

---

# **Read the Docs Template Documentation**

*Release v1.0-62-gf4d7d6e*

**Read the Docs**

Jul 22, 2021



---

## Contents

---

<b>1</b>	<b>Espressif DSP Library</b>	<b>1</b>
<b>2</b>	<b>Espressif DSP Library Benchmarks</b>	<b>5</b>
<b>3</b>	<b>Espressif DSP Library API Reference</b>	<b>9</b>
	<b>Index</b>	<b>35</b>



### 1.1 Overview

An Espressif DSP Library (esp-dsp) is a library of functions, modules and components that provides the possibility to use Espressif's CPUs as DSPs in an efficient way.

### 1.2 Function Naming

Naming conventions for the Library functions are similar for all covered domains. You can distinguish signal processing functions by the `dsps` prefix, while image and video processing functions have `dspi` prefix, and functions that are specific for operations on small matrices have `dspm` prefix in their names. Function names in the Library have the following general format:

```
dsp<data-domain>_<name>_<datatype1><datatype_ext>_<datatype2><datatype_ext>[_<descriptor>]<_impl>(<parameters>);
```

The elements of this format are explained in the sections that follow.

#### 1.2.1 Data-Domain

The data-domain is a single character that expresses the subset of functionality to which a given function belongs. The Library designed to support the following data-domains:

- `s` - for signals (expected data type is a 1D signal)
- `i` - for images and video (expected data type is a 2D image)
- `m` - for matrices (expected data type is a matrix)
- `r` - for realistic rendering functionality and 3D data processing (expected data type depends on supported rendering techniques)
- `q` - for signals of fixed length

For example, function names that begin with dspI signify that respective functions are used for image or video processing.

### **1.2.2 Name**

The name is an abbreviation for the core operation that the function really does, for example Add, Sqrt, followed in some cases by a function-specific modifier: = [\_modifier]

This modifier, if present, denotes a slight modification or variation of the given function.

### **1.2.3 Data Types**

The library supports two main data types – int16 for fixed point arithmetic and float for floating point arithmetic. The datatype described as:

#### **1.2.4 Data type suffices:**

- s - signed
- u - unsigned
- f - float

#### **1.2.5 Data type extensions:**

- c - complex

#### **1.2.6 Data type Bits resolution:**

- 16
- 32

For example: dsps\_mac\_sc16 defines that mac operation with 1d array will be made with 16 bit signed complex data.

### **1.2.7 Implementation Type**

Each function could be implemented different for different platform and could use different style and resources. That's why every implemented function will have name extension <\_impl> that will define which kind of implementation it is. User can use universal function without extension.

### **1.2.8 Implementation extensions:**

By default all functions could be used without extensions. The option that select optimized/ansi can be chosen in menuconfig.

**Inside library the extensions means:**

- `_ansi` - a universal function where body of function implemented on ANSI C. This implementation not includes any hardware optimization
- `_ae32` - written on ESP32 assembler and optimized for ESP32

- `_platform` - header file with definitions of available CPUs instructions for different functions
- `others`- depends on amount of supported CPUs. This list will be extended in future





## Espressif DSP Library Benchmarks

The table below contains benchmarks of functions provided by ESP-DSP library. The values are CPU cycle counts taken to execute each of the functions. The Values in the column “O2” are made with compiler optimization for size, and in the column “Os” column are made with compiler optimization for speed. The values in “ESP32” and “ESP32S3” column are for the optimized (assembly) implementation, values in “ANSI” column are for the non-optimized implementation.

Function Optimization	ESP32 O2	ESP32S3 O2	ANSI O2	ESP32 Os	ESP32S3 Os	ANSI Os
<b>Dot Product</b>						
dsps_dotprod_f32 for N=256 points	1058	589	1325	1058	448	4129
dsps_dotprod_f32 for N=256 points with step 1	1317	1325	2853	1317	1325	3621
dsps_dotprod_s16 for N=256 points	1447	325	3647	447	323	6466
<b>FIR Filters</b>						
dsps_fir_f32 1024 input samples and 256 coefficients	1079312	1074223	2150685	1079599	1074222	5147785

Continued on next page

Table 2.1 – continued from previous page

Function Optimization	ESP32 O2	ESP32S3 O2	ANSI O2	ESP32 Os	ESP32S3 Os	ANSI Os
dsps_fird_f32 1024 samples 256 coeffs and decimation 4	350915	347436	614234	350520	347607	1317367
<b>FFTs Radix-2 32 bit Floating Point</b>						
dsps_fft2r_fc32 for 64 complex points	6079	5142	7037	5452	5142	8333
dsps_fft2r_fc32 for 128 complex points	13031	11707	15907	12399	11707	19035
dsps_fft2r_fc32 for 256 complex points	27828	26303	35562	27828	26303	42922
dsps_fft2r_fc32 for 512 complex points	61753	58435	78705	61753	58435	95673
dsps_fft2r_fc32 for 1024 complex points	135742	128586	172664	135742	128585	211252
<b>FFTs Radix-4 32 bit Floating Point</b>						
dsps_fft4r_fc32 for 64 complex points	3125	3262	5185	3247	3174	5631
dsps_fft4r_fc32 for 256 complex points	15551	15450	26115	16056	15789	28397
dsps_fft4r_fc32 for 1024 complex points	75547	75185	127669	77587	76548	138522
<b>FFTs 16 bit Fixed Point</b>						
dsps_fft2r_sc16 for 64 complex points	8786	933	14575	8786	793	15861

