32 双二次级联滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t nstage;
  float32_t *state;
  float32_t *coeff;
} riscv_dsp_bq_stereo_df2T_f32_t;
```

其中，

- nstage 过滤器的各个阶段

- *state 指向状态向量的指针，其大小为 4*nstage
- *coeff 指向系数向量的指针，其大小为 5*nstage

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 每个阶段的样本数量

返回值：

无。

示例：

```
#define nstage 3
#define size 1024
float32_t state[4 * nstage] = {0.0};
float32_t coeff[5 * nstage] = {0.40, 0.10, 0.24, -0.40, -0.34, 0.20, 0.06, 0.28, -0.04, -0.20, 0.08, 0.40, 0.60, -1.00, -0.14};
float32_t src[size * 2] = {…};
float32_t dst[size * 2];
riscv_dsp_bq_stereo_df2T_f32_t inst = {nstage, state, coeff};
riscv_dsp_bq_stereo_df2T_f32(&inst, src, dst, size);
```

3.4.3 卷积函数

卷积函数，在数学上把两个信号组合为一个信号，卷积方程可以表示为：

$$dst[n] = \sum_{k=0}^{len1} (src1[k] \times src2[n-k])$$

其中，src1[n]的大小为 len1，src2[n]的大小为 len2，dst[n]的大小为 len1 + len2 - 1。

DSP 软件编程函数库支持以下不同数据类型的卷积函数：浮点型、Q31、Q15 和 Q7，以下各节详细描述各个卷积函数。

riscv_dsp_conv_f32

原型：

```
void riscv_dsp_conv_f32 (float32_t *src1, uint32_t len1, float32_t *src2, uint32_t len2, float32_t *dst)
```

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] len1 第一个输入向量的长度
- [in] *scr2 指向第二个输入向量的指针
- [in] len2 第二个输入向量的长度
- [out] *dst 指向输出向量的指针

返回值：

无。

示例：

```
#define len1 6
#define len2 4
float32_t src1[len1] = {0.3, 0.2, -0.1, 0.2, 0.4, 0.2};
float32_t src2[len2] = {-0.1, 0.3, 0.2, -0.2};
uint32_t dst[len1 + len2 - 1];
riscv_dsp_conv_f32(src1, len1, src2, len2, dst);
```

本示例同样适用于 Q31、Q15 或 Q7 类型的卷积函数。

riscv_dsp_conv_q31

原型：

```
void riscv_dsp_conv_q31 (q31_t *src1, uint32_t len1, q31_t *src2, uint32_t len2, q31_t *dst)
```

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] len1 第一个输入向量的长度
- [in] *scr2 指向第二个输入向量的指针
- [in] len2 第二个输入向量的长度
- [out] *dst 指向输出向量的指针

返回值：

无。

riscv_dsp_conv_q15

原型：

```
void riscv_dsp_conv_q15 (q15_t *src1, uint32_t len1, q15_t *src2, uint32_t len2, q15_t *dst)
```

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] len1 第一个输入向量的长度
- [in] *scr2 指向第二个输入向量的指针
- [in] len2 第二个输入向量的长度
- [out] *dst 指向输出向量的指针

返回值：

无。

riscv_dsp_conv_q7

原型：

void riscv_dsp_conv_q7 (q7_t *src1, uint32_t len1, q7_t *src2, uint32_t len2, q7_t *dst)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] len1 第一个输入向量的长度
- [in] *scr2 指向第二个输入向量的指针
- [in] len2 第二个输入向量的长度
- [out] *dst 指向输出向量的指针

返回值：

无。

3.4.4 部分卷积函数

通过指定起始索引和大小，部分卷积函数对两个信号进行部分卷积。部分卷积结果将生成到[start_index，…，start_index + size - 1]范围内的目的向量中。因此，如果部分卷积结果不在[0，…，len1 + len2 - 2]范围内，则给出参数误差的返回值-1。

DSP 软件编程函数库支持以下不同数据类型的部分卷积函数：浮点型、Q31、Q15 和 Q7，以下各节详细描述各个部分卷积函数。

riscv_dsp_conv_partial_f32

原型：

int32_t riscv_dsp_conv_partial_f32 (float32_t *src1, uint32_t len1, float32_t *src2, uint32_t len2, float32_t *dst, uint32_t startindex, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针

- [in] len1 第一个输入向量的长度
- [in] *scr2 指向第二个输入向量的指针
- [in] len2 第二个输入向量的长度
- [out] *dst 指向输出向量的指针
- [in] startindex 部分卷积的起始索引
- [in] size 部分卷积的长度

返回值：

0 成功

-1 失败

示例：

```
#define len1 6
#define len2 4
float32_t src1[len1] = {0.3, 0.2, -0.1, 0.2, 0.4, 0.2};
float32_t src2[len2] = {-0.1, 0.3, 0.2, -0.2};
uint32_t dst[len1 + len2 - 1];
riscv_dsp_conv_partial_f32(src1, len1, src2, len2, dst, 3, 4);
```

本示例同样适用于 Q31、Q15 或 Q7 类型的部分卷积函数。

riscv_dsp_conv_partial_q31

原型：

```
int32_t riscv_dsp_conv_partial_q31 (q31_t *src1, uint32_t len1, q31_t *src2, uint32_t len2, q31_t *dst, uint32_t startindex, uint32_t size)
```

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] len1 第一个输入向量的长度
- [in] *scr2 指向第二个输入向量的指针
- [in] len2 第二个输入向量的长度
- [out] *dst 指向输出向量的指针
- [in] startindex 部分卷积的起始索引
- [in] size 部分卷积的长度

返回值：

0 成功

-1 失败

riscv_dsp_conv_partial_q15

原型：

int32_t riscv_dsp_conv_partial_q15(q15_t *src1, uint32_t len1, q15_t *src2, uint32_t len2, q15_t *dst, uint32_t startindex, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] len1 第一个输入向量的长度
- [in] *scr2 指向第二个输入向量的指针
- [in] len2 第二个输入向量的长度
- [out] *dst 指向输出向量的指针
- [in] startindex 部分卷积的起始索引
- [in] size 部分卷积的长度

返回值：

0 成功

-1 失败

riscv_dsp_conv_partial_q7

原型：

int32_t riscv_dsp_conv_partial_q7(q7_t *src1, uint32_t len1, q7_t *src2, uint32_t len2, q7_t *dst, uint32_t startindex, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] len1 第一个输入向量的长度
- [in] *scr2 指向第二个输入向量的指针
- [in] len2 第二个输入向量的长度
- [out] *dst 指向输出向量的指针
- [in] startindex 部分卷积的起始索引
- [in] size 部分卷积的长度

返回值：

0 成功

-1 失败

3.4.5 相关函数

相关函数，在数学上使用两个信号形成另一个信号（也称为互相关），

相关函数的数学运算与卷积函数类似，可以表示为：

$$\mathrm{dst[n]} = \sum_{k=0}^{len1} (\mathrm{src[k]} \times \ src2[k-n])$$

其中，src1 和 src2 为输入向量，长度分别为 len1 和 len2；dst 为输出向量，其大小为 2 * max (len1, len2) - 1。

DSP 软件编程函数库支持以下不同数据类型的相关函数：浮点型、Q31、Q15 和 Q7，以下各节详细描述各个相关函数。

riscv_dsp_corr_f32

原型：

void riscv_dsp_corr_f32 (float32_t *src1, uint32_t len1, float32_t *src2, uint32_t len2, float32_t *dst)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] len1 第一个输入向量的长度
- [in] *scr2 指向第二个输入向量的指针
- [in] len2 第二个输入向量的长度
- [out] *dst 指向输出向量的指针

返回值：

无。

示例：

```
#define len1 7
#define len2 5
float32_t src1[len1] = {0.3, 0.2, -0.1, 0.2, 0.4, 0.2, -0.1};
float32_t src2[len2] = {-0.1, 0.3, 0.2, -0.2, -0.2};
float32_t dst[2 * len1 - 1];
riscv_dsp_conv_f32(src1, len1, src2, len2, dst);
```

本示例同样适用于 Q31、Q15 或 Q7 类型的相关函数。

riscv_dsp_corr_q31

原型：

void riscv_dsp_corr_q31 (q31_t *src1, uint32_t len1, q31_t *src2, uint32_t len2, q31_t *dst)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] len1 第一个输入向量的长度
- [in] *scr2 指向第二个输入向量的指针
- [in] len2 第二个输入向量的长度
- [out] *dst 指向输出向量的指针

返回值：

无。

riscv_dsp_corr_q15

原型：

void riscv_dsp_corr_q15 (q15_t *src1, uint32_t len1, q15_t *src2, uint32_t len2, q15_t *dst)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] len1 第一个输入向量的长度
- [in] *scr2 指向第二个输入向量的指针
- [in] len2 第二个输入向量的长度
- [out] *dst 指向输出向量的指针

返回值：

无。

riscv_dsp_corr_q7

原型：

void riscv_dsp_corr_q7 (q7_t *src1, uint32_t len1, q7_t *src2, uint32_t len2, q7_t *dst)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] len1 第一个输入向量的长度
- [in] *scr2 指向第二个输入向量的指针
- [in] len2 第二个输入向量的长度
- [out] *dst 指向输出向量的指针

返回值：

无。

3.4.6 FIR 滤波器函数

FIR 滤波器函数，等式如下所示：

$$\mathrm{dst[n]} = \sum_{k=0}^{Size} (\mathrm{b[k]} \times src[n-k])$$

其中，b[k]是滤波器系数，Size 是滤波器系数的数量。

DSP 软件编程函数库支持以下不同数据类型的 FIR 滤波器函数和实例化结构体：浮点型、Q31、Q15 和 Q7。使用 FIR 滤波器函数前，确保已初始化相应的实例化结构体。

以下各节详细描述各个 FIR 滤波器函数。

riscv_dsp_fir_f32

原型：

void riscv_dsp_fir_f32 (const riscv_dsp_fir_f32_t *instance, float32_t *src, float32_t *dst, uint32_t size)

参数：

- riscv_dsp_fir_f32_t 浮点 FIR 滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t coeff_size;
  float32_t *state;
  float32_t *coeff;
} riscv_dsp_fir_f32_t;
```

其中，

  - coeff_size 系数的数量
  - *state 指向状态向量的指针，其大小为 coeff_size + size - 1
  - *coeff 指向时间逆系数向量的指针，其大小为 coeff_size
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

示例：

```
#define coeff_size 4
#define size 6
float32_t state[coeff_size + size - 1];
float32_t coeff[coeff_size] = {0.3, -0.1, 0.4, 0.5};
riscv_dsp_fir_f32_t inst = {coeff_size, state, coeff};
float32_t src[size] = {0.1, -0.2, 0.2, 0.3, 0.4, 0.1};
float32_t dst[size];
riscv_dsp_fir_f32(&inst, src, dst, size);
```

本示例同样适用于 Q31、fast_Q31、Q15、fast_Q15 或 Q7 类型的 FIR 滤波器函数。

riscv_dsp_fir_q31

原型：

```
void riscv_dsp_fir_q31 (const riscv_dsp_fir_q31_t *instance, q31_t *src, q31_t *dst, uint32_t size)
```

参数：

- riscv_dsp_fir_q31_t Q31 Q31 FIR 滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t coeff_size;
  q31_t *state;
  q31_t *coeff;
} riscv_dsp_fir_q31_t;
```

其中，

  - coeff_size 系数的数量
  - *state 指向状态向量的指针，其大小为 coeff_size + size - 1
  - *coeff 指向时间逆系数向量的指针，其大小为 coeff_size
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针

- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

注！

系数和状态变量都以 Q31 类型表示，乘法产生一个 Q2.62 结果，这些结果右移 31 位，饱和为 Q31 类型，保存到缓存区中。因此，为避免溢出，在调用此函数之前，必须将输入信号按 log2(coeff_size)位缩小。与 riscv_dsp_fir_fast_q31 相比，此函数精度更高，代码更精简。

riscv_dsp_fir_fast_q31

原型：

void riscv_dsp_fir_fast_q31 (const riscv_dsp_fir_q31_t *instance, q31_t *src, q31_t *dst, uint32_t size)

参数：

- riscv_dsp_fir_q31_t Q31 Q31 FIR 滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t coeff_size;
  q31_t *state;
  q31_t *coeff;
} riscv_dsp_fir_q31_t;
```

其中，

  - coeff_size 系数的数量
  - *state 指向状态向量的指针，其大小为 coeff_size + size - 1
  - *coeff 指向时间逆系数向量的指针，其大小为 coeff_size
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

注！

系数和状态变量都以 Q31 类型表示，乘法产生一个 Q1.31 结果，这些结果左移 1 位，保存

到缓存区中。因此，为避免溢出，在调用此函数之前，必须将输入信号按 log2(coeff_size) 位缩小。与 riscv_dsp_fir_q31 相比，此函数以较低的精度和较大的代码，换取更高的性能。

riscv_dsp_fir_q15

原型：

void riscv_dsp_fir_q15 (const riscv_dsp_fir_q15_t *instance, q15_t *src, q15_t *dst, uint32_t size)

参数：

- riscv_dsp_fir_q15_t Q15 FIR 滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t coeff_size;
  q15_t *state;
  q15_t *coeff;
} riscv_dsp_fir_q15_t;
```

其中，

  - coeff_size 系数的数量
  - *state 指向状态向量的指针，其大小为 coeff_size + size - 1
  - *coeff 指向时间逆系数向量的指针，其大小为 coeff_size
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

注！

系数和状态变量都以 Q15 类型表示，乘法产生一个 Q30 的结果，结果以 Q34.30 类型累积在 64 位累加器中，然后通过丢掉低 15 位截断为 Q34.15 类型，最后输出结果饱和为 Q1.15 类型。与 riscv_dsp_fir_fast_q15 相比，此函数精度更高，代码更精简。

riscv_dsp_fir_fast_q15

原型：

void riscv_dsp_fir_fast_q15 (const riscv_dsp_fir_q15_t *instance, q15_t *src, q15_t *dst, uint32_t size)

参数：

- riscv_dsp_fir_q15_t Q15 FIR 滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t coeff_size;
  q15_t *state;
  q15_t *coeff;
} riscv_dsp_fir_q15_t;
```

其中，

- coeff_size 系数的数量
- *state 指向状态向量的指针，其大小为 coeff_size + size - 1
- *coeff 指向时间逆系数向量的指针，其大小为 coeff_size

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

注！

系数和状态变量都以 Q15 类型表示，乘法产生一个 Q30 结果，结果以 Q2.30 类型累积在 32 位累加器中，然后饱和为 Q1.15 类型。与 riscv_dsp_fir_q15 相比，此函数以较低的精度和较大的代码，换取更高的性能。

riscv_dsp_fir_q7

原型：

```
void riscv_dsp_fir_q7 (const riscv_dsp_fir_q7_t *instance, q7_t *src, q7_t *dst, uint32_t size)
```

参数：

- riscv_dsp_fir_q7_t Q7 FIR 滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t coeff_size;
```

```
    q7_t *state;
    q7_t *coeff;
} riscv_dsp_fir_q7_t;
```

其中，

- coeff_size 系数的数量
- *state 指向状态向量的指针，其大小为 coeff_size + size - 1.
- *coeff 指向时间逆系数向量的指针，其大小为 coeff_size

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

注！

两个输入都是 Q7 类型，乘法产生一个 Q14 结果。中间结果按 Q18.14 类型在 32 位累加器中累加，然后转换为 Q18.7 类型，丢掉低 7 位，饱和为 Q1.7 类型。

3.4.7 FIR 抽取滤波器函数

FIR 抽取滤波器函数，实现了抽取一个整数因子 M 的 FIR 滤波器，等式如下所示：

$$\mathrm{dst[n]} = \sum_{k=0}^{Size} (\mathrm{b[k]} \times src[n-k])$$

其中，b[k]是滤波器系数，src 是长度为 Size 的输入数据，dst 是长度为 Size/M 的输出数据。Size 必须是抽取因子 M 的倍数，以保证输出数据的长度为整数。

DSP 软件编程函数库支持以下不同数据类型的 FIR 抽取滤波器函数和实例化结构体：浮点型、Q31 和 Q15。使用 FIR 抽取滤波器函数前，确保已初始化相应的实例化结构体。

以下各节详细描述各个 FIR 抽取滤波器函数。

riscv_dsp_dcmfir_f32

原型：

void riscv_dsp_dcmfir_f32 (const riscv_dsp_dcmfir_f32_t *instance, float32_t *src, float32_t *dst, uint32_t size)

参数：

- riscv_dsp_dcmfir_f32_t 浮点抽取 FIR 滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t M;
  uint32_t coeff_size;
  float32_t *coeff;
  float32_t *state;
} riscv_dsp_dcmfir_f32_t;
```

其中，

  - M 降采样因子
  - coeff_size 滤波器系数的数量
  - *coeff 指向时间逆系数向量的指针，其大小为 coeff_size
  - *state 指向状态向量的指针，其大小为 coeff_size + size - 1
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

示例：

输入数据大小为 24，抽取因子 M 为 4，滤波器系统大小为 6，FIR 抽取滤波器设置如下所示：

```
#define size 24
#define M 4
#define coeff_size 6
float32_t coeff[coeff_size] = {…};
float32_t state[coeff_size + size - 1];
riscv_dsp_dcmfir_f32_t inst = {M, coeff_size, coeff, state};
float32_t src[size] = {0.1, -0.2, 0.2, 0.3, 0.4, 0.1,  …};
float32_t dst[size / M];
riscv_dsp_dcmfir_f32(&inst, src, dst, size);
```

本示例同样适用于 Q31 或 Q15 类型的 FIR 抽取滤波器函数。

riscv_dsp_dcmfir_q31

原型：

void riscv_dsp_dcmfir_q31 (const riscv_dsp_dcmfir_q31_t *instance, q31_t *src, q31_t *dst, uint32_t size)

参数：

- riscv_dsp_dcmfir_q31_t Q31 抽取 FIR 滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t M;
  uint32_t coeff_size;
  q31_t *coeff;
  q31_t *state;
} riscv_dsp_dcmfir_q31_t;
```

其中，

  - M 降采样因子
  - coeff_size 滤波器系数的数量
  - *coeff 指向时间逆系数向量的指针，其大小为 coeff_size
  - *state 指向状态向量的指针，其大小为 coeff_size + size - 1
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

注！

系数和状态变量都是 Q31 类型，64 位累加器用于执行带有饱和度的乘法累加操作，产生 Q1.62 类型的结果。所有操作完成后，结果饱和为 Q31 类型。

riscv_dsp_dcmfir_fast_q31

原型：

void riscv_dsp_dcmfir_fast_q31 (const riscv_dsp_dcmfir_q31_t

*instance, q31_t *src, q31_t *dst, uint32_t size)

参数：

- riscv_dsp_dcmfir_q31_t Q31 抽取 FIR 滤波器的实例化结构体，定义如下：

  typedef struct
  {
  uint32_t M;
  uint32_t coeff_size;
  q31_t *coeff;
  q31_t *state;
  } riscv_dsp_dcmfir_q31_t;

  其中，

  - M 降采样因子
  - coeff_size 滤波器系数的数量
  - *coeff 指向时间逆系数向量的指针，其大小为 coeff_size
  - *state 指向状态向量的指针，其大小为 coeff_size + size - 1
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

注！

- 系数和状态变量都以 Q31 类型表示，32 位累加器用于执行带有饱和度的乘法累加操作，并截断为 Q2.30 类型的结果。为避免导致结果失真的溢出，输入信号必须按 log2(coeff_size)位的比例缩小。所有操作完成后，结果饱和为 Q1.31 类型。
- 与 riscv_dsp_dcmfir_q31 函数相比，此函数以较低的精度，换取更高的性能。

riscv_dsp_dcmfir_q15

原型：

void riscv_dsp_dcmfir_q15 (const riscv_dsp_dcmfir_q15_t *instance, q15_t *src, q15_t *dst, uint32_t size)

参数：

- riscv_dsp_dcmfir_q15_t Q15 抽取 FIR 滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t M;
  uint32_t coeff_size;
  q15_t *coeff;
  q15_t *state;
} riscv_dsp_dcmfir_q15_t;
```

其中，

  - M 降采样因子
  - coeff_size 滤波器系数的数量
  - *coeff 指向时间逆系数向量的指针，其大小为 coeff_size
  - *state 指向状态向量的指针，其大小为 coeff_size + size - 1
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

注！

系数和状态变量都以 Q15 类型表示，64 位累加器用于带有饱和度的执行乘法累加操作，以产生 Q33.30 的结果。所有操作完成后，结果饱和为 Q15 类型。

riscv_dsp_dcmfir_fast_q15

原型：

```
void riscv_dsp_dcmfir_fast_q15 (const riscv_dsp_dcmfir_q15_t *instance, q15_t *src, q15_t *dst, uint32_t size)
```

参数：

- riscv_dsp_dcmfir_q15_t Q15 抽取 FIR 滤波器的实例化结构体，定义如下：

```
typedef struct
{
```

```
    uint32_t M;
    uint32_t coeff_size;
    q15_t *coeff;
    q15_t *state;
} riscv_dsp_dcmfir_q15_t;
```

其中，

- M 降采样因子
- coeff_size 滤波器系数的数量
- *coeff 指向时间逆系数向量的指针，其大小为 coeff_size
- *state 指向状态向量的指针，其大小为 coeff_size + size - 1