Continued on next page

Table 2.1 – continued from previous page

Function Optimization	ESP32 O2	ESP32S3 O2	ANSI O2	ESP32 Os	ESP32S3 Os	ANSI Os
dsps_fft2r_sc16 for 128 complex points	20214	1734	33121	20214	1627	36238
dsps_fft2r_sc16 for 256 complex points	45755	3507	74290	45755	3430	81638
dsps_fft2r_sc16 for 512 complex points	102208	7312	164803	102416	7311	181758
dsps_fft2r_sc16 for 1024 complex points	225862	15641	362193	225861	15640	400853
<b>IIR Filters</b>						
dsps_biquad_f32 - biquad filter for 1024 input samples	217450	17458	24613	17451	17459	36895
<b>Matrix Multiplication</b>						
dspm_mult_f32 - C[16;16] = A[16;16]*B[16;16]	24669	6298	51502	24670	6529	78197
dspm_mult_s16 - C[16;16] = A[16;16]*B[16;16]	24707	1847	83699	24707	2047	99353
dspm_mult_3x3_f32 - C[3;1] = A[3;3]*B[3;1]	79	88	226	80	86	271
dspm_mult_3x3_f32 - C[3;3] = A[3;3]*B[3;3]	221	217	492	210	217	611
dspm_mult_4x4_f32 - C[4;1] = A[4;4]*B[4;1]	112	160	334	113	121	425
dspm_mult_4x4_f32 - C[4;4] = A[4;4]*B[4;4]	405	191	1008	404	331	1335
<b>Image processing prototypes</b>						

Continued on next page

Table 2.1 – continued from previous page

Function Optimization	ESP32 O2	ESP32S3 O2	ANSI O2	ESP32 Os	ESP32S3 Os	ANSI Os
dspi_dotprod_s - dotproduct of two im- ages 16x16	81827	179	3828	4010	178	4011
dspi_dotprod_of16x16 - dotproduct of two im- ages 16x16	41428	243	4142	4772	244	4774
dspi_dotprod_s dotproduct of two images 64x64	85809	704	58068	58825	706	58826
dspi_dotprod_of64x64 - dotproduct of two im- ages 64x64	62368	1010	62366	71233	1010	71062
dspi_dotprod_s - dotprod- uct of two images 8x8	16456	162	1453	1804	302	1806
dspi_dotprod_of8x8/u16 - dotprod- uct of two images 8x8	520	429	1531	2074	363	2074
dspi_dotprod_s - dotproduct of two im- ages 32x32	100190	424	20029	25300	425	25301
dspi_dotprod_of32x32/u16 - dotproduct of two im- ages 32x32	1000	578	21089	29432	576	29432

The benchmark test could be reproduced by executing test cases found in [test/test\\_dsp.c](#).

## 3.1 Header Files

To use the library, include `esp_dsp.h` header file into the source code.

- `modules/common/include/esp_dsp.h`

## 3.2 Signal (1D) Processing APIs

Signal processing APIs use `dsps` prefix. The following modules are available:

- *Dot-product* - Calculates dot-product of two vectors
- *FFT* - Fast Fourier Transform functionality
- *DCT* - Discrete Cosine Transform functionality
- *IIR* - IIR filter functionality
- *FIR* - FIR filter functionality
- *Math* - Basic vector operations
- *Conv* - Convolution/correlation functionality
- *Support* - Support functions
- *Window functions* - FFT window generation functions

### 3.2.1 Dot-product

#### Header File

- `modules/dotprod/include/dsps_dotprod.h`

## Functions

`esp_err_t dsps_dotprod_s16_ansi (const int16_t *src1, const int16_t *src2, int16_t *dest, int len, int8_t shift)`

dot product of two 16 bit vectors Dot product calculation for two signed 16 bit arrays: `*dest += (src1[i] * src2[i]) >> (15-shift); i= [0..N)` The extension (`_ansi`) use ANSI C and could be compiled and run on any platform. The extension (`_ae32`) is optimized for ESP32 chip.

### Return

- ESP\_OK on success
- One of the error codes from DSP library

### Parameters

- `src1`: source array 1
- `src2`: source array 2
- `dest`: destination pointer
- `len`: length of input arrays
- `shift`: shift of the result.

`esp_err_t dsps_dotprod_s16_ae32 (const int16_t *src1, const int16_t *src2, int16_t *dest, int len, int8_t shift)`

`esp_err_t dsps_dotprod_f32_ansi (const float *src1, const float *src2, float *dest, int len)`

dot product of two float vectors Dot product calculation for two floating point arrays: `*dest += (src1[i] * src2[i]); i= [0..N)` The extension (`_ansi`) use ANSI C and could be compiled and run on any platform. The extension (`_ae32`) is optimized for ESP32 chip.

### Return

- ESP\_OK on success
- One of the error codes from DSP library

### Parameters

- `src1`: source array 1
- `src2`: source array 2
- `dest`: destination pointer
- `len`: length of input arrays

`esp_err_t dsps_dotprod_f32_ae32 (const float *src1, const float *src2, float *dest, int len)`

`esp_err_t dsps_dotprod_f32_aes3 (const float *src1, const float *src2, float *dest, int len)`

`esp_err_t dsps_dotprode_f32_ansi (const float *src1, const float *src2, float *dest, int len, int step1, int step2)`

dot product of two float vectors with step Dot product calculation for two floating point arrays: `*dest += (src1[i*step1] * src2[i*step2]); i= [0..N)` The extension (`_ansi`) use ANSI C and could be compiled and run on any platform. The extension (`_ae32`) is optimized for ESP32 chip.

### Return

- ESP\_OK on success

- One of the error codes from DSP library

#### Parameters

- `src1`: source array 1
- `src2`: source array 2
- `dest`: destination pointer
- `len`: length of input arrays
- `step1`: step over elements in first array
- `step2`: step over elements in second array

`esp_err_t dsps_dotprode_f32_ae32` (`const float *src1`, `const float *src2`, `float *dest`, `int len`, `int step1`, `int step2`)

#### Macros

`dsps_dotprod_s16`

`dsps_dotprod_f32`

`dsps_dotprode_f32`

### 3.2.2 FFT

#### Header File

- `modules/fft/include/dsps_fft2r.h`

#### Functions

`esp_err_t dsps_fft2r_init_fc32` (`float *fft_table_buff`, `int table_size`)  
init fft tables

Initialization of Complex FFT. This function initialize coefficients table. The implementation use ANSI C and could be compiled and run on any platform

#### Return

- `ESP_OK` on success
- `ESP_ERR_DSP_PARAM_OUTOFRANGE` if `table_size > CONFIG_DSP_MAX_FFT_SIZE`
- `ESP_ERR_DSP_REINITIALIZED` if buffer already allocated internally by other function
- One of the error codes from DSP library

#### Parameters

- `fft_table_buff`: pointer to floating point buffer where sin/cos table will be stored if this parameter set to `NULL`, and `table_size` value is more then 0, then `dsps_fft2r_init_fc32` will allocate buffer internally
- `table_size`: size of the buffer in float words if `fft_table_buff` is `NULL` and `table_size` is not 0, buffer will be allocated internally. If `table_size` is 0, buffer will not be allocated.

esp\_err\_t **dsps\_fft2r\_init\_sc16** (int16\_t \*fft\_table\_buff, int table\_size)

void **dsps\_fft2r\_deinit\_fc32** ()  
deinit fft tables

Free resources of Complex FFT. This function delete coefficients table if it was allocated by dsps\_fft2r\_init\_fc32. The implementation use ANSI C and could be compiled and run on any platform

#### Return

void **dsps\_fft2r\_deinit\_sc16** ()

esp\_err\_t **dsps\_fft2r\_fc32\_ansi\_** (float \*data, int N, float \*w)  
complex FFT of radix 2

Complex FFT of radix 2 The extension (\_ansi) use ANSI C and could be compiled and run on any platform. The extension (\_ae32) is optimized for ESP32 chip.

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- data: input/output complex array. An elements located: Re[0], Im[0], ... Re[N-1], Im[N-1] result of FFT will be stored to this array.
- N: Number of complex elements in input array
- w: pointer to the sin/cos table

esp\_err\_t **dsps\_fft2r\_fc32\_ae32\_** (float \*data, int N, float \*w)

esp\_err\_t **dsps\_fft2r\_fc32\_aes3\_** (float \*data, int N, float \*w)

esp\_err\_t **dsps\_fft2r\_sc16\_ansi\_** (int16\_t \*data, int N, int16\_t \*w)

esp\_err\_t **dsps\_fft2r\_sc16\_ae32\_** (int16\_t \*data, int N, int16\_t \*w)

esp\_err\_t **dsps\_fft2r\_sc16\_aes3\_** (int16\_t \*data, int N, int16\_t \*w)

esp\_err\_t **dsps\_bit\_rev\_fc32\_ansi** (float \*data, int N)  
bit reverse operation for the complex input array

Bit reverse operation for the complex input array The implementation use ANSI C and could be compiled and run on any platform

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- data: input/ complex array. An elements located: Re[0], Im[0], ... Re[N-1], Im[N-1] result of FFT will be stored to this array.
- N: Number of complex elements in input array

esp\_err\_t **dsps\_bit\_rev\_sc16\_ansi** (int16\_t \*data, int N)

esp\_err\_t **dsps\_bit\_rev2r\_fc32** (float \*data, int N)



esp\_err\_t **dsps\_bit\_rev\_lookup\_fc32\_ansi** (float \*data, int reverse\_size, uint16\_t \*reverse\_tab)

esp\_err\_t **dsps\_bit\_rev\_lookup\_fc32\_ae32** (float \*data, int reverse\_size, uint16\_t \*reverse\_tab)

esp\_err\_t **dsps\_bit\_rev\_lookup\_fc32\_aes3** (float \*data, int reverse\_size, uint16\_t \*reverse\_tab)

esp\_err\_t **dsps\_gen\_w\_r2\_fc32** (float \*w, int N)

Generate coefficients table for the FFT radix 2.

Generate coefficients table for the FFT radix 2. This function called inside init. The implementation use ANSI C and could be compiled and run on any platform

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- w: memory location to store coefficients. By default coefficients will be stored to the dsps\_fft\_w\_table\_fc32. Maximum size of the FFT must be setup in menuconfig
- N: maximum size of the FFT that will be used

esp\_err\_t **dsps\_gen\_w\_r2\_sc16** (int16\_t \*w, int N)

esp\_err\_t **dsps\_cplx2reC\_fc32\_ansi** (float \*data, int N)

Convert complex array to two real arrays.

Convert complex array to two real arrays in case if input was two real arrays. This function have to be used if FFT used to process real data. The implementation use ANSI C and could be compiled and run on any platform

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- data: Input complex array and result of FFT2R. input has size of 2\*N, because contains real and imaginary part. result will be stored to the same array. Input1: input[0..N-1], Input2: input[N..2\*N-1]
- N: Number of complex elements in input array

esp\_err\_t **dsps\_cplx2reC\_sc16** (int16\_t \*data, int N)

esp\_err\_t **dsps\_cplx2real\_sc16\_ansi** (int16\_t \*data, int N)

Convert complex FFT result to real array.

Convert FFT result of complex FFT for resl input to real array. This function have to be used if FFT used to process real data. The implementation use ANSI C and could be compiled and run on any platform

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- data: Input complex array and result of FFT2R. input has size of 2\*N, because contains real and imaginary part. result will be stored to the same array. Input1: input[0..N-1], Input2: input[N..2\*N-1]

- N: Number of complex elements in input array

esp\_err\_t **dsps\_cplx2real256\_fc32\_ansi** (float \**data*)

esp\_err\_t **dsps\_gen\_bitrev2r\_table** (int *N*, int *step*, char \**name\_ext*)

## Macros

**CONFIG\_DSP\_MAX\_FFT\_SIZE**

**dsps\_fft2r\_fc32\_ae32** (data, N)

**dsps\_fft2r\_fc32\_aes3** (data, N)

**dsps\_fft2r\_sc16\_ae32** (data, N)

**dsps\_fft2r\_sc16\_aes3** (data, N)

**dsps\_fft2r\_fc32\_ansi** (data, N)

**dsps\_fft2r\_sc16\_ansi** (data, N)

**dsps\_fft2r\_fc32**

**dsps\_bit\_rev\_fc32**

**dsps\_cplx2reC\_fc32**

**dsps\_bit\_rev\_sc16**

**dsps\_bit\_rev\_lookup\_fc32**

## 3.2.3 DCT

### Header File

- `modules/dct/include/dsps_dct.h`

### Functions

esp\_err\_t **dsps\_dct\_f32** (float \**data*, int *N*)

DCT of radix 2, unscaled.

DCT type II of radix 2, unscaled Function is FFT based The extension (`_ansi`) use ANSIC and could be compiled and run on any platform. The extension (`_ae32`) is optimized for ESP32 chip.