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

注！

- 系数和状态都是 Q15 类型表示，一个 32 位累加器用于执行带有饱和度的乘法累加操作，以产生 Q2.30 的结果。如果累加器的结果溢出，则回绕失真结果。为完全避免溢出，输入信号必须按 log2(coeff_size)位缩小。在执行所有操作之后，Q2.30 累加器被截断为 Q2.15 类型，饱和为 Q1.15 结果。
- 与 riscv_dsp_dcmfir_q15 相比，此函数具有更高的性能。

3.4.8 FIR 格型滤波器函数

FIR 格型滤波器函数，等式如下所示：

$y_0[n] = u_0[n] = x[n]$

$y_z[n] = y_{z-1}[n] + k_z u_{z-1}[n-1)$

$u_z[n] = k_z y_{z-1}[n] + u_{z-1}[n-1]$

其中，x 是输入，y 是输出，u 是先前输入的状态，$0 \leq z < M$，M 是级数，k_z 是反射系数。

DSP 软件编程函数库支持以下不同数据类型的 FIR 格型滤波器函数和实例化结构体：浮点型、Q31 和 Q15。使用 FIR 格型滤波器函数前，确保已初始化相应的实例化结构体。

以下各节详细描述各个 FIR 格型滤波器函数。

riscv_dsp_lfir_f32

原型：

void riscv_dsp_lfir_f32 (const riscv_dsp_lfir_f32_t *instance, float32_t *src, float32_t *dst, uint32_t size)

参数：

- riscv_dsp_lfir_f32_t 浮点 FIR 格型滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t stage;
  float32_t *state;
  float32_t *coeff;
} riscv_dsp_lfir_f32_t;
```

其中，

  - stage 过滤级数
  - *state 指向状态向量的指针，其大小为 stage
  - *coeff 指向系数向量的指针，其大小为 stage
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

示例：

滤波器长度为 6，输入数据大小为 8，FIR 格型滤波器设置如下所示：

```
#define M 6
#define size 8
float32_t state[M];
float32_t coeff[M] = {…};
float32_t src[size] = {…};
float32_t dst[size];
riscv_dsp_lfir_f32_t inst = {M, state, coeff};
riscv_dsp_lfir_f32 (&inst, src, dst, size);
```

本示例同样适用于 Q31 或 Q15 类型的 FIR 格型滤波器函数。

riscv_dsp_lfir_q31

原型：

void riscv_dsp_lfir_q31 (const riscv_dsp_lfir_q31_t *instance, q31_t *src, q31_t *dst, uint32_t size)

参数：

- riscv_dsp_lfir_q31_t Q31 FIR 格型滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t stage;
  q31_t *state;
  q31_t *coeff;
} riscv_dsp_lfir_q31_t;
```

其中，

  - stage 过滤级数
  - *state 指向状态向量的指针，其大小为 stage
  - *coeff 指向系数向量的指针，其大小为 stage
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

riscv_dsp_lfir_q15

原型：

void riscv_dsp_lfir_q15 (const riscv_dsp_lfir_q15_t *instance, q15_t *src, q15_t *dst, uint32_t size)

参数：

- riscv_dsp_lfir_q15_t Q15 FIR 格型滤波器的实例化结构体，定义如下：

```
typedef struct
```

```
{
    uint32_t stage;
    q15_t *state;
    q15_t *coeff;
} riscv_dsp_lfir_q15_t;
```

其中，

- stage 过滤级数
- *state 指向状态向量的指针，其大小为 stage
- *coeff 指向系数向量的指针，其大小为 stage

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

3.4.9 IIR 格型滤波器函数

IIR 格型滤波器函数等式如下所示：

$y_0[n] = x[n]$

$y_{z-1}[n] = yz[n] - k_z * u_{z-1}[n-1]$

$u_z[n] = k_z * y_{z-1}[n] + u_{z-1}[n-1]$

and $w[n] = v_N * u_N[n] + v_{N-1} * u_{N-1}[n] + ...+ v_0 * u_0[n]$

其中，N 是状态数量，z = N, N-1, ... , 1，系数 k = {r_N, r_{N-1}, ...r_1}，v_N 是阶梯系数，u 是状态。

DSP 软件编程函数库支持以下不同数据类型的 IIR 格型滤波器函数和实例化结构体：浮点型、Q31 和 Q15。使用 IIR 格型滤波器函数前，确保已初始化相应的实例化结构体。

以下各节详细描述各个 IIR 格型滤波器函数。

riscv_dsp_liir_f32

原型：

```
void riscv_dsp_liir_f32 (const riscv_dsp_liir_f32_t *instance, float32_t *src, float32_t *dst, uint32_t size)
```

参数：

- riscv_dsp_liir_f32_t 浮点 IIR 格型滤波器的实例化结构体，定义如下：

```
typedef struct
{
   uint32_t nstage;
   float32_t *state;
   float32_t *rcoeff;
   float32_t *lcoeff;
} riscv_dsp_liir_f32_t;
```

其中，

  - nstage 过滤级数
  - *state 指向状态向量的指针，其大小为 nstage + size
  - *rcoeff 指向时间反向反射系数向量的指针，其大小为 nstage
  - *lcoeff 指向时间反转阶梯系数向量的指针，其大小为 nstage + 1
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

示例：

```
#define size 8
#define nstage 6
float32_t state[nstage + size];
float32_t rcoeff[nstage] = {…};
float32_t lcoeff[nstage + 1] = {…};
riscv_dsp_liir_f32_t inst = {nstage, state, rcoeff, lcoeff}.
float32_t src[size] = {…};
float32_t dst[size];
riscv_dsp_liir_f32(&inst, src, dst, size);
```

本示例同样适用于 Q31、fast_Q31、Q15 或 fast_Q15 类型的 IIR 格型滤波器函数。

riscv_dsp_liir_q31

原型：

void riscv_dsp_liir_q31 (const riscv_dsp_liir_q31_t *instance, q31_t *src, q31_t *dst, uint32_t size)

参数：

- riscv_dsp_liir_q31_t Q31 IIR 格型滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t nstage;
  q31_t *state;
  q31_t *rcoeff;
  q31_t *lcoeff;
} riscv_dsp_liir_q31_t;
```

其中，

  - nstage 过滤级数
  - *state 指向状态向量的指针，其大小为 nstage + size
  - *rcoeff 指向时间反向反射系数向量的指针，其大小为 nstage
  - *lcoeff 指向时间反转阶梯系数向量的指针，其大小为 nstage + 1
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

riscv_dsp_liir_fast_q31

原型：

void riscv_dsp_liir_fast_q31 (const riscv_dsp_liir_q31_t *instance, q31_t *src, q31_t *dst, uint32_t size)

参数：

- riscv_dsp_liir_q31_t Q31 IIR 格型滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t nstage;
  q31_t *state;
  q31_t *rcoeff;
  q31_t *lcoeff;
} riscv_dsp_liir_q31_t;
```

其中，

  - nstage 过滤级数
  - *state 指向状态向量的指针，其大小为 nstage + size
  - *rcoeff 指向时间反向反射系数向量的指针，其大小为 nstage
  - *lcoeff 指向时间反转阶梯系数向量的指针，其大小为 nstage + 1
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量，其大小必须为 2 的倍数

返回值：

无。

注！

为避免溢出，反射系数必须按比例减小 2。

riscv_dsp_liir_q15

原型：

```
void riscv_dsp_ liir_q15 (const riscv_dsp_liir_q15_t *instance, q15_t *src, q15_t *dst, uint32_t size)
```

参数：

- riscv_dsp_liir_q15_t Q15 IIR 格型滤波器的实例化结构体，定义如下：

```
typedef struct
```

```
{
  uint32_t nstage;
  q15_t *state;
  q15_t *rcoeff;
  q15_t *lcoeff;
} riscv_dsp_liir_q15_t;
```

其中，

- nstage 过滤级数
- *state 指向状态向量的指针，其大小为 nstage + size
- *rcoeff 指向时间反向反射系数向量的指针，其大小为 nstage
- *lcoeff 指向时间反转阶梯系数向量的指针，其大小为 nstage + 1

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

riscv_dsp_liir_fast_q15

原型：

```
void riscv_dsp_liir_fast_q15 (const riscv_dsp_liir_q15_t *instance, q15_t *src, q15_t *dst, uint32_t size)
```

参数：

- riscv_dsp_liir_q15_t Q15 IIR 格型滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t nstage;
  q15_t *state;
  q15_t *rcoeff;
  q15_t *lcoeff;
} riscv_dsp_liir_q15_t;
```

其中，

- nstage 过滤级数
- *state 指向状态向量的指针，其大小为 nstage + size
- *rcoeff 指向时间反向反射系数向量的指针，其大小为 nstage
- *lcoeff 指向时间反转阶梯系数向量的指针，其大小为 nstage + 1

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量，其大小必须为 2 的倍数

返回值：

无。

3.4.10 LMS 滤波器函数

LMS（Least mean square）滤波器函数为自适应滤波器，执行以下等式来更新系统，获取最小均方误差：

y[n] = h[0] * x[n] + h[1] * x [n - 1] + ... + h[L - 1] * y[n - L + 1];

e[n] = d[n] - y[n];

h[k] = h[k - 1] + μ * e[n] * x[n - k];

其中：

- y 输出向量
- h 系数向量
- x 输入向量
- L 滤波器阶数
- e 误差向量
- d 期望向量
- μ 可以调整系数的步长

DSP 软件编程函数库支持以下不同数据类型的 LMS 滤波器函数和实例化结构体：浮点型、Q31 和 Q15。使用 LMS 滤波器函数前，确保已初始化相应的实例化结构体。

以下各节详细描述各个 LMS 滤波器函数。

riscv_dsp_lms_f32

原型：

void riscv_dsp_lms_f32 (const riscv_dsp_lms_f32_t *instance, float32_t *src, float32_t *ref, float32_t *dst, float32_t *err, uint32_t size)

参数：

- riscv_dsp_lms_f32_t 浮点 LMS 滤波器的实例化结构体，定义如下：

  typedef struct
  {
  uint32_t coeff_size;
  float32_t *state;
  float32_t *coeff;
  float32_t mu;
  } riscv_dsp_lms_f32_t;

  其中，

  - coeff_size 滤波器系数的数量
  - *state 指向状态向量的指针，其大小为 coeff_size + size - 1
  - *coeff 指向时间逆系数向量的指针，其大小为 coeff_size
  - mu 可以调整系数的步长
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [in] *ref 指向期望向量的指针
- [out] *dst 指向输出向量的指针
- [out] *err 指向误差向量的指针
- [in] size 样本数量

返回值：

无。

示例：

```
#define coeff_size 5
#define size 6
#define mu 0.05
float32_t state[coeff_size + size - 1] = {0.0};
float32_t coeff[coeff_size] = {0.40, 0.10, 0.24, -0.40, -0.34};
float32_t src[size] = {1.0, 0.5, 0.4, -0.1, -0.1, 0.3};
```

```
float32_t ref[size] = {0.1, 0.2, -0.1, -0.02, -0.1, 0.2};
float32_t err[size];
float32_t dst[size];
riscv_dsp_lms_f32_t instance = {coeff_size, state, coeff, mu};
riscv_dsp_lms_f32(&instance, src, ref, dst, err, size);
```

本示例同样适用于 Q31 和 Q15 类型的 LMS 滤波器函数。

riscv_dsp_lms_q31

原型：

```
void riscv_dsp_lms_q31 (const riscv_dsp_lms_q31_t *instance, q31_t *src, q31_t *ref, q31_t *dst, q31_t *err, uint32_t size)
```

参数：

- riscv_dsp_lms_q31_t Q31 LMS 滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t coeff_size;
  q31_t *state;
  q31_t *coeff;
  q31_t mu;
  q31_t shift;
} riscv_dsp_lms_q31_t;
```

其中，

  - coeff_size 滤波器系数的数量
  - *state 指向状态向量的指针，其大小为 coeff_size + size - 1
  - *coeff 指向时间逆系数向量的指针，其大小为 coeff_size
  - mu 可以调整系数的步长
  - shift 要移位的系数位数
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [in] *ref 指向期望向量的指针
- [out] *dst 指向输出向量的指针
- [out] *err 指向误差向量的指针

- [in] size 样本数量

返回值：

无。

riscv_dsp_lms_q15

原型：

void riscv_dsp_lms_q15 (const riscv_dsp_lms_q15_t *instance, q15_t *src, q15_t *ref, q15_t *dst, q15_t *err, uint32_t size)

参数：

- riscv_dsp_lms_q15_t Q15 LMS 滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t coeff_size;
  q15_t *state;
  q15_t *coeff;
  q15_t mu;
  q15_t shift;
} riscv_dsp_lms_q15_t;
```

其中，

  - coeff_size 滤波器系数的数量
  - *state 指向状态向量的指针，其大小为 coeff_size + size - 1
  - *coeff 指向时间逆系数向量的指针，其大小为 coeff_size
  - mu 可以调整系数的步长
  - shift 要移位的系数位数
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [in] *ref 指向期望向量的指针
- [out] *dst 指向输出向量的指针
- [out] *err 指向误差向量的指针
- [in] size 样本数量

返回值：

无。

3.4.11 NLMS 滤波器函数

NLMS（Normalized Least Mean Square）滤波器函数为 LMS 滤波器函数的扩展，等式如下所示：

y[n] = h[0] * x[n] + h[1] * x [n - 1] + ... + h[L - 1] * y[n - L + 1];

e[n] = d[n] - y[n];

μ[n] = 1/ X[n]2;

h[n] = h[h -1] + μ[n]*e[n]*x[n];

其中，

- y 输出向量
- h 系数向量
- x 输入向量
- L 滤波器阶数
- e 误差向量
- d 期望向量
- μ 可以调整系数的步长，与归一化输入信号有关

DSP 软件编程函数库支持以下不同数据类型的 NLMS 滤波器函数和实例化结构体：浮点型、Q31 和 Q15。使用 NLMS 滤波器函数前，确保已初始化相应的实例化结构体。

以下各节详细描述各个 NLMS 滤波器函数。

riscv_dsp_nlms_f32

原型：

void riscv_dsp_nlms_f32 (riscv_dsp_nlms_f32_t *instance, float32_t *src, float32_t *ref, float32_t *dst, float32_t *err, uint32_t size)

参数：

- riscv_dsp_nlms_f32_t 浮点 NLMS 滤波器的实例化结构体，定义如下：

```
typedef struct
{
   uint32_t coeff_size;
   float32_t *state;
```

```
    float32_t *coeff;
    float32_t mu;
    float32_t energy
} riscv_dsp_nlms_f32_t;
```

其中，

- coeff_size 滤波器系数的数量
- *state 指向状态向量的指针，其大小为 coeff_size + size - 1
- *coeff 指向时间逆系数向量的指针，其大小为 coeff_size
- mu 可以调整系数的步长
- energy 前一坐标系的能量

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [in] *ref 指向期望向量的指针
- [out] *dst 指向输出向量的指针
- [out] *err 指向误差向量的指针
- [in] size 样本数量

返回值：

无。

示例：

```
#define coeff_size 5
#define size 6
#define mu 0.05
#define energy 0.1
float32_t state[coeff_size + size - 1] = {0.0};
float32_t coeff[coeff_size] = {0.40, 0.10, 0.24, -0.40, -0.34};
float32_t src[size] = {1.0, 0.5, 0.4, -0.1, -0.1, 0.3};
float32_t ref[size] = {0.1, 0.2, -0.1, -0.02, -0.1, 0.2};
float32_t err[size];
float32_t dst[size];
riscv_dsp_nlms_f32_t instance = {coeff_size, state, coeff, mu,
energy};
riscv_dsp_nlms_f32(&instance, src, ref, dst, err, size);
```

本示例同样适用于 Q31 和 Q15 类型的 NLMS 滤波器函数。

riscv_dsp_nlms_q31

原型：

void riscv_dsp_nlms_q31(riscv_dsp_nlms_q31_t *instance, q31_t *src, q31_t *ref, q31_t *dst, q31_t *err, uint32_t size);

参数：

- riscv_dsp_nlms_q31_t Q31 NLMS 滤波器的实例化结构体，定义如下：

```
typedef struct
{
   uint32_t coeff_size;
   q31_t *state;
   q31_t *coeff;
   q31_t mu;
   uint8_t postshift;
   q31_t energy
   q31_t x0;
} riscv_dsp_nlms_q31_t;
```

  其中，
  - coeff_size 滤波器系数的数量
  - *state 指向状态向量的指针，其大小为 coeff_size + size - 1
  - *coeff 指向时间逆系数向量的指针，其大小为 coeff_size
  - mu 可以调整系数的步长
  - postshift 要移位的系数位数
  - energy 前一坐标系的能量
  - x0 之前的输入样本
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [in] *ref 指向期望向量的指针
- [out] *dst 指向输出向量的指针
- [out] *err 指向误差向量的指针
- [in] size 样本数量

返回值：

无。

riscv_dsp_nlms_q15

原型：

void riscv_dsp_nlms_q15(riscv_dsp_nlms_q15_t *instance, q15_t *src, q15_t *ref, q15_t *dst, q15_t *err, uint32_t size);

参数：

- riscv_dsp_nlms_q15_t Q15 NLMS 滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t coeff_size;
  q15_t *state;
  q15_t *coeff;
  q15_t mu;
  uint8_t postshift;
  q15_t energy
  q15_t x0;
} riscv_dsp_nlms_q15_t;
```

  其中，
  - coeff_size 滤波器系数的数量
  - *state 指向状态向量的指针，其大小为 coeff_size + size - 1
  - *coeff 指向时间逆系数向量的指针，其大小为 coeff_size
  - mu 可以调整系数的步长
  - postshift 要移位的系数位数
  - energy 前一坐标系的能量
  - x0 之前的输入样本
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [in] *ref 指向期望向量的指针
- [out] *dst 指向输出向量的指针

- [out] *err 指向误差向量的指针
- [in] size 样本数量

返回值：

无。

3.4.12 稀疏 FIR 滤波器函数

稀疏 FIR 滤波器函数为大多数系数为零的 FIR 滤波器，由于这一特性，稀疏 FIR 滤波器只保留非零系数，忽略信号与零系数的乘法运算。

DSP 软件编程函数库支持以下不同数据类型的稀疏 FIR 滤波器函数和实例化结构体：浮点型、Q31、Q15 和 Q7。使用稀疏 FIR 滤波器函数前，确保已初始化相应的实例化结构体。

以下各节详细描述各个稀疏 FIR 滤波器函数。

riscv_dsp_spafir_f32

原型：

void riscv_dsp_spafir_f32 (riscv_dsp_spafir_f32_t *instance, float32_t *src, float32_t *dst, float32_t *buf, uint32_t size)

参数：

- riscv_dsp_spafir_f32_t 浮点稀疏 FIR 滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint16_t coeff_size;
  uint16_t index;
  float32_t *state;
  float32_t *coeff;
  uint16_t delay;
  int32_t *nezdelay;
} riscv_dsp_spafir_f32_t;
```

其中，

  - coeff_size 滤波器系数的数量
  - index 状态缓存区的索引
  - *state 指向状态向量的指针，其大小为 delay + size

- *coeff 指向系数向量的指针，其大小为 coeff_size
- delay nezdelay 向量的最大值
- *nezdelay 存储非零系数的向量，其大小为 coeff_size

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] *buf 指向临时向量的指针
- [in] size 样本数量

返回值：

无。

注！

临时缓存区的大小应等于输入向量的大小。

示例：

以下示例，可用以下的公式计算：

y(n)=coeff(0)*x(n - nezdelay(0)) + coeff(1)*x(n - nezdelay(1)) + coeff(2)*x(n - nezdelay(2)) + coeff(3)*x(n - nezdelay(3))

```
#define coeff_size 4
#define index 1
#define size 6
int32_t nezdelay[coeff_size] = {3, 0, 2, 1}
#define delay 3
float32_t state[delay + size] = {0.0};
float32_t coeff[coeff_size] = {0.40, 0.10, 0.24, -0.40};
float32_t src[size] = {1.0, 0.5, 0.4, -0.1, -0.1, 0.3};
float32_t buf[size];
float32_t dst[size];
riscv_dsp_spafir_f32_t instance = {coeff_size, index, state, coeff,
delay, nezdelay};
riscv_dsp_spafir_f32(&instance, src, dst, buf, size);
```

本示例同样适用于 Q31、Q15 和 Q7 类型的稀疏 FIR 滤波器。

riscv_dsp_spafir_q31

原型：

void riscv_dsp_spafir_q31 (riscv_dsp_spafir_q31_t *instance, q31_t *src, q31_t *dst, q31_t *buf, uint32_t size)

参数：

- riscv_dsp_spafir_q31_t Q31 稀疏 FIR 滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint16_t coeff_size;
  uint16_t index;
  q31_t *state;
  q31_t *coeff;
  uint16_t delay;
  int32_t *nezdelay;
} riscv_dsp_spafir_q31_t;
```

  其中，
  - coeff_size 滤波器系数的数量
  - index 状态缓存区的索引
  - *state 指向状态向量的指针，其大小为 delay + size
  - *coeff 指向系数向量的指针，其大小为 coeff_size
  - delay nezdelay 向量的最大值
  - *nezdelay 存储非零系数的向量，其大小为 coeff_size
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] *buf 指向临时向量的指针
- [in] size 样本数量

返回值：

无。

注！

临时缓存区的大小应等于输入向量的大小。

riscv_dsp_spafir_q15

原型：

void riscv_dsp_spafir_q15 (riscv_dsp_spafir_q15_t *instance, q15_t *src, q15_t *dst, q15_t *buf1, q31_t *buf2, uint32_t size)

参数：

- riscv_dsp_spafir_q15_t Q15 稀疏 FIR 滤波器的实例化结构体，定义如下：

  ```
  typedef struct
  {
    uint16_t coeff_size;
    uint16_t index;
    q15_t *state;
    q15_t *coeff;
    uint16_t delay;
    int32_t *nezdelay;
  } riscv_dsp_spafir_q15_t;
  ```

  其中，
  - coeff_size 滤波器系数的数量
  - index 状态缓存区的索引
  - *state 指向状态向量的指针，其大小为 delay + size
  - *coeff 指向系数向量的指针，其大小为 coeff_size
  - delay nezdelay 向量的最大值
  - *nezdelay 存储非零系数的向量，其大小为 coeff_size
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] *buf1 指向临时缓存区 1 的指针
- [in] *buf2 指向临时缓存区 2 的指针
- [in] size 样本数量

返回值：

无。

注！

临时缓存区 1 和 2 都应等于输入向量的大小。

riscv_dsp_spafir_q7

原型：

void riscv_dsp_spafir_q7 (riscv_dsp_spafir_q7_t *instance, q7_t *src, q7_t *dst, q7_t *buf1, q31_t *buf2, uint32_t size)

参数：

- riscv_dsp_spafir_q7_t Q7 稀疏 FIR 滤波器的实例化结构体，定义如下：

  ```
  typedef struct
  {
    uint16_t coeff_size;
    uint16_t index;
    q7_t *state;
    q7_t *coeff;
    uint16_t delay;
    int32_t *nezdelay;
  } riscv_dsp_spafir_q7_t;
  ```

  其中，

  - coeff_size 滤波器系数的数量
  - index 状态缓存区的索引
  - *state 指向状态向量的指针，其大小为 delay + size
  - *coeff 指向系数向量的指针，其大小为 coeff_size
  - delay nezdelay 向量的最大值
  - *nezdelay 存储非零系数的向量，其大小为 coeff_size
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] *buf1 指向临时缓存区 1 的指针
- [in] *buf2 指向临时缓存区 2 的指针

- [in] size 样本数量

返回值：

无。

注！

临时缓存区 1 和 2 都应等于输入向量的大小。

3.4.13 上采样 FIR 滤波器函数

上采样 FIR 滤波器函数，用于上采样输入信号来产生多速率输出，通常应用于音频、声音或图像处理，例如，增加图像的分辨率或调整音频从 16kbps 到 32kbps 的输出采样率。

DSP 软件编程函数库支持以下不同数据类型的上采样 FIR 滤波器函数和实例化结构体：浮点型、Q31 和 Q15。使用上采样 FIR 滤波器函数前，确保已初始化相应的实例化结构体。

以下各节详细描述各个上采样 FIR 滤波器函数。

riscv_dsp_upsplfir_f32

原型：

void riscv_dsp_upsplfir_f32 (const riscv_dsp_upsplfir_f32_t *instance, float32_t *src, float32_t *dst, uint32_t size)

参数：

- riscv_dsp_upsplfir_f32_t 浮点上采样 FIR 滤波器的实例化结构体，定义如下：

```
typedef struct
{
   uint32_t L;
   uint32_t plen;
   float32_t *coeff;
   float32_t *state;
} riscv_dsp_upsplfir_f32_t;
```

其中，

- L 上采样滤波器的因数
- plen 系数的数量
- *coeff 指向时间逆系数向量的指针，其大小为 L * plen

- *state 指向状态向量的指针，其大小为 plen + size - 1

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

示例：

```
#define L 2
#define plen 4
#define size 6
float32_t state[plen + size - 1] = {0.0};
float32_t coeff[L * plen] = {0.40, 0.10, 0.24, -0.40, 1.0, 0.5, 0.4, -0.1};
float32_t src[size] = {1.0, 0.5, 0.4, -0.1, -0.1, 0.3};
float32_t dst[size * L];
riscv_dsp_upsplfir_f32_t instance = {L, plen, state, coeff};
riscv_dsp_upsplfir_f32(&instance, src, dst, size);
```

本示例同样适用于 Q31 和 Q15 类型的上采样 FIR 滤波器。

riscv_dsp_upsplfir_q31

原型：

```
void riscv_dsp_upsplfir_q31 (const riscv_dsp_upsplfir_q31_t *instance, q31_t *src, q31_t *dst, uint32_t size)
```

参数：

- riscv_dsp_upsplfir_q31_t Q31 上采样 FIR 滤波器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t L;
  uint32_t plen;
  q31_t *coeff;
```

```
    q31_t *state;
} riscv_dsp_upsplfir_q31_t;
```

其中，

- L 上采样滤波器的因数
- plen 系数的数量
- *coeff 指向时间逆系数向量的指针，其大小为 L * plen
- *state 指向状态向量的指针，其大小为 plen + size - 1

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

注！

系数和状态变量都以 1.31 类型表示，乘法产生一个 2.62 结果，2.62 结果在 64 位累加器中累加。所有操作完成后，2.62 累加器被截断为 1.32 类型，然后饱和为 1.31 类型。

riscv_dsp_upsplfir_q15

原型：

```
void riscv_dsp_upsplfir_q15 (const riscv_dsp_upsplfir_q15_t *instance, q15_t *src, q15_t *dst, uint32_t size)
```

参数：

- riscv_dsp_upsplfir_q15_t Q15 上采样 FIR 滤波器的实例化结构体，定义如下：

```
typedef struct
{
    uint32_t L;
    uint32_t plen;
    q15_t *coeff;
    q15_t *state;
} riscv_dsp_upsplfir_q15_t;
```

其中，

- L 上采样滤波器的因数
- plen 系数的数量
- *coeff 指向时间逆系数向量的指针，其大小为 L * plen
- *state 指向状态向量的指针，其大小为 plen + size - 1

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 样本数量

返回值：

无。

注！

系数和状态变量都以 1.15 类型表示，乘法产生一个 2.30 结果，2.30 结果以 34.30 格式累加在 64 位累加器中，丢掉低 15 位将结果截断为 34.15 类型，最后饱和为 1.15 类型。

3.5 矩阵函数

一个矩阵可以被看作是一个有 m 行 n 列和 m * n 个元素的矩形数组。

DSP 软件编程函数库中，为便于编程，矩阵以行为主顺序分别保存为向量，即向量的大小等于矩阵的个数（size = m * n）。

3.5.1 矩阵加法函数

矩阵加法函数，分别从两个源向量获取两个元素相加，结果依次写入目的向量，实现过程如下所示：

```
size = row * col;
for (i = 0; i < size; i++)
  dst[i] = src1[i] + src2[i];
```

DSP 软件编程函数库支持以下不同数据类型的矩阵加法函数：浮点型、Q31 和 Q15，以下各节详细描述各个矩阵加法函数。

riscv_dsp_mat_add_f32

原型：

```
void riscv_dsp_mat_add_f32 (const float32_t *src1, const float32_t *src2, float32_t *dst, uint32_t row, uint32_t col)
```

参数：

- [in] *src1 指向第一个输入矩阵的指针

- [in] *src2 指向第二个输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] row 矩阵的行数
- [in] col 矩阵的列数

返回值：

无。

示例：

以下等式展示两个矩阵的加法和结果。

$$\begin{matrix} 0.1 & 0.4 \\ -0.2 & 0.1 \end{matrix} + \begin{matrix} -0.2 & -0.1 \\ 0.3 & 0.5 \end{matrix} = \begin{matrix} -0.1 & 0.3 \\ 0.1 & 0.6 \end{matrix}$$

示例代码，如下所示：

```
#define row 2
#define col 2
float32_t src1[row * col] = {0.1, 0.4, -0.2, 0.1};
float32_t src2[row * col] = {-0.2, -0.1, 0.3, 0.5};
float32_t dst[row * col];
riscv_dsp_mat_add_f32(src1, src2, dst, row, col);
```

本示例同样适用于 Q32 或 Q15 类型的矩阵加法函数。

riscv_dsp_mat_add_q31

原型：

```
void riscv_dsp_mat_add_q31 (const q31_t *src1, const q31_t *src2, q31_t *dst, uint32_t row, uint32_t col)
```

参数：

- [in] *src1 指向第一个输入矩阵的指针
- [in] *src2 指向第二个输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] row 矩阵的行数
- [in] col 矩阵的列数

返回值：

无。

注！

结果饱和为 Q31 范围[0x80000000, 0x7FFFFFFF]。

riscv_dsp_mat_add_q15

原型：

void riscv_dsp_mat_add_q15 (const q15_t *src1, const q15_t *src2, q15_t *dst, uint32_t row, uint32_t col)

参数：

- [in] *src1 指向第一个输入矩阵的指针
- [in] *src2 指向第二个输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] row 矩阵的行数
- [in] col 矩阵的列数

返回值：

无。

注！

结果饱和为 Q15 范围[0x8000, 0x7FFF]。

3.5.2 矩阵逆函数

矩阵逆函数返回一个方阵的逆。方阵在水平方向和垂直方向上具有相同的维数，因此矩阵逆函数只接受一个参数（即 size）来指定行数和列数，这样不仅可以避免行和列接收两个不同的数字，而且还可以提高性能。

矩阵逆函数实现高斯-若尔当消元法来求矩阵的逆。首先用 0 和 1 填充输出矩阵，形成单位矩阵；然后对输入和输出矩阵同时进行初等行运算，使输入矩阵成为单位矩阵。如果原始的输入矩阵是可逆的，那么输出矩阵最终将是它的逆矩阵，并且返回 0。实现过程如下所示：

[A | I] → 初等行变换 → [I | A^{-1}]

其中，A 是输入矩阵，A^{-1} 是逆矩阵。I 是单位矩阵，产生于初等行变换之前的输出矩阵，存在于初等行变换后的输入矩阵中。

如果在初等行变换时主元值为零，则该函数需要从下面的行中寻找非零的主元，一旦找到，交换两行并再次执行初等行变换，直到输入矩阵转换为单位矩阵。如果非零的主元无效，则返回-1，表示输入矩阵是奇异的。

DSP 软件编程函数库支持以下不同数据类型的矩阵逆函数：单精度和双精度浮点型，以下各节详细描述各个矩阵逆函数。

riscv_dsp_mat_inv_f32

原型：

int32_t riscv_dsp_mat_inv_f32 (float32_t *src, float32_t *dst, uint32_t size)

参数：

- [in] *src 指向输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] size 矩阵的行/列数（即 size = rows = columns）

返回值：

0 成功

-1 失败

注！

- 如果返回值为 0，则输入矩阵是可逆的，变换为单位矩阵；如果返回值为-1，则输入矩阵为奇异矩阵；
- 无论返回何值，执行矩阵逆函数后，输入矩阵和输出矩阵的内容都会发生变化。

示例：

一个方阵 A，及其逆矩阵，如下所示：

$$A = \begin{matrix} 0.1 & 0.4 \\ -0.2 & -0.3 \end{matrix} \qquad inv(A) = \begin{matrix} -6 & -8 \\ 4 & 2 \end{matrix}$$

矩阵 A 的求逆示例代码，如下所示：

```
#define size 2
float32_t src[size * size] = {0.1, 0.4, -0.2, -0.3};
float32_t dst[size * size];
if (riscv_dsp_mat_inv_f32(src, dst, size) == 0)
Success…
else
Fail…
```

本示例同样适用于 F64 类型的矩阵逆函数。

riscv_dsp_mat_inv_f64

原型：

int32_t riscv_dsp_mat_inv_f64 (float64_t *src, float64_t *dst, uint32_t

size)

参数：

- [in] *src 指向输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] size 矩阵的行/列数（即 size = rows = columns）

返回值：

0 成功

-1 失败

3.5.3 矩阵乘法函数

矩阵乘法函数用于计算两个源矩阵的乘法，结果写入目的矩阵。以下展示矩阵 A 与矩阵 B 的乘法，其中两个矩阵都是 2 行 2 列。

$$\begin{matrix} A11 & A12 \\ A21 & A22 \end{matrix} \times \begin{matrix} B11 & B12 \\ B21 & B22 \end{matrix} =$$

$$\begin{matrix} A11 \times B11 + A12 \times B21 & A11 \times B12 + A12 \times B22 \\ A21 \times B11 + A22 \times B21 & A21 \times B12 + A22 \times B22 \end{matrix}$$

根据矩阵乘法的定义，矩阵 B 的行数必须与矩阵 A 的列数相同，即，

$C_{ik} = A_{ij} * B_{jk}$

其中，A 是 i 行 j 列的矩阵，B 是 j 行 k 列的矩阵，C 是 i 行 k 列的目的矩阵。

DSP 软件编程函数库支持以下不同数据类型的矩阵乘法函数：单精度浮点型、双精度浮点型、Q31、Q15 和 Q7，以下各节详细描述各个矩阵乘法函数。

riscv_dsp_mat_mul_f32

原型：

void riscv_dsp_mat_mul_f32 (const float32_t *src1, const float32_t *src2, float32_t *dst, uint32_t row, uint32_t col, uint32_t col2)

参数：

- [in] *src1 指向第一个输入矩阵的指针
- [in] *src2 指向第二个输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] row 第一个矩阵的行数
- [in] col 第一个矩阵的列数

- [in] col2 第二个矩阵的列数

返回值：

无。

示例：

以下示例展示两个矩阵的乘法：

$A_{2,3} \times B_{3,2} = C_{2,2}$

给定元素值，计算过程如下所示：

$$\begin{matrix} 0.1 & -0.1 & 0.1 \\ 0.2 & -0.2 & 0.3 \end{matrix} \times \begin{matrix} 0.2 & 0.2 \\ -0.1 & 0.3 \\ -0.7 & -0.2 \end{matrix} = \begin{matrix} -0.04 & -0.03 \\ -0.15 & -0.08 \end{matrix}$$

示例代码，如下所示：

```
#define Arow 2
#define Acol 3
#define Bcol 2
float32_t src1[Arow * Acol] = {0.1, -0.1, 0.1, 0.2, -0.2, 0.3};
float32_t src2[Acol * Bcol] = {0.2, 0.2, -0.1, 0.3, -0.7, -0.2};
float32_t dst[Arow * Bcol];
riscv_dsp_mat_mul_f32 (src1, src2, dst, Arow, Acol, Bcol);
```

本示例同样适用于 F64、Q31 或 Q15 类型的矩阵乘法函数。

riscv_dsp_mat_mul_f64

原型：

```
void riscv_dsp_mat_mul_f64 (const float64_t *src1, const float64_t *src2, float64_t *dst, uint32_t row, uint32_t col, uint32_t col2)
```

参数：

- [in] *src1 指向第一个输入矩阵的指针
- [in] *src2 指向第二个输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] row 第一个矩阵的行数
- [in] col 第一个矩阵的列数
- [in] col2 第二个矩阵的列数

返回值：

无。

riscv_dsp_mat_mul_q31

原型：

void riscv_dsp_mat_mul_q31 (const q31_t *src1, const q31_t *src2, q31_t *dst, uint32_t row, uint32_t col, uint32_t col2)

参数：

- [in] *src1 指向第一个输入矩阵的指针
- [in] *src2 指向第二个输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] row 第一个矩阵的行数
- [in] col 第一个矩阵的列数
- [in] col2 第二个矩阵的列数

返回值：

无。

riscv_dsp_mat_mul_fast_q31

原型：

void riscv_dsp_mat_mul_fast_q31 (const q31_t *src1, const q31_t *src2, q31_t *dst, uint32_t row, uint32_t col, uint32_t col2)

参数：

- [in] *src1 指向第一个输入矩阵的指针
- [in] *src2 指向第二个输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] row 第一个矩阵的行数
- [in] col 第一个矩阵的列数
- [in] col2 第二个矩阵的列数

返回值：

无。

注！

- 使用 32 位累加器进行乘法累加操作并截断为 Q2.30 结果。为避免导致结果失真的溢出，输入矩阵必须按 log2(row)位的比例缩小。所有操作完成后，结果饱和为 Q1.31 类型。

- 与 riscv_dsp_mat_mul_q31 相比，此函数以较低的精度，换取更高的性能。

riscv_dsp_mat_mul_q15

原型：

void riscv_dsp_mat_mul_q15 (const q15_t *src1, const q15_t *src2, q15_t *dst, uint32_t row, uint32_t col, uint32_t col2)

参数：

- [in] *src1 指向第一个输入矩阵的指针
- [in] *src2 指向第二个输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] row 第一个矩阵的行数
- [in] col 第一个矩阵的列数
- [in] col2 第二个矩阵的列数

返回值：

无。

注！

对于 RV64，此函数将分配一个 256 字节的临时缓存区，用于关键运算。

riscv_dsp_mat_mul_fast_q15

原型：

void riscv_dsp_mat_mul_fast_q15 (const q15_t *src1, const q15_t *src2, q15_t *dst, uint32_t row, uint32_t col, uint32_t col2)

参数：

- [in] *src1 指向第一个输入矩阵的指针
- [in] *src2 指向第二个输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] row 第一个矩阵的行数
- [in] col 第一个矩阵的列数
- [in] col2 第二个矩阵的列数

返回值：

无。

注！

- 对于 RV64，此函数将分配一个 128 字节的临时缓存区，用于关键运算；
- 与 riscv_dsp_mat_mul_q15 函数相比，此函数以较低的精度，换取更高的性能。

riscv_dsp_mat_mul_q7

原型：

void riscv_dsp_mat_mul_q7 (const q7_t *src1, const q7_t *src2, q7_t *dst, uint32_t row, uint32_t col, uint32_t col2)

参数：

- [in] *src1 指向第一个输入矩阵的指针
- [in] *src2 指向第二个输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] row 第一个矩阵的行数
- [in] col 第一个矩阵的列数
- [in] col2 第二个矩阵的列数

返回值：

无。

注！

对于 RV64，此函数将分配一个 512 字节的临时缓存区，用于关键运算；对于 RV32，分配 128 字节的临时缓存区。

riscv_dsp_mat_mul_vxm_q7

原型：

void riscv_dsp_mat_mul_vxm_q7 (const q7_t *src1, const q7_t *src2, q7_t *dst, uint32_t col, uint32_t col2)

参数：

- [in] *src1 指向输入向量的指针
- [in] *src2 指向输入矩阵的指针
- [out] *dst 指向输出向量的指针
- [in] col 输入向量的列数
- [in] col2 输入矩阵的列数

返回值：

无。

注！

此函数相乘一个 col 列的向量 src1[1, col]与一个 col 行和 col2 列的矩阵 src2[col, col2]，结果保存为一个 col2 列的向量 dst[1, col2]。

3.5.4 矩阵缩放函数

矩阵缩放函数，实现矩阵乘以缩放值，结果写入目的矩阵，实现过程如下所示：

```
size = m * n;
for (n = 0; n < size; n++)
  dst[i] = src[i] * scale;
```

在分数表示的情况下，引入参数 shift 来调整结果值的范围，关于该参数的详细介绍，具体参考分数数据类型函数 riscv_dsp_mat_scale_q31 和 riscv_dsp_mat_scale_q15。

DSP 软件编程函数库支持以下不同数据类型的矩阵缩放函数：浮点型、Q31 和 Q15，以下各节详细描述各个矩阵缩放函数。

riscv_dsp_mat_scale_f32

原型：

```
void riscv_dsp_mat_scale_f32 (const float32_t *src, float32_t scale, float32_t *dst, uint32_t row, uint32_t col)
```

参数：

- [in] *src 指向输入矩阵的指针
- [in] scale 恒定缩放值
- [out] *dst 指向输出矩阵的指针
- [in] row 矩阵的行数
- [in] col 矩阵的列数

返回值：

无。

示例：

给定一个均值和一个缩放值 0.2，矩阵缩放等式如下所示：

$$\begin{matrix} 0.1 & 0.4 \\ -0.2 & 0.1 \end{matrix} \times \text{scale} = \begin{matrix} 0.02 & 0.08 \\ -0.04 & 0.02 \end{matrix}$$

示例代码，如下所示：

```
#define row 2
```

```
#define col 2
float32_t src[row * col] = {0.1, 0.4, -0.2, 0.1};
flato32_t scale = 0.2;
float32_t dst[row * col];
riscv_dsp_mat_scale_f32 (src1, scale, dst, row, col);
```

本示例同样适用于 Q31 或 Q15 类型的矩阵缩放函数。

riscv_dsp_mat_scale_q31

原型：

```
void riscv_dsp_mat_scale_q31 (const q31_t *src, q31_t scale_fract, int32_t shift, q31_t *dst, uint32_t row, uint32_t col)
```

参数：

- [in] *src 指向输入矩阵的指针
- [in] scale_fract 恒定的分数缩放值
- [in] shift 移位位数
- [out] *dst 指向输出矩阵的指针
- [in] row 矩阵的行数
- [in] col 矩阵的列数

返回值：

无。

riscv_dsp_mat_scale_q15

原型：

```
void riscv_dsp_mat_scale_q15 (const q15_t *src, q15_t scale_fract, int32_t shift, q15_t *dst, uint32_t row, uint32_t col)
```

参数：

- [in] *src 指向输入矩阵的指针
- [in] scale_fract 恒定的分数缩放值
- [in] shift 移位位数
- [out] *dst 指向输出矩阵的指针
- [in] row 矩阵的行数
- [in] col 矩阵的列数

返回值：

无。

3.5.5 矩阵减法函数

矩阵减法函数，用于将两个维数相同的矩阵相减，结果写入目的矩阵，实现过程如下所示：