#### Return

- `ESP_OK` on success
- One of the error codes from DSP library

#### Parameters

- *data*: input/output array with size of  $N*2$ . An elements located: `Re[0],Re[1], , ... Re[N-1]`, any data... up to  $N*2$  result of DCT will be stored to this array from  $0 \dots N-1$ . Size of data array must be  $N*2!!!$
- *N*: Size of DCT transform. Size of data array must be  $N*2!!!$

`esp_err_t dsps_dct_inv_f32` (float \**data*, int *N*)

Inverce DCT of radix 2.

Inverce DCT type III of radix 2, unscaled Function is FFT based The extension (`_ansi`) use ANSI C and could be compiled and run on any platform. The extension (`_ae32`) is optimized for ESP32 chip.

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- *data*: input/output array with size of  $N*2$ . An elements located: `Re[0],Re[1], , ... Re[N-1]`, any data... up to  $N*2$  result of DCT will be stored to this array from `0...N-1`. Size of data array must be  $N*2!!!$
- *N*: Size of DCT transform. Size of data array must be  $N*2!!!$

`esp_err_t dsps_dct_f32_ref` (float \**data*, int *N*, float \**result*)

DCTs.

Direct DCT type II and Inverce DCT type III, unscaled These functions used as a reference for general purpose. These functions are not optimyzed! The extension (`_ansi`) use ANSI C and could be compiled and run on any platform. The extension (`_ae32`) is optimized for ESP32 chip.

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- *data*: input/output array with size of *N*. An elements located: `Re[0],Re[1], , ... Re[N-1]`
- *N*: Size of DCT transform. Size of data array must be  $N*2!!!$
- *result*: output result array with size of *N*.

`esp_err_t dsps_dct_inverce_f32_ref` (float \**data*, int *N*, float \**result*)

## 3.2.4 FIR

### Header File

- `modules/fir/include/dsps_fir.h`

### Functions

`esp_err_t dsps_fir_f32_ansi` (*fir\_f32\_t* \**fir*, const float \**input*, float \**output*, int *len*)

FIR filter.

Function implements FIR filter The extension (`_ansi`) use ANSI C and could be compiled and run on any platform. The extension (`_ae32`) is optimized for ESP32 chip.

#### Return

- ESP\_OK on success

- One of the error codes from DSP library

#### Parameters

- `fir`: pointer to fir filter structure, that must be initialized before
- `input`: input array
- `output`: array with result of FIR filter
- `len`: length of input and result arrays

`esp_err_t dsps_fir_f32_ae32 (fir_f32_t *fir, const float *input, float *output, int len)`

`esp_err_t dsps_fir_f32_aes3 (fir_f32_t *fir, const float *input, float *output, int len)`

`int dsps_fird_f32_ansi (fir_f32_t *fir, const float *input, float *output, int len)`

Decimation FIR filter.

Function implements FIR filter with decimation The extension (`_ansi`) use ANSI C and could be compiled and run on any platform. The extension (`_ae32`) is optimized for ESP32 chip.

**Return** : function returns amount of samples stored to the output array depends on the previous state value could be  $[0..len/decimation]$

#### Parameters

- `fir`: pointer to fir filter structure, that must be initialized before
- `input`: input array
- `output`: array with result of FIR filter
- `len`: length of input and result arrays

`int dsps_fird_f32_ae32 (fir_f32_t *fir, const float *input, float *output, int len)`

`esp_err_t dsps_fir_init_f32 (fir_f32_t *fir, float *coeffs, float *delay, int N)`

initialize structure for 32 bit FIR filter

Function initialize structure for 32 bit floating point FIR filter The implementation use ANSI C and could be compiled and run on any platform

#### Return

- `ESP_OK` on success
- One of the error codes from DSP library

#### Parameters

- `fir`: pointer to fir filter structure, that must be preallocated
- `coeffs`: array with FIR filter coefficients. Must be length N
- `delay`: array for FIR filter delay line. Must be length N
- `N`: FIR filter length. Length of coeffs and delay arrays.

`esp_err_t dsps_fird_init_f32 (fir_f32_t *fir, float *coeffs, float *delay, int N, int decim, int start_pos)`

initialize structure for 32 bit Decimation FIR filter Function initialize structure for 32 bit floating point FIR filter with decimation The implementation use ANSI C and could be compiled and run on any platform

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- `fir`: pointer to fir filter structure, that must be preallocated
- `coeffs`: array with FIR filter coefficients. Must be length N
- `delay`: array for FIR filter delay line. Must be length N
- `N`: FIR filter length. Length of `coeffs` and `delay` arrays.
- `decim`: decimation factor.
- `start_pos`: initial value of decimation counter. Must be [0..d)

#### Structures

##### **struct fir\_f32\_s**

Data struct of f32 fir filter.

This structure used by filter internally. User should access this structure only in case of extensions for the DSP Library. All fields of this structure initialized by `dsps_fir_init_f32(...)` function.

#### Public Members

float \***coeffs**

Pointer to the coefficient buffer.

float \***delay**

Pointer to the delay line buffer.

int **N**

FIR filter coefficients amount.

int **pos**

Position in delay line.

int **decim**

Decimation factor.

int **d\_pos**

Actual decimation counter.

#### Macros

**dsps\_fir\_f32**

**dsps\_fird\_f32**

#### Type Definitions

**typedef struct *fir\_f32\_s* fir\_f32\_t**

Data struct of f32 fir filter.

This structure used by filter internally. User should access this structure only in case of extensions for the DSP Library. All fields of this structure initialized by `dsps_fir_init_f32(...)` function.

## 3.2.5 IIR

### Header File

- `modules/iir/include/dsps_biquad_gen.h`

### Functions

`esp_err_t dsps_biquad_gen_lpf_f32` (float \**coeffs*, float *f*, float *qFactor*)

LPF IIR filter coefficients Coefficients for low pass 2nd order IIR filter (bi-quad) The implementation use ANSI C and could be compiled and run on any platform.