```
size = m * n;
for (i = 0; i < size; i++)
  dst[i] = src1[i] - src2[i];
```

DSP 软件编程函数库支持以下不同数据类型的矩阵减法函数：浮点型、Q31 和 Q15，以下各节详细描述各个矩阵减法函数。

riscv_dsp_mat_sub_f32

原型：

```
void riscv_dsp_mat_sub_f32 (const float32_t *src1, const float32_t *src2, float32_t *dst, uint32_t row, uint32_t col)
```

参数：

- [in] *src1 指向第一个输入矩阵的指针
- [in] *src2 指向第二个输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] row 矩阵的行数
- [in] col 矩阵的列数

返回值：

无。

示例：

给定两个矩阵，矩阵减法如下所示：

$$\begin{matrix} 0.1 & 0.4 \\ -0.2 & 0.1 \end{matrix} - \begin{matrix} -0.2 & -0.1 \\ 0.3 & 0.5 \end{matrix} = \begin{matrix} 0.3 & 0.5 \\ -0.5 & -0.4 \end{matrix}$$

矩阵减法的示例代码，如下所示：

```
#define row 2
#define col 2
float32_t src1[row * col] = {0.1, 0.4, -0.2, 0.1};
```

```
float32_t src2[row * col] = {-0.2, -0.1, 0.3, 0.5};
float32_t dst[row * col];
riscv_dsp_mat_sub_f32 (src1, src2, dst, row, col);
```

本示例同样适用于 Q31 或 Q15 类型的矩阵减法函数。

riscv_dsp_mat_sub_q31

原型：

```
void riscv_dsp_mat_sub_q31 (const q31_t *src1, const q31_t *src2, q31_t *dst, uint32_t row, uint32_t col)
```

参数：

- [in] *src1 指向第一个输入矩阵的指针
- [in] *src2 指向第二个输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] row 矩阵的行数
- [in] col 矩阵的列数

返回值：

无。

注！

结果饱和为 Q31 范围 [0x80000000, 0x7FFFFFFF]。

riscv_dsp_mat_sub_q15

原型：

```
void riscv_dsp_mat_sub_q15 (const q15_t *src1, const q15_t *src2, q15_t *dst, uint32_t row, uint32_t col)
```

参数：

- [in] *src1 指向第一个输入矩阵的指针
- [in] *src2 指向第二个输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] row 矩阵的行数
- [in] col 矩阵的列数

返回值：

无。

注！

结果饱和为 Q15 范围[0x8000, 0x7FFF]。

3.5.6 矩阵转置函数

矩阵转置函数，实现矩阵转置，结果写入目的矩阵，实现过程如下所示：

dst[n, m] = src[m, n], where 0 <= m < row, O <= n < col

DSP 软件编程函数库支持以下不同数据类型的矩阵转置函数：浮点型、Q31、Q15 和其他数据类型，以下各节详细描述各个矩阵转置函数。

riscv_dsp_mat_trans_f32

原型：

void riscv_dsp_mat_trans_f32 (const float32_t *src, float32_t *dst, uint32_t row, uint32_t col)

参数：

- [in] *src 指向输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] row 矩阵的行数
- [in] col 矩阵的列数

返回值：

无。

示例：

给定一个矩阵转置等式：

$$\begin{matrix} 0.1 & -0.1 & 0.1 \\ 0.2 & -0.2 & 0.3 \end{matrix}_{\mathrm{T}} = \begin{matrix} 0.1 & 0.2 \\ -0.1 & -0.2 \\ 0.1 & 0.3 \end{matrix}$$

示例代码，如下所示：

```
#define row 2
#define col 3
float32_t src[row * col] = {0.1, -0.1, 0.1, 0.2, -0.2, 0.3};
float32_t dst[col * row];
riscv_dsp_mat_trans_f32 (src, dst, row, col);
```

本示例同样适用于 Q31 或 Q15 类型的矩阵转置函数。

riscv_dsp_mat_trans_q31

原型：

void riscv_dsp_mat_trans_q31 (const q31_t *src, q31_t *dst, uint32_t row, uint32_t col)

参数：

- [in] *src 指向输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] row 矩阵的行数
- [in] col 矩阵的列数

返回值：

无。

riscv_dsp_mat_trans_q15

原型：

void riscv_dsp_mat_trans_q15 (const q15_t *src, q15_t *dst, uint32_t row, uint32_t col)

参数：

- [in] *src 指向输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] row 矩阵的行数
- [in] col 矩阵的列数

返回值：

无。

riscv_dsp_mat_trans_u8

原型：

void riscv_dsp_mat_trans_u8 (const uint8_t *src, uint8_t *dst, uint32_t row, uint32_t col)

参数：

- [in] *src 指向输入矩阵的指针

- [out] *dst 指向输出矩阵的指针
- [in] row 矩阵的行数
- [in] col 矩阵的列数

返回值：

无。

3.5.7 矩阵 2 次幂函数

矩阵 2 次幂函数，实现方阵 2 次方运算，结果写入目的矩阵，实现过程如下所示：

dst = src^2, where src is a square matrix

DSP 软件编程函数库支持以下不同数据类型的矩阵 2 次幂函数：双精度浮点型，以下各节详细描述各个矩阵 2 次幂函数。

riscv_dsp_mat_pwr2_cache_f64

原型：

int32_t riscv_dsp_mat_pwr2_cache_f64(const float64_t *src, float64_t *dst, uint32_t size)

参数：

- [in] *src 指向输入矩阵的指针
- [out] *dst 指向输出矩阵的指针
- [in] size 矩阵的行数或列数

返回值：

0 成功

-1 失败

注！

- 输入矩阵必须是一个方阵，其行数或列数等于 size，另外，size 必须是 4 的倍数（如 28、32、64、1024），否则，返回错误值-1。
- 此函数执行一个块矩阵乘法，块大小是一条 cache line 的大小，使用从内存中获取的每条 cache line，用于减少 cache 未命中。
- 当 size 小于 40 时，建议使用 riscv_dsp_mat_mul_f64 函数，以获得更好的性能。

3.5.8 矩阵外积函数

矩阵外积函数，用于计算两个源向量的乘法，结果写入目的矩阵，下面展示了向量 A 与向量 B 的乘法，其中向量 A 为 3 行，向量 B 为 4 列。

$$A11$$
$$A21 \times B11 \quad B12 \quad B13 \quad B14 =$$
$$A31$$

$$\begin{matrix} A11 \times B11 & A11 \times B12 & A11 \times B13 & A11 \times B14 \\ A21 \times B11 & A21 \times B12 & A21 \times B13 & A21 \times B11 \\ A31 \times B11 & A31 \times B12 & A31 \times B13 & A31 \times B14 \end{matrix}$$

外积可以看作为矩阵乘法，其中，源矩阵 A_{ij} 和 B_{jk} 有一个约束，即 j 的值必须为 1，如下所示。

$$C_{ik} = A_{ij} * B_{jk}, \text{ where } j = 1$$

DSP 软件编程函数库支持以下不同数据类型的矩阵外积函数：Q31，以下各节详细描述各个矩阵外积函数。

riscv_dsp_mat_oprod_q31

原型：

void riscv_dsp_mat_oprod_q31 (const q31_t *src1, const q31_t *src2, q31_t *dst, uint32_t size1, uint32_t size2)

参数：

- [in] *src1 指向第一个输入矩阵的指针，其大小为 size1*1
- [in] *src2 指向第二个输入矩阵的指针，其大小为 1*size2
- [out] *dst 指向输出矩阵的指针，其大小为 size1 * size2
- [in] size1 第一个输入矩阵的行数
- [in] size2 第二个输入矩阵的列数

返回值：

无。

注！

此函数相乘一个包含 size1 行的单列矩阵 src1[size1, 1]和一个包含 size2 列的单行矩阵 src2[1, size2]，结果保存到一个包含 size1 行和 size2 列的矩阵 dst[size1, size2]中。与常规矩阵乘法相比，此函数实现了更好的矢量矩阵乘法效率。

示例：

以下等式展示了两个矩阵的外积及其结果。

$$\begin{matrix} 0x200000 \\ 0x100000 \\ 0x50000 \end{matrix} + \begin{matrix} 0x10000 \end{matrix} \quad \begin{matrix} 0x30000 \end{matrix} = \begin{matrix} 0x40 & 0xc0 \\ 0x20 & 0x60 \\ 0xa & 0x1e \end{matrix}$$

示例代码，如下所示：

```
#define Arow 3
#define Bcol 2
q31_t src1[Arow] = {0x200000, 0x100000, 0x50000};
q31_t src2[Bcol] = {0x10000, 0x30000};
q31_t dst[Arow * Bcol];
riscv_dsp_mat_oprod_q31 (src1, src2, dst, Arow, Bcol);
```

3.5.9 复矩阵乘法函数

复矩阵乘法函数，用于计算两个源复矩阵的乘法，结果写入目的复矩阵。下面展示了复矩阵 A 与复矩阵 B 的乘法，其中两个矩阵都是 2 行 2 列。

$$\begin{matrix} A11 & A12 \\ A21 & A22 \end{matrix} \times \begin{matrix} B11 & B12 \\ B21 & B22 \end{matrix} =$$

$$\begin{matrix} A11 \times B11 + A12 \times B21 & A11 \times B12 + A12 \times B22 \\ A21 \times B11 + A22 \times B21 & A21 \times B12 + A22 \times B22 \end{matrix}$$

其中，每个乘法都是复乘法。

$A_n = a_n(re) = a_n(im)i$, $B_n = b_n(re) + b_n(im)i$,

$A_n \times B_n = (a_n(re) \times b_n(re) - a_n(im) \times b_n(im)) + (a_n(re) \times b_n(im) + a_n(im) \times b_n(re))i$

根据矩阵乘法的定义，复矩阵 B 的行数必须与复矩阵 A 的列数相同，即，

$C_{ik} = A_{ij} * B_{jk}$

其中，A 是 i 行 j 列的复矩阵，B 是 j 行 k 列的复矩阵，C 是 i 行 k 列的复矩阵。

DSP 软件编程函数库支持以下不同数据类型的复矩阵乘法函数：浮点型、Q31 和 Q15，以下各节详细描述各个复矩阵乘法函数。

riscv_dsp_cmat_mul_f32

原型：

```
void riscv_dsp_cmat_mul_f32 (const float32_t *src1, const float32_t *src2, float32_t *dst, uint32_t row, uint32_t col, uint32_t col2)
```

参数：

- [in] *src1 指向第一个复数矩阵的指针
- [in] *src2 指向第二个复数矩阵的指针
- [out] *dst 指向输出复数矩阵的指针

- [in] row 第一个复数矩阵的行数
- [in] col 第一个复数矩阵的列数
- [in] col2 第二个复数矩阵的列数

返回值：

无。

示例：

以下等式展示了两个复矩阵的乘法：

A2,3 x B3,2 = C2,2

给定每个元素值，计算过程如下：

$$\begin{matrix} 0.1+0.2i & -0.1-0.3i & 0.1+0x4i \\ 0.2-0.1i & -0.2+0.5i & 0.3-0.4i \end{matrix} \times \begin{matrix} 0.2+0.5i & 0.2-0.4i \\ -0.1+0.2i & 0.3+0.2i \\ -0.7-0.4i & -0.2-0.1i \end{matrix} =$$

$$\begin{matrix} 0.08-0.22i & 0.15-0.2i \\ -0.36+0.15i & -0.26+0.6i \end{matrix}$$

示例代码，如下所示：

```
#define Arow 2
#define Acol 3
#define Bcol 2
float32_t src1[2 * Arow * Acol] = {0.1, 0.2, -0.1, -0.3, 0.1, 0.4, 0.2, -0.1, -0.2, 0.5, 0.3, -0.4};
float32_t src2[2 * Acol * Bcol] = {0.2, 0.5, 0.2, -0.4, -0.1, 0.2, 0.3, 0.2, -0.7, -0.4, -0.2, -0.1};
float32_t dst[2 * Arow * Bcol];
riscv_dsp_cmat_mul_f32 (src1, src2, dst, Arow, Acol, Bcol);
```

本示例同样适用于 Q31 或 Q15 类型的复矩阵乘法函数。

riscv_dsp_cmat_mul_q31

原型：

```
void riscv_dsp_cmat_mul_q31 (const q31_t *src1, const q31_t *src2, q31_t *dst, uint32_t row, uint32_t col, uint32_t col2)
```

参数：

- [in] *src1 指向第一个复数矩阵的指针
- [in] *src2 指向第二个复数矩阵的指针

- [out] *dst 指向输出复数矩阵的指针
- [in] row 第一个复数矩阵的行数
- [in] col 第一个复数矩阵的列数
- [in] col2 第二个复数矩阵的列数

返回值：

无。

riscv_dsp_cmat_mul_q15

原型：

void riscv_dsp_cmat_mul_q15 (const q15_t *src1, const q15_t *src2, q15_t *dst, uint32_t row, uint32_t col, uint32_t col2)

参数：

- [in] *src1 指向第一个复数矩阵的指针
- [in] *src2 指向第二个复数矩阵的指针
- [out] *dst 指向输出复数矩阵的指针
- [in] row 第一个复数矩阵的行数
- [in] col 第一个复数矩阵的列数
- [in] col2 第二个复数矩阵的列数

返回值：

无。

3.6 统计函数

3.6.1 最大值函数

最大值函数，用于比较一个向量中的所有值，返回一个最大值及其下标。

DSP 软件编程函数库支持以下不同数据类型的最大值函数：浮点型、Q31、Q15、Q7 和其他数据类型，以下各节详细描述各个最大值函数。

riscv_dsp_max_f32

原型：

float32_t riscv_dsp_max_f32 (const float32_t *src, uint32_t size, uint32_t *index)

- 参数：
- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量
- [out] *index 最大值的索引

返回值：

最大值。

riscv_dsp_max_value_f32

原型：

float32_t riscv_dsp_max_value_f32 (const float32_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

最大值。

riscv_dsp_max_q31

原型：

q31_t riscv_dsp_max_q31 (const q31_t *src, uint32_t size, uint32_t *index)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量
- [out] *index 最大值的索引

返回值：

最大值。

riscv_dsp_max_q15

原型：

q15_t riscv_dsp_max_q15 (const q15_t *src, uint32_t size, uint32_t *index)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量
- [out] *index 最大值的索引

返回值：

最大值。

riscv_dsp_max_q7

原型：

q7_t riscv_dsp_max_q7 (const q7_t *src, uint32_t size, uint32_t *index)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量
- [out] *index 最大值的索引

返回值：

最大值。

riscv_dsp_max_u8

原型：

uint8_t riscv_dsp_max_u8 (const uint8_t *src, uint32_t size, uint32_t *index)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量
- [out] *index 最大值的索引

返回值：

最大值。

3.6.2 均值函数

均值函数，用于计算一个向量的算术平均值。

DSP 软件编程函数库支持以下不同数据类型的均值函数：浮点型、Q31、Q15、Q7 和其他数据类型，以下各节详细描述各个均值函数。

riscv_dsp_mean_f32

原型：

float32_t riscv_dsp_mean_f32 (const float32_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

平均值。

riscv_dsp_mean_q31

原型：

q31_t riscv_dsp_mean_q31 (const q31_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

平均值。

riscv_dsp_mean_q15

原型：

q15_t riscv_dsp_mean_q15 (const q15_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

平均值。

riscv_dsp_mean_q7

原型：

q7_t riscv_dsp_mean_q7 (const q7_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

平均值。

riscv_dsp_mean_u8

原型：

uint8_t riscv_dsp_mean_u8 (const uint8_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

平均值。

3.6.3 最小值函数

最小值函数，用于比较一个向量中的所有值，然后返回最小值及其下标。

DSP 软件编程函数库支持以下不同数据类型的最小值函数：浮点型、Q31、Q15、Q7 和其他数据类型，以下各节详细描述各个最小值函数。

riscv_dsp_min_f32

原型：

float32_t riscv_dsp_min_f32 (const float32_t *src, uint32_t size, uint32_t *index)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量
- [out] *index 最小值的索引

返回值：

最小值。

riscv_dsp_min_q31

原型：

q31_t riscv_dsp_min_q31 (const q31_t *src, uint32_t size, uint32_t *index)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量
- [out] *index 最小值的索引

返回值：

最小值。

riscv_dsp_min_q15

原型：

q15_t riscv_dsp_min_q15 (const q15_t *src, uint32_t size, uint32_t *index)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量
- [out] *index 最小值的索引

返回值：

最小值。

riscv_dsp_min_q7

原型：

q7_t riscv_dsp_min_q7 (const q7_t *src, uint32_t size, uint32_t *index)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量
- [out] *index 最小值的索引

返回值：

最小值。

riscv_dsp_min_u8

原型：

uint8_t riscv_dsp_min_u8 (const uint8_t *src, uint32_t size, uint32_t *index)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量
- [out] *index 最小值的索引

返回值：

最小值。

3.6.4 RMS 函数

RMS（Root mean square）函数，用于计算一个向量的均方根，实现过程如下所示：

$$RMS = sqrt((src[0]^2 + src[1]^2 + src[2]^2 + ... + src[size-1]^2) / size)$$

对于 sqrt()函数，具体参考 3.8.7 平方根函数。

DSP 软件编程函数库支持以下不同数据类型的 RMS 函数：浮点型、Q31 和 Q15，以下各节详细描述各个 RMS 函数。

riscv_dsp_rms_f32

原型：

float32_t riscv_dsp_rms_f32 (const float32_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

RMS 值。

riscv_dsp_rms_q31

原型：

q31_t riscv_dsp_rms_q31 (const q31_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

RMS 值。

riscv_dsp_rms_q15

原型：

q15_t riscv_dsp_rms_q15 (const q15_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

RMS 值。

3.6.5 幂函数

幂函数，用于计算一个向量的平方和，实现过程如下所示：

$SOS = src[0]^2 + src[1]^2 + src[2]^2 + ... + src[size-1]^2$

DSP 软件编程函数库支持以下不同数据类型的幂函数：浮点型、Q31、Q15 和 Q7，以下各节详细描述各个幂函数。

riscv_dsp_pwr_f32

原型：

float32_t riscv_dsp_pwr_f32 (const float32_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

平方和。

riscv_dsp_pwr_q31

原型：

q63_t riscv_dsp_pwr_q31 (const q31_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

平方和。

注！

返回值为 Q48 类型。

riscv_dsp_pwr_q15

原型：

q63_t riscv_dsp_pwr_q15 (const q15_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

平方和。

注！

返回值为 Q30 类型。

riscv_dsp_pwr_q7

原型：

q31_t riscv_dsp_pwr_q7 (const q7_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

平方和。

注！

返回值为 Q14 类型。

3.6.6 标准差函数

标准差函数，用于计算一个向量的标准差值，实现过程如下所示：

sos = src[0]^2 + src[1]^2 + ... + src[size-1]^2;

sqrsum = (src[0] + src[1] + ... + src[size-1])^2;

std = sqrt((sos - sqrsum / size) / (size - 1));

对于 sqrt()函数，具体参考 3.8.7 平方根函数。

DSP 软件编程函数库支持以下不同数据类型的标准差函数：浮点型、Q31、Q15 和其他数据类型，以下各节详细描述各个标准差函数。

riscv_dsp_std_f32

原型：

float32_t riscv_dsp_std_f32 (const float32_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

标准偏差值。

riscv_dsp_std_q31

原型：

q31_t riscv_dsp_std_q31 (const q31_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

标准偏差值。

riscv_dsp_std_q15

原型：

q15_t riscv_dsp_std_q15 (const q15_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

标准偏差值。

riscv_dsp_std_u8

原型：

q15_t riscv_dsp_std_u8 (const uint8_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

标准偏差值。

3.6.7 方差函数

方差函数，用于计算一个向量的方差值，实现过程如下所示：

$sos = src[0]^2 + src[1]^2 + ... + src[size-1]^2;$

$sqrsum = (src[0] + src[1] + ... + src[size-1])^2;$

$v = (sos - sqrsum / size) / (size - 1);$

DSP 软件编程函数库支持以下不同数据类型的方差函数：浮点型、Q31 和 Q15，以下各节详细描述各个方差函数。

riscv_dsp_var_f32

原型：

float32_t riscv_dsp_var_f32 (const float32_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

方差值。

riscv_dsp_var_q31

原型：

q63_t riscv_dsp_var_q31 (const q31_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

方差值。

riscv_dsp_var_q15

原型：

q31_t riscv_dsp_var_q15 (const q15_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

方差值。

3.6.8 熵函数

熵函数，用于计算一个向量的熵，实现过程如下所示：

$$\text{Entropy} = -(\sum_{i=0}^{size-1} src[i] \times \ln(src[i]))$$

DSP 软件编程函数库支持以下不同数据类型的熵函数：浮点型，以下各节详细描述各个熵函数。

riscv_dsp_entropy_f32

原型：

float32_t riscv_dsp_entropy_f32 (const float32_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

熵值。

3.6.9 相对熵函数

相对熵函数，用于计算两个向量的熵，也称为 DKL（Kullback–Leibler divergence），实现过程如下所示：

$$DKL(src1 \square src2) = -(\sum_{i=0}^{n} src1[i] \times \ln(\frac{src2[i]}{src1[i]}))$$

DSP 软件编程函数库支持以下不同数据类型的相对熵函数：浮点型，以下各节详细描述各个相对熵函数。

riscv_dsp_relative_entropy_f32

原型：

float32_t riscv_dsp_relative_entropy_f32 (const float32_t *src1, const float32_t *src2, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] size 向量的元素数量

返回值：

相对熵值。

3.6.10 LSE 函数

LSE（Log-Sum-Exp）函数，用于计算一个向量的指数和的对数，实现过程如下所示：

$LSE(src_1, \ldots, src_n) = src^* + \log(\exp(src_1 - src^*) + \ldots + \exp(src_n - src^*))$

where $src^* = \max\{src_1, \ldots, src_n\}$

DSP 软件编程函数库支持以下不同数据类型的 LSE 函数：浮点型，以下各节详细描述各个 LSE 函数。

riscv_dsp_lse_f32

原型：

float32_t riscv_dsp_lse_f32 (const float32_t *src, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] size 向量的元素数量

返回值：

LSE 值。

3.6.11 LSE 点积函数

LSE（Log-Sum-Exp）点积函数，用于计算两个向量的点积的指数和的对数，实现过程如下所示：

$LSE(X_1, \ldots, X_n) = X^* + \log(\exp(X_1-X^*) + \ldots + \exp(X_n-X^*))$

where $X^* = \max\{X_1, \ldots, X_n\}$, X_n = src1[n] + src2[n]

DSP 软件编程函数库支持以下不同数据类型的 LSE 点积函数：浮点型，以下各节详细描述各个 LSE 点积函数。

riscv_dsp_lse_dprod_f32

原型：

float32_t riscv_dsp_lse_dprod_f32 (const float32_t *src1, const float32_t *src2, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] size 向量的元素数量

返回值：

LSE 点积值。

3.6.12 朴素高斯贝叶斯预测函数

朴素高斯贝叶斯预测函数，实现朴素高斯贝叶斯预测器。

DSP 软件编程函数库支持以下不同数据类型的朴素高斯贝叶斯预测函数：浮点型，以下各节详细描述各个朴素高斯贝叶斯预测函数。

riscv_dsp_gaussian_naive_bayes_est_f32

原型：

uint32_t riscv_dsp_gaussian_naive_bayes_est_f32 (const riscv_dsp_gaussian_naivebayes_f32_t *instance, const float32_t *src, float32_t *buf)

参数：

- riscv_dsp_gaussian_naivebayes_f32_t 浮点朴素高斯贝叶斯预测器的实例化结构体，定义如下：