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- *coeffs*: result coefficients. b0,b1,b2,a1,a2, a0 are not placed to the array and expected by IIR as 1
- *f*: filter cut off frequency in range of 0..0.5 (normalized to sample frequency)
- *qFactor*: Q factor of filter

`esp_err_t dsps_biquad_gen_hpf_f32` (float \**coeffs*, float *f*, float *qFactor*)

HPF IIR filter coefficients.

Coefficients for high pass 2nd order IIR filter (bi-quad) The implementation use ANSI C and could be compiled and run on any platform

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- *coeffs*: result coefficients. b0,b1,b2,a1,a2, a0 are not placed to the array and expected by IIR as 1
- *f*: filter cut off frequency in range of 0..0.5 (normalized to sample frequency)
- *qFactor*: Q factor of filter

`esp_err_t dsps_biquad_gen_bpf_f32` (float \**coeffs*, float *f*, float *qFactor*)

BPF IIR filter coefficients.

Coefficients for band pass 2nd order IIR filter (bi-quad) The implementation use ANSI C and could be compiled and run on any platform

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- *coeffs*: result coefficients. b0,b1,b2,a1,a2, a0 are not placed to the array and expected by IIR as 1

- *f*: filter center frequency in range of 0..0.5 (normalized to sample frequency)
- *qFactor*: Q factor of filter

`esp_err_t dsps_biquad_gen_bpf0db_f32` (float \**coeffs*, float *f*, float *qFactor*)  
0 dB BPF IIR filter coefficients

Coefficients for band pass 2nd order IIR filter (bi-quad) with 0 dB gain in passband The implementation use ANSI C and could be compiled and run on any platform

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- *coeffs*: result coefficients. b0,b1,b2,a1,a2, a0 are not placed to the array and expected by IIR as 1
- *f*: filter center frequency in range of 0..0.5 (normalized to sample frequency)
- *qFactor*: Q factor of filter

`esp_err_t dsps_biquad_gen_notch_f32` (float \**coeffs*, float *f*, float *gain*, float *qFactor*)  
Notch IIR filter coefficients.

Coefficients for notch 2nd order IIR filter (bi-quad) The implementation use ANSI C and could be compiled and run on any platform

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- *coeffs*: result coefficients. b0,b1,b2,a1,a2, a0 are not placed to the array and expected by IIR as 1
- *f*: filter notch frequency in range of 0..0.5 (normalized to sample frequency)
- *gain*: gain in stopband in dB
- *qFactor*: Q factor of filter

`esp_err_t dsps_biquad_gen_allpass360_f32` (float \**coeffs*, float *f*, float *qFactor*)  
Allpass 360 degree IIR filter coefficients.

Coefficients for all pass 2nd order IIR filter (bi-quad) with 360 degree phase shift The implementation use ANSI C and could be compiled and run on any platform

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- *coeffs*: result coefficients. b0,b1,b2,a1,a2, a0 are not placed to the array and expected by IIR as 1
- *f*: filter notch frequency in range of 0..0.5 (normalized to sample frequency)
- *qFactor*: Q factor of filter

`esp_err_t dsps_biquad_gen_allpass180_f32` (float \*coeffs, float f, float qFactor)

Allpass 180 degree IIR filter coefficients.

Coefficients for all pass 2nd order IIR filter (bi-quad) with 180 degree phase shift The implementation use ANSI C and could be compiled and run on any platform

**Return**

- ESP\_OK on success
- One of the error codes from DSP library

**Parameters**

- coeffs: result coefficients. b0,b1,b2,a1,a2, a0 are not placed to the array and expected by IIR as 1
- f: filter notch frequency in range of 0..0.5 (normalized to sample frequency)
- qFactor: Q factor of filter

`esp_err_t dsps_biquad_gen_peakingEQ_f32` (float \*coeffs, float f, float qFactor)

peak IIR filter coefficients

Coefficients for peak 2nd order IIR filter (bi-quad) The implementation use ANSI C and could be compiled and run on any platform

**Return**

- ESP\_OK on success
- One of the error codes from DSP library

**Parameters**

- coeffs: result coefficients. b0,b1,b2,a1,a2, a0 are not placed to the array and expected by IIR as 1
- f: filter notch frequency in range of 0..0.5 (normalized to sample frequency)
- qFactor: Q factor of filter

`esp_err_t dsps_biquad_gen_lowShelf_f32` (float \*coeffs, float f, float gain, float qFactor)

low shelf IIR filter coefficients

Coefficients for low pass Shelf 2nd order IIR filter (bi-quad) The implementation use ANSI C and could be compiled and run on any platform

**Return**

- ESP\_OK on success
- One of the error codes from DSP library

**Parameters**

- coeffs: result coefficients. b0,b1,b2,a1,a2, a0 are not placed to the array and expected by IIR as 1
- f: filter notch frequency in range of 0..0.5 (normalized to sample frequency)
- gain: gain in stopband in dB
- qFactor: Q factor of filter



`esp_err_t dsps_biquad_gen_highShelf_f32` (`float *coeffs`, `float f`, `float gain`, `float qFactor`)  
 high shelf IIR filter coefficients

Coefficients for high pass Shelf 2nd order IIR filter (bi-quad) The implementation use ANSI C and could be compiled and run on any platform

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- `coeffs`: result coefficients. `b0,b1,b2,a1,a2`, `a0` are not placed to the array and expected by IIR as 1
- `f`: filter notch frequency in range of 0..0.5 (normalized to sample frequency)
- `gain`: gain in stopband in dB
- `qFactor`: Q factor of filter

#### Header File

- `modules/iir/include/dsps_biquad.h`

#### Functions

`esp_err_t dsps_biquad_f32_ansi` (`const float *input`, `float *output`, `int len`, `float *coef`, `float *w`)  
 IIR filter.

IIR filter 2nd order direct form II (bi quad) The extension (`_ansi`) use ANSI C and could be compiled and run on any platform. The extension (`_ae32`) is optimized for ESP32 chip.

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- `input`: input array
- `output`: output array
- `len`: length of input and output vectors
- `coef`: array of coefficients. `b0,b1,b2,a1,a2` expected that `a0 = 1`. `b0..b2` - numerator, `a0..a2` - denominator
- `w`: delay line `w0,w1`. Length of 2.

`esp_err_t dsps_biquad_f32_ae32` (`const float *input`, `float *output`, `int len`, `float *coef`, `float *w`)

`esp_err_t dsps_biquad_f32_aes3` (`const float *input`, `float *output`, `int len`, `float *coef`, `float *w`)

#### Macros

`dsps_biquad_f32`

## 3.2.6 Math

### Header File

- `modules/math/add/include/dsps_add.h`

### Functions

`esp_err_t dsps_add_f32_ansi` (**const** float \**input1*, **const** float \**input2*, float \**output*, int *len*, int *step1*, int *step2*, int *step\_out*)

add two arrays

The function add one input array to another  $out[i*step\_out] = input1[i*step1] + input2[i*step2]$ ;  $i=[0..len)$  The implementation use ANSI C and could be compiled and run on any platform

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- *input1*: input array 1
- *input2*: input array 2
- *output*: output array
- *len*: amount of operations for arrays
- *step1*: step over input array 1 (by default should be 1)
- *step2*: step over input array 2 (by default should be 1)
- *step\_out*: step over output array (by default should be 1)

`esp_err_t dsps_add_f32_ae32` (**const** float \**input1*, **const** float \**input2*, float \**output*, int *len*, int *step1*, int *step2*, int *step\_out*)

`esp_err_t dsps_add_s16_ansi` (**const** int16\_t \**input1*, **const** int16\_t \**input2*, int16\_t \**output*, int *len*, int *step1*, int *step2*, int *step\_out*, int *shift*)

`esp_err_t dsps_add_s16_ae32` (**const** int16\_t \**input1*, **const** int16\_t \**input2*, int16\_t \**output*, int *len*, int *step1*, int *step2*, int *step\_out*, int *shift*)

### Macros

`dsps_add_f32`

`dsps_add_s16`

### Header File

- `modules/math/sub/include/dsps_sub.h`

## Functions

`esp_err_t dsps_sub_f32_ansi` (`const float *input1`, `const float *input2`, `float *output`, `int len`, `int step1`, `int step2`, `int step_out`)

sub arrays

The function subtract one array from another  $out[i*step\_out] = input1[i*step1] - input2[i*step2]$ ;  $i=[0..len)$  The implementation use ANSI C and could be compiled and run on any platform

### Return

- ESP\_OK on success
- One of the error codes from DSP library

### Parameters

- `input1`: input array 1
- `input2`: input array 2
- `output`: output array
- `len`: amount of operations for arrays
- `step1`: step over input array 1 (by default should be 1)
- `step2`: step over input array 2 (by default should be 1)
- `step_out`: step over output array (by default should be 1)

`esp_err_t dsps_sub_f32_ae32` (`const float *input1`, `const float *input2`, `float *output`, `int len`, `int step1`, `int step2`, `int step_out`)

## Macros

`dsps_sub_f32`

## Header File

- `modules/math/mul/include/dsps_mul.h`

## Functions

`esp_err_t dsps_mul_f32_ansi` (`const float *input1`, `const float *input2`, `float *output`, `int len`, `int step1`, `int step2`, `int step_out`)

Multiply two arrays.

The function multiply one input array to another and store result to other array  $out[i*step\_out] = input1[i*step1] * input2[i*step2]$ ;  $i=[0..len)$  The implementation use ANSI C and could be compiled and run on any platform

### Return

- ESP\_OK on success
- One of the error codes from DSP library

### Parameters

- `input1`: input array 1

- `input2`: input array 2
- `output`: output array
- `len`: amount of operations for arrays
- `step1`: step over input array 1 (by default should be 1)
- `step2`: step over input array 2 (by default should be 1)
- `step_out`: step over output array (by default should be 1)

`esp_err_t dsps_mul_f32_ae32 (const float *input1, const float *input2, float *output, int len, int step1, int step2, int step_out)`

`esp_err_t dsps_mul_s16_ansi (const int16_t *input1, const int16_t *input2, int16_t *output, int len, int step1, int step2, int step_out, int shift)`

Multiply two arrays.

The function multiply one input array to another and store result to other array `out[i*step_out] = input1[i*step1] * input2[i*step2]`; `i=[0..len)` The implementation use ANSI C and could be compiled and run on any platform

#### Return

- `ESP_OK` on success
- One of the error codes from DSP library

#### Parameters

- `input1`: input array 1
- `input2`: input array 2
- `output`: output array
- `len`: amount of operations for arrays
- `step1`: step over input array 1 (by default should be 1)
- `step2`: step over input array 2 (by default should be 1)
- `step_out`: step over output array (by default should be 1)
- `shift`: output shift after multiplication (by default should be 15)

#### Macros

`dsps_mul_f32`

`dsps_mul_s16`

#### Header File

- `modules/math/addc/include/dsps_addc.h`

## Functions

`esp_err_t dsps_addc_f32_ansi` (`const float *input`, `float *output`, `int len`, `float C`, `int step_in`, `int step_out`)

add constant

The function adds constant to the input array  $x[i*step\_out] = y[i*step\_in] + C$ ;  $i=[0..len)$  The implementation use ANSI C and could be compiled and run on any platform

### Return

- ESP\_OK on success
- One of the error codes from DSP library

### Parameters

- `input`: input array
- `output`: output array
- `len`: amount of operations for arrays
- `C`: constant value
- `step_in`: step over input array (by default should be 1)
- `step_out`: step over output array (by default should be 1)

`esp_err_t dsps_addc_f32_ae32` (`const float *input`, `float *output`, `int len`, `float C`, `int step_in`, `int step_out`)

## Macros

`dsps_addc_f32`

## Header File

- `modules/math/mulc/include/dsps_mulc.h`

## Functions

`esp_err_t dsps_mulc_f32_ansi` (`const float *input`, `float *output`, `int len`, `float C`, `int step_in`, `int step_out`)

multiply constant

The function multiplies input array to the constant value  $x[i*step\_out] = y[i*step\_in]*C$ ;  $i=[0..len)$  The implementation use ANSI C and could be compiled and run on any platform

### Return

- ESP\_OK on success
- One of the error codes from DSP library

### Parameters

- `input`: input array
- `output`: output array

- `len`: amount of operations for arrays
- `C`: constant value
- `step_in`: step over input array (by default should be 1)
- `step_out`: step over output array (by default should be 1)