```
typedef struct
{
   uint32_t dimofvec;
   uint32_t numofclass;
   const float32_t *mean;
   const float32_t *var;
   const float32_t *classprior;
   float32_t additiveofvar;
} riscv_dsp_gaussian_naivebayes_f32_t;
```

其中，

  - dimofvec 向量空间的维度
  - numofclass 分类的数量
  - *mean 高斯分布的均值，其大小为 dimofvec*numofclass
  - *var 高斯分布的方差，其大小为 dimofvec*numofclass
  - *classprior 先验概率，其大小为 numofclass
  - Additiveofvar 方差的相加值
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [in] *buf 指向缓存区向量的指针

返回值：

估计类。

3.7 转换函数

DSP 软件编程函数库中，分辨率默认设置为 1024（1024，以 2 为底对

数为 10），此默认值会影响余弦和正弦查找表的分辨率（即表中元素的数量），还会影响变换函数，因为变换函数会使用这些表来查找余弦或正弦值。因此，本章为变换函数提供了一些输入向量大小的建议值，请从示例中的函数参数 m 或编译选项 FFT_LOGN 中找到这些值。如果使用了没有建议过的值，尽管程序仍可以运行，但可能会有准确性较差的风险。

注意，在 Q31 和 Q15 转换函数中可能会出现溢出，为避免此问题，须在调用这些函数之前执行算术右移操作，根据不同的输入格式，移位量也不同，这取决于输入的大小。

关于输入格式的详细描述，请参考本章各节中 Q31 和 Q15 转换函数的说明。

3.7.1 Radix-2 CFFT 函数

Radix-2 CFFT（Complex Fast Fourier Transform）函数，实现了著名的 Cooley-Tukey 算法，将信号从时域变换到频域。输入向量中的复数排列为[real, imaginary, real, imaginary..., real, imaginary]。

DSP 软件编程函数库支持以下不同数据类型的 Radix-2 CFFT 和 CIFFT（Complex Inverse FFT）函数：浮点型、Q31 和 Q15。对于 Q31 和 Q15 Radix-2 CFFT 和 CIFFT 函数，须在调用之前进行算术右移操作。

注！

本节中参数 m 表示输入样本数以 2 为底的对数值，范围为[3, 10]。即，如果输入向量有 128 个样本，则 m 等于 7（log2(128) = 7）。

以下各节详细描述各个 Radix-2 CFFT 和 CIFFT 函数。

riscv_dsp_cfft_rd2_f32

原型：

int32_t riscv_dsp_cfft_rd2_f32 (float32_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 10

返回值：

0 成功

-1 失败

示例：

给定 128 个样本（即，FFT_LOGN = 7），浮点型 Radix-2 CFFT 和

CIFFT 示例，如下所示：

```
#define FFT_LOGN 7
float32_t src[2* (1 << FFT_LOGN)] = {…};
int32_t ret;
ret = riscv_dsp_cfft_rd2_f32(src, FFT_LOGN);
if (ret == 0)
Success
Else
Fail
ret = riscv_dsp_cifft_rd2_f32(src, FFT_LOGN);
if (ret == 0)
Success
Else
Fail
```

本示例同样适用于 Q31 和 Q15 类型的 Radix-2 CFFT 和 CIFFT 函数。

riscv_dsp_cifft_rd2_f32

原型：

```
int32_t riscv_dsp_cifft_rd2_f32 (float32_t *src, uint32_t m)
```

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 10

返回值：

0 成功

-1 失败

riscv_dsp_cfft_rd2_q31

原型：

```
int32_t riscv_dsp_cfft_rd2_q31 (q31_t *src, uint32_t m)
```

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向

量中

- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 10

返回值：

0 成功

-1 失败

注！

输入和输出格式如表 3-1 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-1 The Input and Output Formats

m	Size	Input format	Output format
3	8	Q1.31	Q5.27
4	16	Q1.31	Q6.26
5	32	Q1.31	Q7.25
6	64	Q1.31	Q8.24
7	128	Q1.31	Q9.23
8	256	Q1.31	Q10.22
9	512	Q1.31	Q11.21
10	1024	Q1.31	Q12.20

riscv_dsp_cifft_rd2_q31

原型：

int32_t riscv_dsp_cifft_rd2_q31 (q31_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 10

返回值：

0 成功

-1 失败

注！

输入和输出格式如表 3-2 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-2 The Input and Output Formats

m	Size	Input format	Output format
3	8	Q5.27	Q1.31
4	16	Q6.26	Q1.31
5	32	Q7.25	Q1.31
6	64	Q8.24	Q1.31
7	128	Q9.23	Q1.31
8	256	Q10.22	Q1.31
9	512	Q11.21	Q1.31
10	1024	Q12.20	Q1.31

riscv_dsp_cfft_rd2_q15

原型：

int32_t riscv_dsp_cfft_rd2_q15 (q15_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 10

返回值：

0 成功

-1 失败

注！

输入和输出格式如表 3-3 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-3 The Input and Output Formats

m	Size	Input format	Output format
3	8	Q1.15	Q5.11
4	16	Q1.15	Q6.10
5	32	Q1.15	Q7.9
6	64	Q1.15	Q8.8
7	128	Q1.15	Q9.7
8	256	Q1.15	Q10.6

m	Size	Input format	Output format
9	512	Q1.15	Q11.5
10	1024	Q1.15	Q12.4

riscv_dsp_cifft_rd2_q15

原型：

int32_t riscv_dsp_cifft_rd2_q15 (q15_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 10

返回值：

0 成功

-1 失败

注！

输入和输出格式如表 3-4 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-4 The Input and Output Formats

m	Size	Input format	Output format
3	8	Q5.11	Q1.15
4	16	Q6.10	Q1.15
5	32	Q7.9	Q1.15
6	64	Q8.8	Q1.15
7	128	Q9.7	Q1.15
8	256	Q10.6	Q1.15
9	512	Q11.5	Q1.15
10	1024	Q12.4	Q1.15

3.7.2 Radix-4 CFFT 函数

Radix-4 CFFT（Complex Fast Fourier Transform）函数，实现了著名的 Cooley-Tukey 算法，将信号从时域变换到频域。输入向量中的复数排列为[real, imaginary, real, imaginary..., real, imaginary]。

DSP 软件编程函数库支持以下不同数据类型的 Radix-4 CFFT 和

CIFFT（Complex Inverse FFT）函数：浮点型、Q31 和 Q15。对于 Q31 和 Q15 Radix-4 CFFT 和 CIFFT 函数，须在调用之前进行算术右移操作。

注意，本节中，参数 m 表示输入样本数以 2 为底的对数值，范围为[4, 10]。即，如果输入向量有 256 个样本，则 m 等于 7（log2(256) = 8）。

以下各节详细描述各个 Radix-4 CFFT 和 CIFFT 函数。

riscv_dsp_cfft_rd4_f32

原型：

int32_t riscv_dsp_cfft_rd4_f32 (float32_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 4、6、8 或 10

返回值：

0 成功

-1 失败

示例：

给定 256 个样本（即，FFT_LOGN = 8），浮点型 Radix-4 CFFT 和 CIFFT 示例，如下所示：

```
#define FFT_LOGN 8
float32_t src[2* (1 << FFT_LOGN)] = {…};
int32_t ret;
ret = riscv_dsp_cfft_rd4_f32(src, FFT_LOGN);
if (ret == 0)
Success
Else
Fail
ret = riscv_dsp_cifft_rd4_f32(src, FFT_LOGN);
if (ret == 0)
Success
Else
Fail
```

本示例同样适用于 Q31 或 Q15 类型的 Radix-4 CFFT 和 CIFFT 函数。

riscv_dsp_cifft_rd4_f32

原型：

int32_t riscv_dsp_cifft_rd4_f32 (float32_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 4、6、8 或 10

返回值：

0 成功

-1 失败

示例：

请参考第 3.7.2.1 节示例。

riscv_dsp_cfft_rd4_q31

原型：

int32_t riscv_dsp_cfft_rd4_q31 (q31_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 4、6、8 或 10

返回值：

0 成功

-1 失败

注！

输入和输出格式如表 3-5 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-5 Input and Output Formats

m	Size	Input format	Output format
4	16	Q1.31	Q6.26
6	64	Q1.31	Q8.24

m	Size	Input format	Output format
8	256	Q1.31	Q10.22
10	1024	Q1.31	Q12.20

riscv_dsp_cifft_rd4_q31

原型：

int32_t riscv_dsp_cifft_rd4_q31 (q31_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 4、6、8 或 10

返回值：

0 成功

-1 失败

注！

输入和输出格式如表 3-6 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-6 Input and Output Formats

m	Size	Input format	Output format
4	16	Q6.26	Q1.31
6	64	Q8.24	Q1.31
8	256	Q10.22	Q1.31
10	1024	Q12.20	Q1.31

riscv_dsp_cfft_rd4_q15

原型：

int32_t riscv_dsp_cfft_rd4_q15 (q15_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 4、6、8 或 10

返回值：

0 成功

-1 失败

注！

输入和输出格式如表 3-7 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-7 Input and Output Formats

m	Size	Input format	Output format
4	16	Q1.15	Q6.10
6	64	Q1.15	Q8.8
8	256	Q1.15	Q10.6
10	1024	Q1.15	Q12.4

riscv_dsp_cifft_rd4_q15

原型：

int32_t riscv_dsp_cifft_rd4_q15 (q15_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 4、6、8 或 10

返回值：

0 成功

-1 失败

注！

输入和输出格式如表 3-8 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-8 Input and Output Formats

m	Size	Input format	Output format
4	16	Q6.10	Q1.15
6	64	Q8.8	Q1.15
8	256	Q10.6	Q1.15
10	1024	Q12.4	Q1.15

3.7.3 CFFT 函数

本节中的 CFFT（Complex FFT）函数根据参数 m，分别内部调用 3.7.1 Radix-2 CFFT 函数和 3.7.2 Radix-4 CFFT 函数的 Radix-2 或 Radix-4 CFFT/CIFFT 函数，参数 m 表示输入样本数的底数为 2 的对数值。如果 m 的值是 2 的倍数，例如 4 或 6，则调用 Radix-4 的函数以提高性能，否则调用 Radix-2 的函数。

DSP 软件编程函数库支持以下不同数据类型的 CFFT 和 CIFFT 函数：浮点型、Q31 和 Q15。对于 Q31 和 Q15 Radix-4 CFFT 和 CIFFT 函数，须在调用之前进行算术右移操作。

以下各节详细描述各个 CFFT 和 CIFFT 函数。

riscv_dsp_cfft_f32

原型：

int32_t riscv_dsp_cfft_f32 (float32_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 10

返回值：

0 成功

-1 失败

示例：

给定 128 个样本（即，FFT_LOGN = 7），浮点型 CFFT 和 CIFFT 示例，如下所示：

```
#define FFT_LOGN 7
float32_t src[2* (1 << FFT_LOGN)] = {…};
int32_t ret;
ret = riscv_dsp_cfft_f32(src, FFT_LOGN);
if (ret == 0)
Success
Else
Fail
ret = riscv_dsp_cifft_f32(src, FFT_LOGN);
```

```
if (ret == 0)
Success
Else
Fail
```

本示例同样适用于 Q31 和 Q15 类型的 CFFT 和 CIFFT 函数。

riscv_dsp_cifft_f32

原型：

```
int32_t riscv_dsp_cifft_f32 (float32_t *src, uint32_t m)
```

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 10

返回值：

0 成功

-1 失败

riscv_dsp_cfft_q31

原型：

```
int32_t riscv_dsp_cfft_q31 (q31_t *src, uint32_t m)
```

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 10

返回值：

0 成功

-1 失败

注！

输入和输出格式如表 3-9 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-9 Input and Output Formats

m	Size	Input format	Output format
3	8	Q1.31	Q5.27
4	16	Q1.31	Q6.26
5	32	Q1.31	Q7.25
6	64	Q1.31	Q8.24
7	128	Q1.31	Q9.23
8	256	Q1.31	Q10.22
9	512	Q1.31	Q11.21
10	1024	Q1.31	Q12.20

riscv_dsp_cifft_q31

原型：

int32_t riscv_dsp_cifft_q31 (q31_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 10

返回值：

0 成功

-1 失败

注！

输入和输出格式如表 3-10 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-10 Input and Output Formats

m	Size	Input format	Output format
3	8	Q5.27	Q1.31
4	16	Q6.26	Q1.31
5	32	Q7.25	Q1.31
6	64	Q8.24	Q1.31
7	128	Q9.23	Q1.31
8	256	Q10.22	Q1.31

m	Size	Input format	Output format
9	512	Q11.21	Q1.31
10	1024	Q12.20	Q1.31

riscv_dsp_cfft_q15

原型：

int32_t riscv_dsp_cfft_q15 (q15_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 10

返回值：

0 成功

-1 失败

注！

输入和输出格式如表 3-11 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-11 Input and Output Formats

m	Size	Input format	Output format
3	8	Q1.15	Q5.11
4	16	Q1.15	Q6.10
5	32	Q1.15	Q7.9
6	64	Q1.15	Q8.8
7	128	Q1.15	Q9.7
8	256	Q1.15	Q10.6
9	512	Q1.15	Q11.5
10	1024	Q1.15	Q12.4

riscv_dsp_cifft_q15

原型：

int32_t riscv_dsp_cifft_q15 (q15_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 10

返回值：

0 成功

-1 失败

注！

输入和输出格式如表 3-12 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-12 Input and Output Formats

m	Size	Input format	Output format
3	8	Q5.11	Q1.15
4	16	Q6.10	Q1.15
5	32	Q7.9	Q1.15
6	64	Q8.8	Q1.15
7	128	Q9.7	Q1.15
8	256	Q10.6	Q1.15
9	512	Q11.5	Q1.15
10	1024	Q12.4	Q1.15

3.7.4 DCT Type II 函数

DCT（Discrete Cosine Transform）Type II 函数实现 DCT，等式如下所示：

$$y[k] = \sum_{n=0}^{N-1}(x[n] * \cos(\pi * (2 * n + 1) * \frac{k}{(2 * N)}))$$

以及 IDCT（DCT Type III，又被称为 DCT II 的逆型），等式如下所示：

$$x[k] = \frac{2.0}{N} * \sum_{n=0}^{N-1}(c[n] * \mathrm{y[n]} * \cos(\pi * (2 * k + 1) * \frac{n}{(2 * N)}))$$

其中，c[0] = 1 / 2.0，c[n] = 1 for n != 0。

DSP 软件编程函数库支持以下不同数据类型的 DCT Type II 和 IDCT 函数：浮点型、Q31 和 Q15。对于 Q31 和 Q15 DCT Type II 和 IDCT 函数，须在调用之前进行算术右移操作。

以下各节详细描述各个 DCT Type II 和 IDCT 函数。

riscv_dsp_dct_f32

原型：

```
void riscv_dsp_dct_f32(float32_t *src, uint32_t m)
```

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 8

返回值：

无。

示例：

给定 256 个样本（即，FFT_LOGN = 8），浮点型 DCT Type II 和 IDCT 示例，如下所示：

```
#define FFT_LOGN 8
float32_t src[(1 << FFT_LOGN)] = {…};
riscv_dsp_dct_f32(src, FFT_LOGN);
riscv_dsp_idct_f32(src, FFT_LOGN);
```

本示例同样适用于 Q31 或 Q15 类型的 DCT Type II 和 IDCT 函数。

riscv_dsp_idct_f32

原型：

```
void riscv_dsp_idct_f32(float32_t *src, uint32_t m)
```

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 8

返回值：

无。

示例：

请参考 riscv_dsp_dct_f32 示例。

riscv_dsp_dct_q31

原型：

void riscv_dsp_dct_q31(q31_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 8

返回值：

无。

注！

输入和输出格式如表 3-13 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-13 Input and Output Formats

m	Size	Input format	Output format
3	8	Q2.30	Q5.27
4	16	Q2.30	Q6.26
5	32	Q2.30	Q7.25
6	64	Q2.30	Q8.24
7	128	Q2.30	Q9.23
8	256	Q2.30	Q10.22

riscv_dsp_idct_q31

原型：

void riscv_dsp_idct_q31(q31_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 8

返回值：

无。

注！

输入和输出格式如表 3-14 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-14 Input and Output Formats

m	Size	Input format	Output format
3	8	Q5.27	Q2.30
4	16	Q6.26	Q2.30
5	32	Q7.25	Q2.30
6	64	Q8.24	Q2.30
7	128	Q9.23	Q2.30
8	256	Q10.22	Q2.30

riscv_dsp_dct_q15

原型：

void riscv_dsp_dct_q15(q15_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 8

返回值：

无。

注！

输入和输出格式如表 3-15 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-15 Input and Output Formats

m	Size	Input format	Output format
3	8	Q2.14	Q5.11
4	16	Q2.14	Q6.10
5	32	Q2.14	Q7.9
6	64	Q2.14	Q8.8
7	128	Q2.14	Q9.7
8	256	Q2.14	Q10.6

riscv_dsp_idct_q15

原型：

void riscv_dsp_idct_q15(q15_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 8

返回值：

无。

注！

输入和输出格式如表 3-16 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-16 Input and Output Formats

m	Size	Input format	Output format
3	8	Q5.11	Q2.14
4	16	Q6.10	Q2.14
5	32	Q7.9	Q2.14
6	64	Q8.8	Q2.14
7	128	Q9.7	Q2.14
8	256	Q10.6	Q2.14

3.7.5 DCT Type IV 函数

DCT（Discrete Cosine Transform）Type IV 函数，实现 DCT 转换，等式如下所示：

$$y[k] = \sum_{n=0}^{N-1}(x[n] * \cos(PI * (2 * k + 1) * \frac{(2 * n + 1)}{(4 * N)}))$$

以及 IDCT 函数，等式如下所示：

$$x[k] = \frac{2}{N} * \sum_{n=0}^{N-1}(y[n] * \cos(PI * (2 * k + 1) * \frac{(2 * n + 1)}{(4 * N)}))$$

DSP 软件编程函数库支持以下不同数据类型的 DCT Type IV 和 IDCT 函数：浮点型、Q31 和 Q15。对于 Q31 和 Q15 DCT Type IV 和 IDCT 函数，须在调用之前进行算术右移操作。

以下各节详细描述各个 DCT Type IV 和 IDCT 函数。

riscv_dsp_dct4_f32

原型：

void riscv_dsp_dct4_f32(float32_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 7

返回值：

无。

示例：

给定 128 个样本（即，FFT_LOGN = 7），浮点型 DCT 或 IDCT Type IV 函数示例，如下所示：