`esp_err_t dsps_mulc_f32_ae32` (`const float *input`, `float *output`, `int len`, `float C`, `int step_in`, `int step_out`)

`esp_err_t dsps_mulc_s16_ae32` (`const int16_t *input`, `int16_t *output`, `int len`, `int16_t C`, `int step_in`, `int step_out`)

`esp_err_t dsps_mulc_s16_ansi` (`const int16_t *input`, `int16_t *output`, `int len`, `int16_t C`, `int step_in`, `int step_out`)

## Macros

`dsps_mulc_f32`

`dsps_mulc_s16`

## 3.2.7 Conv

### Header File

- `modules/conv/include/dsps_conv.h`

### Functions

`esp_err_t dsps_conv_f32_ae32` (`const float *Signal`, `const int siglen`, `const float *Kernel`, `const int kernlen`, `float *convout`)

Convolution.

The function convolve `Signal` array with `Kernel` array. The implementation use ANSI C and could be compiled and run on any platform

#### Return

- `ESP_OK` on success
- One of the error codes from DSP library

#### Parameters

- `Signal`: input array with signal
- `siglen`: length of the input signal
- `Kernel`: input array with convolution kernel
- `kernlen`: length of the `Kernel` array
- `convout`: output array with convolution result length of (`siglen` + `Kernel` -1)

`esp_err_t dsps_conv_f32_ansi` (`const float *Signal`, `const int siglen`, `const float *Kernel`, `const int kernlen`, `float *convout`)

## Macros

```
dsps_conv_f32  
    }@
```

## Header File

- `modules/conv/include/dsps_corr.h`

## Functions

```
esp_err_t dsps_corr_f32_ansi (const float *Signal, const int siglen, const float *Pattern, const  
    int patlen, float *dest)
```

Correlation with pattern.

The function correlate input sigla array with pattern array. The implementation use ANSI C and could be compiled and run on any platform

### Return

- ESP\_OK on success
- One of the error codes from DSP library (one of the input array are NULL, or if ( $\text{siglen} < \text{patlen}$ ))

### Parameters

- *Signal*: input array with signal values
- *siglen*: length of the signal array
- *Pattern*: input array with pattern values
- *patlen*: length of the pattern array. The *siglen* must be bigger then *patlen*!
- *dest*: output array with result of correlation

```
esp_err_t dsps_corr_f32_ae32 (const float *Signal, const int siglen, const float *Pattern, const  
    int patlen, float *dest)
```

## Macros

```
dsps_corr_f32  
    }@
```

## 3.2.8 Support

### Header File

- `modules/support/include/dsps_d_gen.h`

## Functions

esp\_err\_t **dsps\_d\_gen\_f32** (float \**output*, int *len*, int *pos*)  
delta function

The function generate delta function.  $output[i]=0$ , if  $i=[0..N)$   $output[i]=1$ , if  $i=pos$ , pos:  $[0..N-1)$  The implementation use ANSI C and could be compiled and run on any platform

### Return

- ESP\_OK on success
- One of the error codes from DSP library

### Parameters

- *output*: output array.
- *len*: length of the input signal
- *pos*: delta function position

## Header File

- `modules/support/include/dsps_h_gen.h`

## Functions

esp\_err\_t **dsps\_h\_gen\_f32** (float \**output*, int *len*, int *pos*)  
Heviside function.

The Heviside function.  $output[i]=0$ , if  $i=[0..pos)$   $output[i]=1$ , if  $i=[pos..N)$  The implementation use ANSI C and could be compiled and run on any platform

### Return

- ESP\_OK on success
- One of the error codes from DSP library

### Parameters

- *output*: output array.
- *len*: length of the input signal
- *pos*: heviside function position

## Header File

- `modules/support/include/dsps_tone_gen.h`

## Functions

esp\_err\_t **dsps\_tone\_gen\_f32** (float \**output*, int *len*, float *Ampl*, float *freq*, float *phase*)  
tone

The function generate a tone signal.  $x[i]=A*\sin(2*PI*i + ph/180*PI)$  The implementation use ANSI C and could be compiled and run on any platform



**Return**

- ESP\_OK on success
- One of the error codes from DSP library

**Parameters**

- `output`: output array.
- `len`: length of the input signal
- `Ampl`: amplitude
- `freq`: Naiquist frequency -1..1
- `phase`: phase in degree

**Header File**

- `modules/support/include/dsps_view.h`

**Functions**

void **dsps\_view**(**const** float \**data*, int32\_t *len*, int *width*, int *height*, float *min*, float *max*, char *view\_char*)  
plot view

Generic view function. This function takes input samples and show then in console view as a plot. The main purpose to give and draft debug information to the DSP developer.

**Parameters**

- `data`: array with input samples.
- `len`: length of the input array
- `width`: plot width in symbols
- `height`: plot height in lines
- `min`: minimum value that will be limited by Axis Y.
- `max`: maximum value that will be limited by Axis Y.
- `view_char`: character to draw the plot values ('.' or 'l' etc)

void **dsps\_view\_s16**(**const** int16\_t \**data*, int32\_t *len*, int *width*, int *height*, float *min*, float *max*, char *view\_char*)

void **dsps\_view\_spectrum**(**const** float \**data*, int32\_t *len*, float *min*, float *max*)  
spectrum view

The view function to show spectrum values in 64x10 screen. The function based on `dsps_view`.

**Parameters**

- `data`: array with input samples.
- `len`: length of the input array
- `min`: minimum value that will be limited by Axis Y.
- `max`: maximum value that will be limited by Axis Y.

## Header File

- `modules/support/include/dsps_snr.h`

## Functions

float **dsps\_snr\_f32** (**const** float \**input*, int32\_t *len*, uint8\_t *use\_dc*)  
SNR.