```
#define FFT_LOGN 8
float32_t src[(1 << FFT_LOGN)] = {…};
riscv_dsp_dct4_f32(src, FFT_LOGN);
riscv_dsp_idct4_f32(src, FFT_LOGN);
```

本示例同样适用于 Q31 或 Q15 类型的 DCT 和 IDCT Type IV 函数。

riscv_dsp_idct4_f32

原型：

void riscv_dsp_idct4_f32(float32_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 7

返回值：

无。

示例：

请参考 riscv_dsp_dct4_f32 示例。

riscv_dsp_dct4_q31

原型：

void riscv_dsp_dct4_q31(q31_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 7

返回值：

无。

注！

输入和输出格式如表 3-17 Input and Output Formats 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-17 Input and Output Formats

m	Size	Input format	Output format
3	8	Q2.30	Q6.26
4	16	Q2.30	Q7.25
5	32	Q2.30	Q8.24
6	64	Q2.30	Q9.23
7	128	Q2.30	Q10.22

riscv_dsp_idct4_q31

原型：

void riscv_dsp_idct4_q31(q31_t *src, uint32_t m)

参数：

[in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中

[in] m 以 2 为基数的样本数的对数值，其设置为 3 到 7

返回值：

无。

注！

输入和输出格式如表 3-18 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-18 Input and Output Formats

m	Size	Input format	Output format
3	8	Q6.26	Q2.30
4	16	Q7.25	Q2.30
5	32	Q8.24	Q2.30
6	64	Q9.23	Q2.30
7	128	Q10.22	Q2.30

riscv_dsp_dct4_q15

原型：

void riscv_dsp_dct4_q15(q15_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 7

返回值：

无。

注！

输入和输出格式如表 3-19 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-19 Input and Output Formats

m	Size	Input format	Output format
3	8	Q2.14	Q6.10
4	16	Q2.14	Q7.9
5	32	Q2.14	Q8.8
6	64	Q2.14	Q9.7
7	128	Q2.14	Q10.6

riscv_dsp_idct4_q15

原型：

void riscv_dsp_idct4_q15(q15_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 3 到 7

返回值：

无。

注！

输入和输出格式如表 3-20 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-20 Input and Output Formats

m	Size	Input format	Output format
3	8	Q6.10	Q2.14
4	16	Q7.9	Q2.14
5	32	Q8.8	Q2.14
6	64	Q9.7	Q2.14
7	128	Q10.6	Q2.14

3.7.6 RFFT 函数

RFFT（Real Fast Fourier Transform）函数， 和 RIFFT（Real Inverse FFT）函数，将由实数组成的信号，从时域变换到频域。

RFFT 算法将输入的 N 个实数视为 N/2 复数，执行 CFFT。

计算 CFFT 后，频谱 N/2 处的实数保存在第一个虚位置，即 RFFT 的输出布局排列为[r[0], r[N/2], r[1], i[1], ..., r[N/2 - 1], i[N/2 - 1]]，其中，r 是实数，i 是虚数，N 是输入数据的大小。由于在转换期间没有创建额外的数组，因此输出的大小与输入的大小相同。

对于 RIFFT，其输入数据与 RFFT 的输出数据具有相同的格式，计算后的输出数据由频谱转换为时域。

DSP 软件编程函数库支持以下不同数据类型的 RFFT 和 RIFFT 函数：浮点型、Q31 和 Q15。对于 Q31 和 Q15 RFFT 和 RIFFT 函数，须在调用之前进行算术右移操作。

以下各节详细描述各个 RFFT 和 RIFFT 函数。

riscv_dsp_rfft_f32

原型：

int32_t riscv_dsp_rfft_f32(float32_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 4 到 10

返回值：

0 成功

-1 失败

示例：

给定 128 个样本（即，FFT_LOGN = 7），浮点型 RFFT 和 RIFFT 函数示例，如下所示：

```
#define FFT_LOGN 7
float32_t src[(1 << FFT_LOGN)] = {…};
int32_t ret;
ret = riscv_dsp_rfft_f32(src, FFT_LOGN);
if (ret == 0)
Success
else
Fail
ret = riscv_dsp_rifft_f32(src, FFT_LOGN);
if (ret == 0)
Success
else
Fail
```

本示例同样适用于 Q31 或 Q15 类型的 RFFT 和 RIFFT 函数。

riscv_dsp_rifft_f32

原型：

```
int32_t riscv_dsp_rifft_f32(float32_t *src, uint32_t m)
```

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 4 到 10

返回值：

0 成功

-1 失败

riscv_dsp_rfft_q31

原型：

```
int32_t riscv_dsp_rfft_q31(q31_t *src, uint32_t m)
```

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 4 到 10

返回值：

0 成功

-1 失败

注！

输入和输出格式如表 3-21 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-21 Input and Output Formats

m	Size	Input format	Output format
4	16	Q1.31	Q6.26
5	32	Q1.31	Q7.25
6	64	Q1.31	Q8.24
7	128	Q1.31	Q9.23
8	256	Q1.31	Q10.22
9	512	Q1.31	Q11.21
10	1024	Q1.31	Q12.20

riscv_dsp_rifft_q31

原型：

```
int32_t riscv_dsp_rifft_q31(q31_t *src, uint32_t m)
```

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向

量中

- [in] m 以 2 为基数的样本数的对数值，其设置为 4 到 10

返回值：

0 成功

-1 失败

注！

输入和输出格式如表 3-22 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-22 Input and Output Formats

m	Size	Input format	Output format
4	16	Q6.26	Q1.31
5	32	Q7.25	Q1.31
6	64	Q8.24	Q1.31
7	128	Q9.23	Q1.31
8	256	Q10.22	Q1.31
9	512	Q11.21	Q1.31
10	1024	Q12.20	Q1.31

riscv_dsp_rfft_q15

原型：

int32_t riscv_dsp_rfft_q15(q15_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 4 到 10

返回值：

0 成功

-1 失败

注！

输入和输出格式如表 3-23 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-23 Input and Output Formats

m	Size	Input format	Output format
4	16	Q1.15	Q6.10
5	32	Q1.15	Q7.9
6	64	Q1.15	Q8.8
7	128	Q1.15	Q9.7
8	256	Q1.15	Q10.6
9	512	Q1.15	Q11.5
10	1024	Q1.15	Q12.4

riscv_dsp_rifft_q15

原型：

int32_t riscv_dsp_rifft_q15(q15_t *src, uint32_t m)

参数：

- [in, out] *src 指向输入向量的指针，函数执行后，输出将存储在输入向量中
- [in] m 以 2 为基数的样本数的对数值，其设置为 4 到 10

返回值：

0 成功

-1 失败

注！

输入和输出格式如表 3-24 所示。为满足与输入大小相对应的输入格式，须在调用此函数前进行算术移位操作。

表 3-24 Input and Output Formats

m	Size	Input format	Output format
4	16	Q6.10	Q1.15
5	32	Q7.9	Q1.15
6	64	Q8.8	Q1.15
7	128	Q9.7	Q1.15
8	256	Q10.6	Q1.15
9	512	Q11.5	Q1.15
10	1024	Q12.4	Q1.15

3.8 工具函数

3.8.1 arctan 函数

arctan 函数是反三角函数，返回给定输入值的 tan 函数的反函数值。arctan 函数的输入值和输出结果以弧度为单位，范围为[-π/2, π/2]。

DSP 软件编程函数库支持以下不同数据类型的 arctan 函数：浮点型、Q31 和 Q15，以下各节详细描述各个 arctan 函数。

riscv_dsp_atan_f32

原型：

float32_t riscv_dsp_atan_f32 (float32_t src)

参数：

- [in] src 输入值（弧度）

返回值：

弧度值。

示例：

```
float32_t src = 0.5;
float32_t dst;
dst = riscv_dsp_atan_f32 (src);
```

本示例同样适用于 Q31 或 Q15 类型的 arctan 函数。

riscv_dsp_atan_q31

原型：

q31_t riscv_dsp_atan_q31 (q31_t src)

参数：

- [in] src 输入值（弧度）

返回值：

弧度值。

riscv_dsp_atan_q15

原型：

q15_t riscv_dsp_atan_q15 (q15_t src)

参数：

- [in] src 输入值（弧度）

返回值：

弧度值。

3.8.2 arctan2 函数

arctan2 函数类似于 arctan 函数，但接收两个输入 y 和 x 来计算 arctan 值。通常，输出结果都是以弧度为单位，范围为[-π，π]。为方便计算，它们在 Q 型的函数中映射为[- 1,1)，但是可以使用转换公式将它们转换为对应的浮点型。

DSP 软件编程函数库支持以下不同数据类型的 arctan2 函数：浮点型、Q31 和 Q15，以下各节详细描述各个 arctan2 函数。

riscv_dsp_atan2_f32

原型：

q31_t riscv_dsp_atan2_f32 (float32_t y, float32_t x)

参数：

- [in] y 输入值 y
- [in] x 输入值 x

返回值：

弧度值。

注！

浮点型返回值的范围通常为[-π, π]。

riscv_dsp_atan2_q31

原型：

q31_t riscv_dsp_atan2_q31 (q31_t y, q31_t x)

参数：

- [in] y 输入值 y
- [in] x 输入值 x

返回值：

弧度值。

注！

Q31 型返回值的范围通常为[0x80000000, 0x7FFFFFFF]（即，[-1, 1)）。如果要转为浮点型，请使用以下公式：floating point value = (return value * π) / (2 ^ 31)

riscv_dsp_atan2_q15

原型：

```
q15_t riscv_dsp_atan2_q15 (q15_t y, q15_t x)
```

参数：

- [in] y 输入值 y
- [in] x 输入值 x

返回值：

弧度值。

注！

Q15 型返回值的范围通常为[0x8000, 0x7FFF]（即，[-1, 1)）。如果要转为浮点型，请使用以下公式：floating point value = (return value * π) / (2 ^ 15)

3.8.3 cos 和 sin 函数

余弦和正弦函数，用于计算余弦和正弦值。

这些函数的输入值以弧度为单位，浮点型函数和 Q 型函数的输入值范围不同。对于浮点型函数，输入范围没有限制，因为无论输入是什么值，都会被下面的代码段预处理到范围[-2π, 2π]：

```
src = abs(src)
while (src >= 2π)
{
   src -= 2π;
}
```

其中，src 是输入值。

对于 Q 型（Q31 和 Q15）函数，仅支持输入范围[-1, 1)来表示 Q 型范围[-π, π]，因此，在调用这些函数前，须将输入值转换到 Q 型范围内。

DSP 软件编程函数库支持以下不同数据类型的 cos 和 sin 函数：浮点型、Q31 和 Q15，以下各节详细描述各个 cos 和 sin 函数。

riscv_dsp_cos_f32

原型：

float32_t riscv_dsp_cos_f32 (float32_t src)

参数：

- [in] src 输入值（弧度）

返回值：

输入的余弦值。

示例：

```
float32_t src = 0.5;
float32_t dst;
dst = riscv_dsp_cos_f32 (src);
```

本示例同样适用于 Q31 或 Q15 类型的 cosine/sine 函数。

riscv_dsp_cos_q31

原型：

q31_t riscv_dsp_cos_q31 (q31_t src)

参数：

- [in] src 输入值（弧度）

返回值：

输入的余弦值。

注！

输入范围为[0x80000000, 0x7FFFFFFF]，映射到范围[-π, π]。

riscv_dsp_cos_q15

原型：

q15_t riscv_dsp_cos_q15 (q15_t src)

参数：

- [in] src 输入值（弧度）

返回值：

输入的余弦值。

注！

输入范围为[0x8000, 0x7FFF]，映射到范围[-π, π]。

riscv_dsp_sin_f32

原型：

float32_t riscv_dsp_sin_f32 (float32_t src)

参数：

- [in] src 输入值（弧度）

返回值：

输入的正弦值。

riscv_dsp_sin_q31

原型：

q31_t riscv_dsp_sin_q31 (q31_t src)

参数：

- [in] src 输入值（弧度）

返回值：

输入的正弦值。

注！

输入范围为[0x80000000, 0x7FFFFFFF]，映射到范围 [-π, π]。

riscv_dsp_sin_q15

原型：

q15_t riscv_dsp_sin_q15 (q15_t src)

参数：

- [in] src 输入值（弧度）

返回值：

输入的正弦值。

注！

输入范围为[0x8000, 0x7FFF]，映射到范围[-π, π]。

3.8.4 转换函数

转换函数，用于将源向量中的元素值从一种数据类型转换为另一种数据类型，结果写入目的向量，包括以下数据类型转换：

float32_t q31_t

float32_t q15_t

float32_t q7_t

q31_t float32_t

q31_t q15_t

q31_t q7_t

q15_t float32_t

q15_t q31_t

q15_t q7_t

q7_t float32_t

q7_t q31_t

q7_t q15_t

以下各节详细描述各个转换函数。

riscv_dsp_convert_f32_q31

原型：

void riscv_dsp_convert_f32_q31 (float32_t *src, q31_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 此函数可以从 float32_t 到 q31_t 转换值。
- 结果饱和为 Q31 型[0x80000000, 0x7FFFFFFF]。

riscv_dsp_convert_f32_q15

原型：

void riscv_dsp_convert_f32_q15 (float32_t *src, q15_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 此函数可以从 float32_t 到 q15_t 转换值。
- 结果饱和为 Q15 型[0x8000, 0x7FFF]。

riscv_dsp_convert_f32_q7

原型：

void riscv_dsp_convert_f32_q7 (float32_t *src, q7_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 此函数可以从 float32_t 到 q7_t 转换值。
- 结果饱和为 Q7 型[0x80, 0x7F]。

riscv_dsp_convert_q31_f32

原型：

void riscv_dsp_convert_q31_f32 (q31_t *src, float32_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针

- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

此函数可以从 q31_t 到 float32_t 转换值。

riscv_dsp_convert_q31_q15

原型：

void riscv_dsp_convert_q31_q15 (q31_t *src, q15_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 此函数可以从 q31_t 到 q15_t 转换值。
- 不存在溢出，因为值仅是右移 16 位。

riscv_dsp_convert_q31_q7

原型：

void riscv_dsp_convert_q31_q7 (q31_t *src, q7_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

- 此函数可以从 q31_t 到 q17_t 转换值。

- 不存在溢出，因为值仅是右移 24 位。

riscv_dsp_convert_q15_f32

原型：

void riscv_dsp_convert_q15_f32 (q15_t *src, float32_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

此函数可以从 q15_t 到 float32_t 转换值。

riscv_dsp_convert_q15_q31

原型：

void riscv_dsp_convert_q15_q31 (q15_t *src, q31_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

此函数可以从 q15_t 到 q31_t 转换值（左移 16 位）。

riscv_dsp_convert_q15_q7

原型：

void riscv_dsp_convert_q15_q7 (q15_t *src, q7_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

此函数可以从 q15_t 到 q7_t 转换值（右移 8 位）。

riscv_dsp_convert_q7_f32

原型：

void riscv_dsp_convert_q7_f32 (q7_t *src, float32_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

此函数可以从 q7_t 到 float32_t 转换值。

riscv_dsp_convert_q7_q31

原型：

void riscv_dsp_convert_q7_q31 (q7_t *src, q31_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

此函数可以从 q7_t 到 q31_t 转换值（左移 24 位）。

riscv_dsp_convert_q7_q15

原型：

void riscv_dsp_convert_q7_q15 (q7_t *src, q15_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

注！

此函数可以从 q7_t 到 q15_t 转换值（左移 8 位）。

3.8.5 复制函数

复制函数，用于复制源向量中的元素值，依次写入目的向量，实现过程如下所示：

```
for (n = 0; n < size; n++)
  dst[n] = src[n];
```

DSP 软件编程函数库支持以下不同数据类型的复制函数：浮点型、Q31、Q15 和 Q7，以下各节详细描述各个复制函数。

riscv_dsp_dup_f32

原型：

void riscv_dsp_dup_f32 (float32_t *src, float32_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

riscv_dsp_dup_q31

原型：

void riscv_dsp_dup_q31 (q31_t *src, q31_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

riscv_dsp_dup_q15

原型：

void riscv_dsp_dup_q15 (q15_t *src, q15_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

riscv_dsp_dup_q7

原型：

void riscv_dsp_dup_q7 (q7_t *src, q7_t *dst, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

3.8.6 置位函数

置位函数，用于将目的向量中的所有元素值设置为一个常数，实现过程如下所示：

```
for (n = 0; n < size; n++)
  dst[n] = value;
```

DSP 软件编程函数库支持以下不同数据类型的置位函数：浮点型、Q31、Q15 和 Q7，以下各节详细描述各个置位函数。

riscv_dsp_set_f32

原型：

```
void riscv_dsp_set_f32 (float32_t val, float32_t *dst, uint32_t size)
```

参数：

- [in] val 被写入输出向量的值
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

riscv_dsp_set_q31

原型：

```
void riscv_dsp_set_q31 (q31_t val, q31_t *dst, uint32_t size)
```

参数：

- [in] val 被写入输出向量的值
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

riscv_dsp_set_q15

原型：

```
void riscv_dsp_set_q15 (q15_t val, q15_t *dst, uint32_t size)
```

参数：

- [in] val 被写入输出向量的值
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

riscv_dsp_set_q7

原型：

void riscv_dsp_set_q7 (q7_t val, q7_t *dst, uint32_t size)

参数：

- [in] val 被写入输出向量的值
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

返回值：

无。

3.8.7 平方根函数

平方根函数，用于返回输入值的平方根值。

DSP 软件编程函数库支持以下不同数据类型的平方根函数：浮点型、Q31 和 Q15，以下各节详细描述各个平方根函数。

riscv_dsp_sqrt_f32

原型：

float32_t riscv_dsp_sqrt_f32 (float32_t src)

参数：

- [in] src 输入值

返回值：

输入的平方根。

riscv_dsp_sqrt_q31

原型：

q31_t riscv_dsp_sqrt_q31 (q31_t src)

参数：

- [in] src 输入值

返回值：

输入的平方根。

riscv_dsp_sqrt_q15

原型：

q15_t riscv_dsp_sqrt_q15 (q15_t src)

参数：

- [in] src 输入值

返回值：

输入的平方根。

3.8.8 Barycenter 函数

Barycenter 函数，用于计算输入向量的重心。

DSP 软件编程函数库支持以下不同数据类型的 Barycenter 函数：浮点型，以下各节详细描述各个 Barycenter 函数。

riscv_dsp_barycenter_f32

原型：

void riscv_dsp_barycenter_f32(const float32_t *src, const float32_t *weights, float32_t *dst, uint32_t numofvec, uint32_t dimofvec)

参数：

- [in] *src 指向输入向量的指针
- [in] *weights 指向权重向量的指针
- [out] *dst 指向输出向量的指针
- [in] numofvec 输入向量的数量
- [in] dimofvec 向量空间的维度

返回值：

无。

3.8.9 加权和函数

加权和函数，用于计算输入向量的加权和。

DSP 软件编程函数库支持以下不同数据类型的加权和函数：浮点型，以下各节详细描述各个加权和函数。

riscv_dsp_weighted_sum_f32

原型：

float32_t riscv_dsp_weighted_sum_f32(const float32_t *src, const float32_t *weight, uint32_t size)

参数：

- [in] *src 指向输入向量的指针
- [in] *weight 指向权重向量的指针
- [in] size 向量的元素数量

返回值：

加权和值。

3.9 SVM 预测函数

SVM（Support Vector Machine）预测函数，根据提供的实例化结构体实现两类 SVM 分类。

DSP 软件编程函数库支持以下不同数据类型的 SVM 预测函数：浮点型，以下各节详细描述各个 SVM 预测函数。

3.9.1 SVM 线性预测

riscv_dsp_svm_linear_est_f32

原型：

void riscv_dsp_svm_linear_est_f32(const riscv_dsp_svm_linear_f32_t *instance, const float32_t *src, int32_t *result)

参数：

- riscv_dsp_svm_linear_f32_t 浮点 SVM 线性预测器的实例化结构体，定义如下：

  typedef struct

  {

```
    uint32_t numofvec;
    uint32_t dimofvec;
    float32_t intercept;
    const float32_t *dualcoe;
    const float32_t *vec;
    const int32_t *class;
} riscv_dsp_svm_linear_f32_t;
```

其中，

- numofvec 支撑向量的数量
- dimofvec 向量空间的维度
- intercept 截距
- *dualcoe 双系数
- *vec 支撑向量
- *class SVM 的两个分类

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *result 决策值

返回值：

无。

示例：

```
#define numofvec 3
#define dimofvec 4
#define intercept 0.1
float32_t src[dimofvec] = {1.0, 0.5, 0.4, -0.1};
float32_t dualcoe[numofvec] = {0.40, 0.10, 0.24};
float32_t vec[dimofvec * numofvec] = {0.40, 0.10, 0.24, -0.40, -0.34, 0.20, 0.06, 0.28, -0.04, -0.20, 0.08, 0.40};
int32_t class[2] = {3, 7};
int32_t *result;
riscv_dsp_svm_linear_f32_t instance = {numofvec, dimofvec, intercept, dualcoe, vec, class};
riscv_dsp_svm_linear_est_f32(&instance, src, result);
```

3.9.2 SVM 径向基（RBF）预测

riscv_dsp_svm_rbf_est_f32

原型：

void riscv_dsp_svm_rbf_est_f32(const riscv_dsp_svm_rbf_f32_t *instance, const float32_t *src, int32_t *result)

参数：

- riscv_dsp_svm_rbf_f32_t 浮点 SVM 线性预测器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t numofvec;
  uint32_t dimofvec;
  float32_t intercept;
  const float32_t *dualcoe;
  const float32_t *vec;
  const int32_t *class;
  float32_t gamma;
} riscv_dsp_svm_rbf_f32_t;
```

其中，

  - numofvec 支撑向量的数量
  - dimofvec 向量空间的维度
  - intercept 截距
  - *dualcoe 双系数
  - *vec 支撑向量
  - *class SVM 的两个分类
  - gamma 伽马因数
- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *result 决策值

返回值：

无。

示例：

```
#define numofvec 3
#define dimofvec 4
#define intercept 0.1
#define gamma 0.2
float32_t src[dimofvec] = {1.0, 0.5, 0.4, -0.1};
float32_t dualcoe[numofvec] = {0.40, 0.10, 0.24};
float32_t vec[dimofvec * numofvec] = {0.40, 0.10, 0.24, -0.40, -0.34, 0.20, 0.06, 0.28, -0.04, -0.20, 0.08, 0.40};
int32_t class[2] = {2, 5};
int32_t *result;
riscv_dsp_svm_rbf_f32_t instance = {numofvec, dimofvec, intercept, dualcoe, vec, class, gamma};
riscv_dsp_svm_rbf_est_f32(&instance, src, result);
```

3.9.3 SVM 多项式预测

riscv_dsp_svm_poly_est_f32

原型：

void riscv_dsp_svm_poly_est_f32(const riscv_dsp_svm_poly_f32_t *instance, const float32_t *src, int32_t *result)

参数：

- riscv_dsp_svm_poly_f32_t 浮点 SVM 线性预测器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t numofvec;
  uint32_t dimofvec;
  float32_t intercept;
  const float32_t *dualcoe;
  const float32_t *vec;
  const int32_t *class;
```

```
    int32_t exponent;
    float32_t coef0;
    float32_t gamma;
} riscv_dsp_svm_poly_f32_t;
```

其中，

- numofvec 支撑向量的数量
- dimofvec 向量空间的维度
- intercept 截距
- *dualcoe 双系数
- *vec 支撑向量
- *class SVM 的两个分类
- exponent 多项式指数
- coef0 独立的常数
- gamma 伽马因数

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *result 决策值

返回值：

无。

示例：

```
#define numofvec 3
#define dimofvec 4
#define intercept 0.1
#define gamma 0.2
#define exponent 3
#define coef0 0.3
float32_t src[dimofvec] = {1.0, 0.5, 0.4, -0.1};
float32_t dualcoe[numofvec] = {0.40, 0.10, 0.24};
float32_t vec[dimofvec * numofvec] = {0.40, 0.10, 0.24, -0.40, -0.34,
0.20, 0.06, 0.28, -0.04, -0.20, 0.08, 0.40};
int32_t class[2] = {-1, 4};
int32_t *result;
```

riscv_dsp_svm_poly_f32_t instance = {numofvec, dimofvec, intercept, dualcoe, vec, class, exponent, coef0, gamma};

riscv_dsp_svm_poly_est_f32(&instance, src, result);

3.9.4. SVM Sigmoid 预测

riscv_dsp_svm_sigmoid_est_f32

原型：

void riscv_dsp_svm_sigmoid_est_f32(const riscv_dsp_svm_sigmoid_f32_t *instance, const float32_t *src, int32_t *result)

参数：

- riscv_dsp_svm_sigmoid_f32_t 浮点 SVM 线性预测器的实例化结构体，定义如下：

```
typedef struct
{
  uint32_t numofvec;
  uint32_t dimofvec;
  float32_t intercept;
  const float32_t *dualcoe;
  const float32_t *vec;
  const int32_t *class;
  float32_t coef0;
  float32_t gamma;
} riscv_dsp_svm_sigmoid_f32_t;
```

其中，

- numofvec 支撑向量的数量
- dimofvec 向量空间的维度
- intercept 截距
- *dualcoe 双系数
- *vec 支撑向量
- *class SVM 的两个分类
- coef0 独立的常数

- gamma 伽马因数

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *result 决策值

返回值：

无。

示例：

```
#define numofvec 3
#define dimofvec 4
#define intercept 0.1
#define gamma 0.2
#define coef0 0.3
float32_t src[dimofvec] = {1.0, 0.5, 0.4, -0.1};
float32_t dualcoe[numofvec] = {0.40, 0.10, 0.24};
float32_t vec[dimofvec * numofvec] = {0.40, 0.10, 0.24, -0.40, -0.34, 0.20, 0.06, 0.28, -0.04, -0.20, 0.08, 0.40};
int32_t class[2] = {-1, 4};
int32_t *result;
riscv_dsp_svm_sigmoid_f32_t instance = {numofvec, dimofvec, intercept, dualcoe, vec, class, coef0, gamma};
riscv_dsp_svm_sigmoid_est_f32(&instance, src, result);
```

3.10 距离函数

距离函数，用于计算两个向量之间的距离。

DSP 软件编程函数库支持以下不同数据类型的距离函数：浮点型和布尔型，以下各节详细描述各个距离函数。

3.10.1 浮点型距离

这组函数，用于计算浮点型向量之间的距离。

以下示例适用于除 riscv_dsp_dist_minkowski_f32 函数之外的所有 RiscV_AE350_SOC 浮点型距离函数，其区别参考 riscv_dsp_dist_minkowski_f32。

```
#define size 4
```

```
float32_t src1[size] = {1.0, 0.5, 0.4, -0.1};
float32_t src2[size] = {0.40, -0.2, 0.24, 0.4};
float32_t dist_out;
dist_out = riscv_dsp_dist_bray_curtis_f32(src1, src2, size);
dist_out = riscv_dsp_dist_canberra_f32(src1, src2, size);
dist_out = riscv_dsp_dist_chebyshev_f32(src1, src2, size);
dist_out = riscv_dsp_dist_city_block_f32(src1, src2, size);
dist_out = riscv_dsp_dist_corr_f32(src1, src2, size);
dist_out = riscv_dsp_dist_cos_f32(src1, src2, size);
dist_out = riscv_dsp_dist_euclidean_f32(src1, src2, size);
```

riscv_dsp_dist_bray_curtis_f32

原型：

float32_t riscv_dsp_dist_bray_curtis_f32 (const float32_t *src1, const float32_t *src2, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] size 向量的元素数量

返回值：

距离值。

riscv_dsp_dist_canberra_f32

原型：

float32_t riscv_dsp_dist_canberra_f32 (const float32_t *src1, const float32_t *src2, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] size 向量的元素数量

返回值：

距离值。

riscv_dsp_dist_chebyshev_f32

原型：

float32_t riscv_dsp_dist_chebyshev_f32 (const float32_t *src1, const float32_t *src2, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] size 向量的元素数量

返回值：

距离值。

riscv_dsp_dist_city_block_f32

原型：

float32_t riscv_dsp_dist_city_block_f32 (const float32_t *src1, const float32_t *src2, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] size 向量的元素数量

返回值：

距离值。

riscv_dsp_dist_corr_f32

原型：

float32_t riscv_dsp_dist_corr_f32 (const float32_t *src1, const float32_t *src2, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] size 向量的元素数量

返回值：

距离值。

riscv_dsp_dist_cos_f32

原型：

float32_t riscv_dsp_dist_cos_f32 (const float32_t *src1, const float32_t *src2, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] size 向量的元素数量

返回值：

距离值。

riscv_dsp_dist_euclidean_f32

原型：

float32_t riscv_dsp_dist_euclidean_f32 (const float32_t *src1, const float32_t *src2, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] size 向量的元素数量

返回值：

距离值。

riscv_dsp_dist_jensen_shannon_f32

原型：

float32_t riscv_dsp_dist_jensen_shannon_f32 (const float32_t *src1, const float32_t *src2, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] size 向量的元素数量

返回值：

距离值。

riscv_dsp_dist_minkowski_f32

原型：

float32_t riscv_dsp_dist_minkowski_f32 (const float32_t *src1, const float32_t *src2, int32_t order, uint32_t size)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] order 距离的顺序
- [in] size 向量的元素数量

返回值：

距离值。

示例：

```
#define size 4
#define order 3
float32_t src1[size] = {1.0, 0.5, 0.4, -0.1};
float32_t src2[size] = {0.40, -0.2, 0.24, 0.4};
float32_t dist_out;
dist_out = riscv_dsp_dist_minkowski_f32(src1, src2, order, size);
```

3.10.2 布尔型距离

这组函数用于计算布尔型向量之间的距离，每个布尔值保存为 1 位，而且从最高有效位开始打包为一个 32 位的字。

如下所示，给定 120 个布尔数，需要用四个 32 位大端字来打包所需布尔型距离函数的每个源输入，即，将 3 个 32 位布尔值放入 3 个不同的字中，并将剩余的 24 位放入第 4 个字的最高有效位中，其中比特 0 到 7 是无关紧要的，本示例适用于所有 RiscV_AE350_SOC 布尔型距离函数。

示例：

```
#define numofbool 120
uint32_t src1[4] = {0x12322111, 0x24421111, 0x012ff241,
```

```
0x33c31100};
    uint32_t src2[4] = {0x22222222, 0x11ea3222, 0x13465648,
0x5b351100};
    float32_t dist_out;
    dist_out = riscv_dsp_bdist_dice_u32_f32 (src1, src2, numofbool);
    dist_out = riscv_dsp_bdist_hamming_u32_f32(src1, src2, numofbool);
    dist_out = riscv_dsp_bdist_jaccard_u32_f32(src1, src2, numofbool);
    dist_out = riscv_dsp_bdist_kulsinski_u32_f32(src1, src2, numofbool);
    dist_out = riscv_dsp_bdist_rogers_tanimoto_u32_f32(src1, src2,
numofbool);
    dist_out = riscv_dsp_bdist_russell_rao_u32_f32(src1, src2,
numofbool);
    dist_out = riscv_dsp_bdist_sokal_michener_u32_f32(src1, src2,
numofbool);
    dist_out = riscv_dsp_bdist_sokal_sneath_u32_f32(src1, src2,
numofbool);
    dist_out = riscv_dsp_bdist_yule_u32_f32(src1, src2, numofbool);
```

riscv_dsp_bdist_dice_u32_f32

原型：

```
float32_t riscv_dsp_bdist_dice_u32_f32 (const uint32_t *src1, const
uint32_t *src2, uint32_t nomofbool)
```

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] numofbool 向量中布尔值的数量

返回值：

距离值。

iscv_dsp_bdist_hamming_u32_f32

原型：

```
float32_t riscv_dsp_bdist_hamming_u32_f32 (const uint32_t *src1,
const uint32_t *src2, uint32_t nomofbool)
```

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] numofbool 向量中布尔值的数量

返回值：

距离值。

riscv_dsp_bdist_jaccard_u32_f32

原型：

float32_t riscv_dsp_bdist_jaccard_u32_f32 (const uint32_t *src1, const uint32_t *src2, uint32_t nomofbool)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] numofbool 向量中布尔值的数量

返回值：

距离值。

riscv_dsp_bdist_kulsinski_u32_f32

原型：

float32_t riscv_dsp_bdist_kulsinski_u32_f32 (const uint32_t *src1, const uint32_t *src2, uint32_t nomofbool)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] numofbool 向量中布尔值的数量

返回值：

距离值。

riscv_dsp_bdist_rogers_tanimoto_u32_f32

原型：

float32_t riscv_dsp_bdist_rogers_tanimoto_u32_f32 (const uint32_t *src1, const uint32_t *src2, uint32_t nomofbool)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] numofbool 向量中布尔值的数量

返回值：

距离值。

riscv_dsp_bdist_russell_rao_u32_f32

原型：

float32_t riscv_dsp_bdist_russell_rao_u32_f32 (const uint32_t *src1, const uint32_t *src2, uint32_t nomofbool)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] numofbool 向量中布尔值的数量

返回值：

距离值。

riscv_dsp_bdist_sokal_michener_u32_f32

原型：

float32_t riscv_dsp_bdist_sokal_michener_u32_f32 (const uint32_t *src1, const uint32_t *src2, uint32_t nomofbool)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] numofbool 向量中布尔值的数量

返回值：

距离值。

riscv_dsp_bdist_sokal_sneath_u32_f32

原型：

float32_t riscv_dsp_bdist_sokal_sneath_u32_f32 (const uint32_t *src1, const uint32_t *src2, uint32_t nomofbool)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] numofbool 向量中布尔值的数量

返回值：

距离值。

riscv_dsp_bdist_yule_u32_f32

原型：

float32_t riscv_dsp_bdist_yule_u32_f32 (const uint32_t *src1, const uint32_t *src2, uint32_t nomofbool)

参数：

- [in] *src1 指向第一个输入向量的指针
- [in] *src2 指向第二个输入向量的指针
- [in] numofbool 向量中布尔值的数量

返回值：

距离值。

3.11 排序函数

3.11.1 泛型排序函数

泛型排序函数根据其实例化结构体中指定的排序算法和排序顺序对向量的元素进行排序，包括双元排序、冒泡排序、堆排序、插入排序、快速排序和选择排序。

DSP 软件编程函数库支持以下不同数据类型的泛型排序函数：浮点型，以下各节详细描述各个泛型排序函数。

riscv_dsp_sort_init_f32

原型：

void riscv_dsp_sort_init_f32 (riscv_dsp_sort_f32_t * instance, riscv_dsp_sort_alg alg, riscv_dsp_sort_order order)

参数：

- riscv_dsp_sort_f32_t 浮点泛型排序的实例化结构体，定义如下：

  typedef struct

  {

  riscv_dsp_sort_alg alg;

  riscv_dsp_sort_order order;

  } riscv_dsp_sort_f32_t;

  其中，

  - alg 期望排序算法
  - order 期望排序顺序
- [in, out] *instance 指向实例化结构体的指针
- [in] alg 期望排序算法
- [in] order 期望排序顺序

注！

- riscv_dsp_sort_f32 函数执行前，须先调用此函数来初始化实例化结构体，具体参考 riscv_dsp_sort_f32。
- 用于泛型排序的可能的排序算法包括：
  - RISCV_DSP_SORT_BITONIC 双元排序
  - RISCV_DSP_SORT_BUBBLE 冒泡排序
  - RISCV_DSP_SORT_HEAP 堆排序
  - RISCV_DSP_SORT_INSERTION 插入排序
  - RISCV_DSP_SORT_QUICK 快速排序
  - RISCV_DSP_SORT_SELECTION 选择排序
- 用于泛型排序的可能的排序顺序，包括：
  - RISCV_DSP_SORT_DESCENDING 降序
  - RISCV_DSP_SORT_ASCENDING 升序

riscv_dsp_sort_f32

原型：

void riscv_dsp_sort_f32 (const riscv_dsp_sort_f32_t * instance, float32_t * src. float32_t * dst, uint32_t size)

参数：

- riscv_dsp_sort_f32_t 浮点泛型排序的实例化结构体，定义如下：

  typedef struct

```
{
    riscv_dsp_sort_alg alg;
    riscv_dsp_sort_order order;
} riscv_dsp_sort_f32_t;
```

其中，

- alg 期望排序算法
- order 期望排序顺序

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

注！

- 用于泛型排序的可能的排序算法，包括：
  - RISCV_DSP_SORT_BITONIC 双元排序
  - RISCV_DSP_SORT_BUBBL 冒泡排序
  - RISCV_DSP_SORT_HEAP 堆排序
  - RISCV_DSP_SORT_INSERTION 插入排序
  - RISCV_DSP_SORT_QUICK 快速排序
  - RISCV_DSP_SORT_SELECTION 选择排序
- 用于泛型排序的可能的排序顺序包括：
  - RISCV_DSP_SORT_DESCENDING 降序
  - RISCV_DSP_SORT_ASCENDING 升序
- 为确保正确结果，在执行 riscv_dsp_sort_f32 函数前，须先调用 riscv_dsp_sort_init_f32 函数初始化实例化结构体。

示例：

1. 输入大小为 100，排序顺序为降序，排序算法为冒泡排序，泛型排序的代码示例，如下所示：

```
#define size 100
riscv_dsp_sort_f32_t *instance;
float32_t src[size] = {…};
float32_t dst[size];
riscv_dsp_sort_init_f32(instance, RISCV_DSP_SORT_BUBBLE,
RISCV_DSP_SORT_DESCENDING);
riscv_dsp_sort_f32(instance, src, dst, size);
```

2. 输入大小为 100，排序顺序为升序，排序算法为快速排序，泛型排序的代码示例，如下所示：

```
#define size 100
riscv_dsp_sort_f32_t *instance;
float32_t src[size] = {…};
float32_t dst[size];
riscv_dsp_sort_init_f32(instance, RISCV_DSP_SORT_QUICK,
RISCV_DSP_SORT_ASCENDING);
riscv_dsp_sort_f32(instance, src, dst, size);
```

3.11.2 归并排序函数

归并排序算法，根据向量实例化结构体中指定的排序顺序和分治算法，对向量中的元素进行排序。分治算法会将输入数组划分为子列表，然后分治这些子列表以生成更长的有序子列表，直到只剩下一个列表。算法需要一个缓存区来保存被处理的值。

DSP 软件编程函数库支持以下不同数据类型的归并排序函数：浮点型，以下各节详细描述各个归并排序函数。

riscv_dsp_sort_merge_init_f32

原型：

```
void riscv_dsp_sort_merge_init_f32 (riscv_dsp_sort_merge_f32_t *
instance, riscv_dsp_sort_order order, float32_t * buf)
```

参数：

- riscv_dsp_sort_merge_f32_t 浮点归并排序的实例化结构体，定义如下：

```
typedef struct
{
  riscv_dsp_sort_order order;
  float32_t * buf;
} riscv_dsp_sort_f32_t;
```

其中，

  - order 期望排序顺序
  - *buf 指向工作缓存区的指针

- [in, out] *instance 指向实例化结构体的指针

- [in] order 期望排序顺序
- [in] *buf 指向工作缓存区的指针

注！

- 执行 riscv_dsp_sort_merge_f32 函数前，须先调用此函数来初始化实例化结构体，具体参考 riscv_dsp_sort_merge_f32。
- 归并排序可能的排序顺序，包括：
  - RISCV_DSP_SORT_DESCENDING 降序
  - RISCV_DSP_SORT_ASCENDING 升序

riscv_dsp_sort_merge_f32

原型：

void riscv_dsp_sort_merge_f32 (const riscv_dsp_sort_merge_f32_t * instance, float32_t * src. float32_t * dst, uint32_t size)

参数：

- riscv_dsp_sort_merge_f32_t 浮点归并排序的实例化结构体，定义如下：

```
typedef struct
{
   riscv_dsp_sort_order order;
   float32_t * buf;
} riscv_dsp_sort_f32_t;
```

Where

- order 期望排序顺序
- *buf 指向工作缓存区的指针

- [in] *instance 指向实例化结构体的指针
- [in] *src 指向输入向量的指针
- [out] *dst 指向输出向量的指针
- [in] size 向量的元素数量

注！

- 1）归并排序可能的排序顺序，包括：
  - RISCV_DSP_SORT_DESCENDING 降序
  - RISCV_DSP_SORT_ASCENDING 升序
- 2）为确保结果正确性，执行 riscv_dsp_sort_merge_f32 函数前，须先调用 riscv_dsp_sort_merge_init_f32 函数初始化实例化结构体。

示例：

输入大小 100，排列顺序为降序，归并排序的示例代码，如下所示：

```
#define size 100
riscv_dsp_sort_merge_f32_t *instance;
float32_t src[size] = {…};
float32_t buf[size];
float32_t dst[size];
riscv_dsp_sort_merge_init_f32(instance,
RISCV_DSP_SORT_DESCENDING, buf);
riscv_dsp_sort_merge_f32(instance, src, dst, size);
```

4 应用程序

4.1 程序描述

DSP 应用程序，描述了如何使用 DSP 软件编程库和数学软件编程库，快速便捷地开发 DSP 系统。

调用 DSP 软件编程库和数学软件编程库，实现 Radix-2 复快速傅里叶变换（CFFT）和快速反傅里叶变换（CIFFT）。这些函数实现了著名的 Cooley-Tukey 算法，从时域到频域转换信号。

4.2 应用程序

RiscV_AE350_SOC 提供 DSP 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_dsp。

4.3 程序运行

DSP 应用程序运行结果如下所示。

```
It's a DSP demo.
randomly generated input array:
    0: 0x0000 [0.00000000]
    1: 0x0000 [0.00000000]
    2: 0x0c7c [0.09753418]
    3: 0x0c7c [0.09753418]
    4: 0x187e [0.19134521]
    5: 0x187e [0.19134521]
    6: 0x238e [0.27777100]
    7: 0x238e [0.27777100]
    8: 0x2d41 [0.35354614]
    9: 0x2d41 [0.35354614]
   10: 0x3537 [0.41574097]
   11: 0x3537 [0.41574097]
   12: 0x3b21 [0.46194458]
   13: 0x3b21 [0.46194458]
```

```
14: 0x3ec5 [0.49038696]
15: 0x3ec5 [0.49038696]
16: 0x3fff [0.49996948]
17: 0x3fff [0.49996948]
18: 0x3ec5 [0.49038696]
19: 0x3ec5 [0.49038696]
20: 0x3b21 [0.46194458]
21: 0x3b21 [0.46194458]
22: 0x3537 [0.41574097]
23: 0x3537 [0.41574097]
24: 0x2d41 [0.35354614]
25: 0x2d41 [0.35354614]
26: 0x238e [0.27777100]
27: 0x238e [0.27777100]
28: 0x187e [0.19134521]
29: 0x187e [0.19134521]
30: 0x0c7c [0.09753418]
31: 0x0c7c [0.09753418]
32: 0x0000 [0.00000000]
33: 0x0000 [0.00000000]
34: 0xf383 [-0.09756470]
35: 0xf383 [-0.09756470]
36: 0xe782 [-0.19134521]
37: 0xe782 [-0.19134521]
38: 0xdc71 [-0.27780151]
39: 0xdc71 [-0.27780151]
40: 0xd2bf [-0.35354614]
41: 0xd2bf [-0.35354614]
42: 0xcac9 [-0.41574097]
43: 0xcac9 [-0.41574097]
44: 0xc4df [-0.46194458]
45: 0xc4df [-0.46194458]
46: 0xc13b [-0.49038696]
47: 0xc13b [-0.49038696]
48: 0xc000 [-0.50000000]
49: 0xc000 [-0.50000000]
50: 0xc13b [-0.49038696]
51: 0xc13b [-0.49038696]
52: 0xc4df [-0.46194458]
53: 0xc4df [-0.46194458]
54: 0xcac9 [-0.41574097]
55: 0xcac9 [-0.41574097]
56: 0xd2bf [-0.35354614]
57: 0xd2bf [-0.35354614]
```

```
   58: 0xdc71 [-0.27780151]
   59: 0xdc71 [-0.27780151]
   60: 0xe782 [-0.19134521]
   61: 0xe782 [-0.19134521]
   62: 0xf383 [-0.09756470]
   63: 0xf383 [-0.09756470]
----------------------------------------
after cFFT:
    0: 0xffff [-0.00003052]
    1: 0xffff [-0.00003052]
    2: 0x0fff [0.12496948]
    3: 0xefff [-0.12503052]
    4: 0x0000 [0.00000000]
    5: 0x0000 [0.00000000]
    6: 0xffff [-0.00003052]
    7: 0xffff [-0.00003052]
    8: 0x0000 [0.00000000]
    9: 0x0000 [0.00000000]
   10: 0xffff [-0.00003052]
   11: 0xffff [-0.00003052]
   12: 0x0000 [0.00000000]
   13: 0x0000 [0.00000000]
   14: 0xffff [-0.00003052]
   15: 0xffff [-0.00003052]
   16: 0x0000 [0.00000000]
   17: 0x0000 [0.00000000]
   18: 0x0000 [0.00000000]
   19: 0x0000 [0.00000000]
   20: 0x0000 [0.00000000]
   21: 0x0000 [0.00000000]
   22: 0x0000 [0.00000000]
   23: 0xffff [-0.00003052]
   24: 0x0000 [0.00000000]
   25: 0x0000 [0.00000000]
   26: 0x0000 [0.00000000]
   27: 0xffff [-0.00003052]
   28: 0x0000 [0.00000000]
   29: 0x0000 [0.00000000]
   30: 0xffff [-0.00003052]
   31: 0x0000 [0.00000000]
   32: 0x0000 [0.00000000]
   33: 0x0000 [0.00000000]
   34: 0x0000 [0.00000000]
   35: 0x0000 [0.00000000]
```

```
   36: 0x0000 [0.00000000]
   37: 0x0000 [0.00000000]
   38: 0x0000 [0.00000000]
   39: 0x0000 [0.00000000]
   40: 0x0000 [0.00000000]
   41: 0x0000 [0.00000000]
   42: 0x0000 [0.00000000]
   43: 0x0000 [0.00000000]
   44: 0x0000 [0.00000000]
   45: 0x0000 [0.00000000]
   46: 0x0000 [0.00000000]
   47: 0x0000 [0.00000000]
   48: 0x0000 [0.00000000]
   49: 0x0000 [0.00000000]
   50: 0x0000 [0.00000000]
   51: 0x0000 [0.00000000]
   52: 0x0000 [0.00000000]
   53: 0x0000 [0.00000000]
   54: 0x0000 [0.00000000]
   55: 0x0000 [0.00000000]
   56: 0x0000 [0.00000000]
   57: 0x0000 [0.00000000]
   58: 0x0000 [0.00000000]
   59: 0x0000 [0.00000000]
   60: 0x0000 [0.00000000]
   61: 0x0000 [0.00000000]
   62: 0xf000 [-0.12500000]
   63: 0x1000 [0.12500000]
----------------------------------------
----------------------------------------
***** output [scaleingout] | ** [golden] * | **** abs diff ****
    0: 0xffff [-0.00390625] | [-0.00000045] | 0x007f [0.00390581]
    1: 0xffff [-0.00390625] | [-0.00000045] | 0x007f [0.00390581]
    2: 0x0fff [15.99609375] | [16.00000000] | 0x0080 [0.00390625]
    3: 0xefff [-16.00390625] | [-15.99999809] | 0x0080 [0.00390816]
    4: 0x0000 [ 0.00000000] | [-0.00000086] | 0x0000 [0.00000086]
    5: 0x0000 [ 0.00000000] | [ 0.00000024] | 0x0000 [0.00000024]
    6: 0xffff [-0.00390625] | [-0.00000048] | 0x007f [0.00390577]
    7: 0xffff [-0.00390625] | [-0.00000012] | 0x007f [0.00390613]
    8: 0x0000 [ 0.00000000] | [ 0.00000025] | 0x0000 [0.00000025]
    9: 0x0000 [ 0.00000000] | [-0.00000040] | 0x0000 [0.00000040]
   10: 0xffff [-0.00390625] | [ 0.00000012] | 0x0080 [0.00390637]
   11: 0xffff [-0.00390625] | [ 0.00000034] | 0x0080 [0.00390659]
   12: 0x0000 [ 0.00000000] | [ 0.00000015] | 0x0000 [0.00000015]
```

```
13: 0x0000 [ 0.00000000] | [-0.00000021] | 0x0000 [0.00000021]
14: 0xffff [-0.00390625] | [ 0.00000072] | 0x0080 [0.00390697]
15: 0xffff [-0.00390625] | [ 0.00000048] | 0x0080 [0.00390673]
16: 0x0000 [ 0.00000000] | [-0.00000030] | 0x0000 [0.00000030]
17: 0x0000 [ 0.00000000] | [ 0.00000090] | 0x0000 [0.00000090]
18: 0x0000 [ 0.00000000] | [-0.00000024] | 0x0000 [0.00000024]
19: 0x0000 [ 0.00000000] | [ 0.00000000] | 0x0000 [0.00000000]
20: 0x0000 [ 0.00000000] | [-0.00000040] | 0x0000 [0.00000040]
21: 0x0000 [ 0.00000000] | [ 0.00000035] | 0x0000 [0.00000035]
22: 0x0000 [ 0.00000000] | [ 0.00000000] | 0x0000 [0.00000000]
23: 0xffff [-0.00390625] | [-0.00000083] | 0x007f [0.00390542]
24: 0x0000 [ 0.00000000] | [-0.00000078] | 0x0000 [0.00000078]
25: 0x0000 [ 0.00000000] | [-0.00000013] | 0x0000 [0.00000013]
26: 0x0000 [ 0.00000000] | [ 0.00000022] | 0x0000 [0.00000022]
27: 0xffff [-0.00390625] | [-0.00000048] | 0x007f [0.00390577]
28: 0x0000 [ 0.00000000] | [ 0.00000010] | 0x0000 [0.00000010]
29: 0x0000 [ 0.00000000] | [-0.00000008] | 0x0000 [0.00000008]
30: 0xffff [-0.00390625] | [ 0.00000000] | 0x0080 [0.00390625]
31: 0x0000 [ 0.00000000] | [ 0.00000000] | 0x0000 [0.00000000]
32: 0x0000 [ 0.00000000] | [ 0.00000021] | 0x0000 [0.00000021]
33: 0x0000 [ 0.00000000] | [ 0.00000021] | 0x0000 [0.00000021]
34: 0x0000 [ 0.00000000] | [ 0.00000000] | 0x0000 [0.00000000]
35: 0x0000 [ 0.00000000] | [ 0.00000000] | 0x0000 [0.00000000]
36: 0x0000 [ 0.00000000] | [-0.00000008] | 0x0000 [0.00000008]
37: 0x0000 [ 0.00000000] | [ 0.00000010] | 0x0000 [0.00000010]
38: 0x0000 [ 0.00000000] | [-0.00000072] | 0x0000 [0.00000072]
39: 0x0000 [ 0.00000000] | [ 0.00000060] | 0x0000 [0.00000060]
40: 0x0000 [ 0.00000000] | [-0.00000013] | 0x0000 [0.00000013]
41: 0x0000 [ 0.00000000] | [-0.00000078] | 0x0000 [0.00000078]
42: 0x0000 [ 0.00000000] | [ 0.00000012] | 0x0000 [0.00000012]
43: 0x0000 [ 0.00000000] | [-0.00000034] | 0x0000 [0.00000034]
44: 0x0000 [ 0.00000000] | [ 0.00000035] | 0x0000 [0.00000035]
45: 0x0000 [ 0.00000000] | [-0.00000040] | 0x0000 [0.00000040]
46: 0x0000 [ 0.00000000] | [-0.00000072] | 0x0000 [0.00000072]
47: 0x0000 [ 0.00000000] | [-0.00000048] | 0x0000 [0.00000048]
48: 0x0000 [ 0.00000000] | [ 0.00000090] | 0x0000 [0.00000090]
49: 0x0000 [ 0.00000000] | [-0.00000030] | 0x0000 [0.00000030]
50: 0x0000 [ 0.00000000] | [ 0.00000024] | 0x0000 [0.00000024]
51: 0x0000 [ 0.00000000] | [ 0.00000000] | 0x0000 [0.00000000]
52: 0x0000 [ 0.00000000] | [-0.00000021] | 0x0000 [0.00000021]
53: 0x0000 [ 0.00000000] | [ 0.00000015] | 0x0000 [0.00000015]
54: 0x0000 [ 0.00000000] | [ 0.00000024] | 0x0000 [0.00000024]
55: 0x0000 [ 0.00000000] | [ 0.00000036] | 0x0000 [0.00000036]
56: 0x0000 [ 0.00000000] | [-0.00000040] | 0x0000 [0.00000040]
```

```
   57: 0x0000 [ 0.00000000] | [ 0.00000025] | 0x0000 [0.00000025]
   58: 0x0000 [ 0.00000000] | [-0.00000046] | 0x0000 [0.00000046]
   59: 0x0000 [ 0.00000000] | [-0.00000048] | 0x0000 [0.00000048]
   60: 0x0000 [ 0.00000000] | [ 0.00000024] | 0x0000 [0.00000024]
   61: 0x0000 [ 0.00000000] | [-0.00000086] | 0x0000 [0.00000086]
   62: 0xf000 [-16.00000000] | [-15.99999809] | 0x0000 [0.00000191]
   63: 0x1000 [16.00000000] | [16.00000000] | 0x0000 [0.00000000]
after CFFT_RD2, maxdiff= 0x0080 [0.00390816]
-----------------------------------------
MAE is 0.00079375, RMSD is 0.00176055, NRMSD is 0.00005502,
MAXDIFF is 0.00390816, SNR is 67.12823486
CFFT_RD2 out scale up by 64
-----------------------------------------
randomly generated input array:
    0: 0x0000 [0.00000000]
    1: 0x0000 [0.00000000]
    2: 0x1000 [0.12500000]
    3: 0xf000 [-0.12500000]
    4: 0x0000 [0.00000000]
    5: 0x0000 [0.00000000]
    6: 0x0000 [0.00000000]
    7: 0x0000 [0.00000000]
    8: 0x0000 [0.00000000]
    9: 0x0000 [0.00000000]
   10: 0x0000 [0.00000000]
   11: 0x0000 [0.00000000]
   12: 0x0000 [0.00000000]
   13: 0x0000 [0.00000000]
   14: 0x0000 [0.00000000]
   15: 0x0000 [0.00000000]
   16: 0x0000 [0.00000000]
   17: 0x0000 [0.00000000]
   18: 0x0000 [0.00000000]
   19: 0x0000 [0.00000000]
   20: 0x0000 [0.00000000]
   21: 0x0000 [0.00000000]
   22: 0x0000 [0.00000000]
   23: 0x0000 [0.00000000]
   24: 0x0000 [0.00000000]
   25: 0x0000 [0.00000000]
   26: 0x0000 [0.00000000]
   27: 0x0000 [0.00000000]
   28: 0x0000 [0.00000000]
   29: 0x0000 [0.00000000]
```

```
   30: 0x0000 [0.00000000]
   31: 0x0000 [0.00000000]
----------------------------------------
after CIFFT:
    0: 0x0000 [0.00000000]
    1: 0x0000 [0.00000000]
    2: 0x0c7a [0.09747314]
    3: 0x0c7e [0.09759521]
    4: 0x187c [0.19128418]
    5: 0x1880 [0.19140625]
    6: 0x238a [0.27764893]
    7: 0x2392 [0.27789307]
    8: 0x2d40 [0.35351563]
    9: 0x2d42 [0.35357666]
   10: 0x3532 [0.41558838]
   11: 0x353a [0.41583252]
   12: 0x3b1c [0.46179199]
   13: 0x3b22 [0.46197510]
   14: 0x3ec2 [0.49029541]
   15: 0x3eca [0.49053955]
   16: 0x3ffe [0.49993896]
   17: 0x4000 [0.50000000]
   18: 0x3ec4 [0.49035645]
   19: 0x3ec6 [0.49041748]
   20: 0x3b1e [0.46185303]
   21: 0x3b20 [0.46191406]
   22: 0x3538 [0.41577148]
   23: 0x3536 [0.41571045]
   24: 0x2d40 [0.35351563]
   25: 0x2d40 [0.35351563]
   26: 0x2390 [0.27783203]
   27: 0x238e [0.27777100]
   28: 0x187c [0.19128418]
   29: 0x187c [0.19128418]
   30: 0x0c80 [0.09765625]
   31: 0x0c7a [0.09747314]
   32: 0x0000 [0.00000000]
   33: 0x0000 [0.00000000]
   34: 0xf386 [-0.09747314]
   35: 0xf382 [-0.09759521]
   36: 0xe784 [-0.19128418]
   37: 0xe780 [-0.19140625]
   38: 0xdc76 [-0.27764893]
   39: 0xdc6e [-0.27789307]
```

```
   40: 0xd2c0 [-0.35351563]
   41: 0xd2be [-0.35357666]
   42: 0xcace [-0.41558838]
   43: 0xcac6 [-0.41583252]
   44: 0xc4e4 [-0.46179199]
   45: 0xc4de [-0.46197510]
   46: 0xc13e [-0.49029541]
   47: 0xc136 [-0.49053955]
   48: 0xc002 [-0.49993896]
   49: 0xc000 [-0.50000000]
   50: 0xc13c [-0.49035645]
   51: 0xc13a [-0.49041748]
   52: 0xc4e2 [-0.46185303]
   53: 0xc4e0 [-0.46191406]
   54: 0xcac8 [-0.41577148]
   55: 0xcaca [-0.41571045]
   56: 0xd2c0 [-0.35351563]
   57: 0xd2c0 [-0.35351563]
   58: 0xdc70 [-0.27783203]
   59: 0xdc72 [-0.27777100]
   60: 0xe784 [-0.19128418]
   61: 0xe784 [-0.19128418]
   62: 0xf380 [-0.09765625]
   63: 0xf386 [-0.09747314]
----------------------------------------
**** output [scaleingout] | ** [golden] * | **** abs diff ****
    0: 0x0000 [ 0.00000000] | [-0.00000001] | 0x0000 [0.00000001]
    1: 0x0000 [ 0.00000000] | [-0.00000001] | 0x0000 [0.00000001]
    2: 0x0c7a [ 0.19494629] | [ 0.19509025] | 0x0004 [0.00014396]
    3: 0x0c7e [ 0.19519043] | [ 0.19509035] | 0x0003 [0.00010008]
    4: 0x187c [ 0.38256836] | [ 0.38268337] | 0x0003 [0.00011501]
    5: 0x1880 [ 0.38281250] | [ 0.38268340] | 0x0004 [0.00012910]
    6: 0x238a [ 0.55529785] | [ 0.55557024] | 0x0008 [0.00027239]
    7: 0x2392 [ 0.55578613] | [ 0.55557019] | 0x0007 [0.00021595]
    8: 0x2d40 [ 0.70703125] | [ 0.70710671] | 0x0002 [0.00007546]
    9: 0x2d42 [ 0.70715332] | [ 0.70710671] | 0x0001 [0.00004661]
   10: 0x3532 [ 0.83117676] | [ 0.83146954] | 0x0009 [0.00029278]
   11: 0x353a [ 0.83166504] | [ 0.83146954] | 0x0006 [0.00019550]
   12: 0x3b1c [ 0.92358398] | [ 0.92387950] | 0x0009 [0.00029552]
   13: 0x3b22 [ 0.92395020] | [ 0.92387950] | 0x0002 [0.00007069]
   14: 0x3ec2 [ 0.98059082] | [ 0.98078519] | 0x0006 [0.00019437]
   15: 0x3eca [ 0.98107910] | [ 0.98078519] | 0x0009 [0.00029391]
   16: 0x3ffe [ 0.99987793] | [ 1.00000000] | 0x0004 [0.00012207]
   17: 0x4000 [ 1.00000000] | [ 1.00000000] | 0x0000 [0.00000000]
```

```
18: 0x3ec4 [ 0.98071289] | [ 0.98078525] | 0x0002 [0.00007236]
19: 0x3ec6 [ 0.98083496] | [ 0.98078513] | 0x0001 [0.00004983]
20: 0x3b1e [ 0.92370605] | [ 0.92387938] | 0x0005 [0.00017333]
21: 0x3b20 [ 0.92382813] | [ 0.92387938] | 0x0001 [0.00005126]
22: 0x3538 [ 0.83154297] | [ 0.83146954] | 0x0002 [0.00007343]
23: 0x3536 [ 0.83142090] | [ 0.83146954] | 0x0001 [0.00004864]
24: 0x2d40 [ 0.70703125] | [ 0.70710677] | 0x0002 [0.00007552]
25: 0x2d40 [ 0.70703125] | [ 0.70710677] | 0x0002 [0.00007552]
26: 0x2390 [ 0.55566406] | [ 0.55557019] | 0x0003 [0.00009388]
27: 0x238e [ 0.55554199] | [ 0.55557019] | 0x0000 [0.00002819]
28: 0x187c [ 0.38256836] | [ 0.38268328] | 0x0003 [0.00011492]
29: 0x187c [ 0.38256836] | [ 0.38268328] | 0x0003 [0.00011492]
30: 0x0c80 [ 0.19531250] | [ 0.19509026] | 0x0007 [0.00022224]
31: 0x0c7a [ 0.19494629] | [ 0.19509029] | 0x0004 [0.00014400]
32: 0x0000 [ 0.00000000] | [-0.00000007] | 0x0000 [0.00000007]
33: 0x0000 [ 0.00000000] | [-0.00000007] | 0x0000 [0.00000007]
34: 0xf386 [-0.19494629] | [-0.19509040] | 0x0004 [0.00014411]
35: 0xf382 [-0.19519043] | [-0.19509053] | 0x0003 [0.00009990]
36: 0xe784 [-0.38256836] | [-0.38268337] | 0x0003 [0.00011501]
37: 0xe780 [-0.38281250] | [-0.38268340] | 0x0004 [0.00012910]
38: 0xdc76 [-0.55529785] | [-0.55557024] | 0x0008 [0.00027239]
39: 0xdc6e [-0.55578613] | [-0.55557019] | 0x0007 [0.00021595]
40: 0xd2c0 [-0.70703125] | [-0.70710683] | 0x0002 [0.00007558]
41: 0xd2be [-0.70715332] | [-0.70710683] | 0x0001 [0.00004649]
42: 0xcace [-0.83117676] | [-0.83146977] | 0x0009 [0.00029302]
43: 0xcac6 [-0.83166504] | [-0.83146977] | 0x0006 [0.00019526]
44: 0xc4e4 [-0.92358398] | [-0.92387962] | 0x0009 [0.00029564]
45: 0xc4de [-0.92395020] | [-0.92387962] | 0x0002 [0.00007057]
46: 0xc13e [-0.98059082] | [-0.98078507] | 0x0006 [0.00019425]
47: 0xc136 [-0.98107910] | [-0.98078519] | 0x0009 [0.00029391]
48: 0xc002 [-0.99987793] | [-1.00000000] | 0x0004 [0.00012207]
49: 0xc000 [-1.00000000] | [-1.00000000] | 0x0000 [0.00000000]
50: 0xc13c [-0.98071289] | [-0.98078525] | 0x0002 [0.00007236]
51: 0xc13a [-0.98083496] | [-0.98078513] | 0x0001 [0.00004983]
52: 0xc4e2 [-0.92370605] | [-0.92387938] | 0x0005 [0.00017333]
53: 0xc4e0 [-0.92382813] | [-0.92387938] | 0x0001 [0.00005126]
54: 0xcac8 [-0.83154297] | [-0.83146942] | 0x0002 [0.00007355]
55: 0xcaca [-0.83142090] | [-0.83146954] | 0x0001 [0.00004864]
56: 0xd2c0 [-0.70703125] | [-0.70710653] | 0x0002 [0.00007528]
57: 0xd2c0 [-0.70703125] | [-0.70710653] | 0x0002 [0.00007528]
58: 0xdc70 [-0.55566406] | [-0.55557030] | 0x0003 [0.00009376]
59: 0xdc72 [-0.55554199] | [-0.55557030] | 0x0000 [0.00002831]
60: 0xe784 [-0.38256836] | [-0.38268340] | 0x0003 [0.00011504]
```

```
   61: 0xe784 [-0.38256836] | [-0.38268340] | 0x0003 [0.00011504]
   62: 0xf380 [-0.19531250] | [-0.19509020] | 0x0007 [0.00022230]
   63: 0xf386 [-0.19494629] | [-0.19509023] | 0x0004 [0.00014395]
after CIFFT_RD2, maxdiff= 0x0009 [0.00029564]
-----------------------------------------
MAE is 0.00012192, RMSD is 0.00014900, NRMSD is 0.00007450,
MAXDIFF is 0.00029564, SNR is 73.52618408
CIFFT_RD2 out scale up by 2
-----------------------------------------
CIFFT_RD2 PASS.

Demo DSP Completed.
```


GOWIN高云
智慧逻辑 定制未来