The function calculates signal to noise ration in case if signal is sine tone. The function makes FFT of the input, then search a spectrum maximum, and then calculated SNR as sum of all harmonics to the maximum value. This function have to be used for debug and unit tests only. It's not optimized for real-time processing. The implementation use ANSI C and could be compiled and run on any platform

### Return

- SNR in dB

### Parameters

- *input*: input array.
- *len*: length of the input signal
- *use\_dc*: this parameter define will be DC value used for calculation or not. 0 - SNR will not include DC power 1 - SNR will include DC power

float **dsps\_snr\_fc32** (**const** float \**input*, int32\_t *len*)

## Header File

- `modules/support/include/dsps_sfdr.h`

## Functions

float **dsps\_sfdr\_f32** (**const** float \**input*, int32\_t *len*, int8\_t *use\_dc*)  
SFDR.

The function calculates Spurious-Free Dynamic Range. The function makes FFT of the input, then search a spectrum maximum, and then compare maximum value with all others. Result calculated as minimum value. This function have to be used for debug and unit tests only. It's not optimized for real-time processing. The implementation use ANSI C and could be compiled and run on any platform

### Return

- SFDR in DB

### Parameters

- *input*: input array.
- *len*: length of the input signal
- *use\_dc*: this parameter define will be DC value used for calculation or not. 0 - SNR will not include DC power 1 - SNR will include DC power

float **dsps\_sfdr\_fc32** (**const** float \**input*, int32\_t *len*)

## 3.2.9 Window Functions

### Header File

- `modules/windows/hann/include/dsps_wind_hann.h`

### Functions

void **dsps\_wind\_hann\_f32** (float \**window*, int *len*)  
Hann window.

The function generates Hann window.

#### Parameters

- *window*: buffer to store window array.
- *len*: length of the window array

## 3.3 Matrix Operations APIs

Matrix operations APIs use `dspm` prefix. The following modules are available:

- Multiplication - basic matrix multiplication operations

### 3.3.1 Matrix Multiplication

#### Header File

- `modules/matrix/include/dspm_mult.h`

#### Functions

esp\_err\_t **dspm\_mult\_f32\_ansi** (**const** float \**A*, **const** float \**B*, float \**C*, int *m*, int *n*, int *k*)  
Matrix multiplication.

Matrix multiplication for two floating point matrices:  $C[m][k] = A[m][n] * B[n][k]$  The extension (`_ansi`) use ANSI C and could be compiled and run on any platform. The extension (`_ae32`) is optimized for ESP32 chip.

#### Return

- `ESP_OK` on success
- One of the error codes from DSP library

#### Parameters

- *A*: input matrix  $A[m][n]$
- *B*: input matrix  $B[n][k]$
- *C*: result matrix  $C[m][k]$
- *m*: matrix dimension
- *n*: matrix dimension

- k: matrix dimension

`esp_err_t dspm_mult_f32_ae32 (const float *A, const float *B, float *C, int m, int n, int k)`

`esp_err_t dspm_mult_f32_aes3 (const float *A, const float *B, float *C, int m, int n, int k)`

`esp_err_t dspm_mult_s16_ansi (const int16_t *A, const int16_t *B, int16_t *C, int m, int n, int k, int  
shift)`

Matrix multiplication 16 bit signed int.

Matrix multiplication for two signed 16 bit fixed point matrices:  $C[m][k] = (A[m][n] * B[n][k]) \gg (15 - \text{shift})$   
The extension (`_ansi`) use ANSI C and could be compiled and run on any platform. The extension (`_ae32`) is optimized for ESP32 chip.

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- A: input matrix A[m][n]
- B: input matrix B[n][k]
- C: result matrix C[m][k]
- m: matrix dimension
- n: matrix dimension
- k: matrix dimension
- `shift`: every result will be shifted and stored as 16 bit signed value.

`esp_err_t dspm_mult_s16_ae32 (const int16_t *A, const int16_t *B, int16_t *C, int m, int n, int k, int  
shift)`

`esp_err_t dspm_mult_s16_aes3 (const int16_t *A, const int16_t *B, int16_t *C, int m, int n, int k, int  
shift)`

`esp_err_t dspm_mult_3x3x1_f32_ae32 (const float *A, const float *B, float *C)`

Matrix multiplication  $A[3 \times 3] \times B[3 \times 1]$ .

Matrix multiplication for two floating point matrices 3x3 and 3x1:  $C[1][3] = A[3][3] * B[3][1]$  The implementation is optimized for ESP32 chip.

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- A: input matrix A[3][3]
- B: input matrix/vector B[3][1]
- C: result matrix/vector C[3][3]

`esp_err_t dspm_mult_3x3x3_f32_ae32 (const float *A, const float *B, float *C)`

Matrix multiplication  $A[3 \times 3] \times B[3 \times 3]$ .

Matrix multiplication for two square 3x3 floating point matrices:  $C[3][3] = A[3][3] * B[3][3]$  The implementation is optimized for ESP32 chip.

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- A: input matrix A[3][3]
- B: input matrix B[3][3]
- C: result matrix C[3][3]

`esp_err_t dspm_mult_4x4x1_f32_ae32 (const float *A, const float *B, float *C)`

Matrix multiplication  $A[4x4]xB[4x1]$ .

Matrix multiplication for two floating point matrices 4x4 and 4x1:  $C[1][4] = A[4][4] * B[4][1]$  The implementation is optimized for ESP32 chip.

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- A: input matrix A[4][4]
- B: input matrix/vector B[4][1]
- C: result matrix/vector C[4][4]

`esp_err_t dspm_mult_4x4x4_f32_ae32 (const float *A, const float *B, float *C)`

Matrix multiplication  $A[4x4]xB[4x4]$ .

Matrix multiplication for two square 4x4 floating point matrices:  $C[4][4] = A[4][4] * B[4][4]$  The implementation is optimized for ESP32 chip.

#### Return

- ESP\_OK on success
- One of the error codes from DSP library

#### Parameters

- A: input matrix A[4][4]
- B: input matrix B[4][4]
- C: result matrix C[4][4]

## Macros

`dspm_mult_s16`

`dspm_mult_f32`

`dspm_mult_3x3x1_f32 (A, B, C)`

`dsps_sub_f32`

`dsps_add_f32`

`dspm_mult_4x4x4_f32` (A, B, C)

## 3.3.2 Matrix Operations

### Header File

- `modules/matrix/include/mat.h`

## 3.4 Miscellaneous

Various common functions used by other modules are included in this module.

### 3.4.1 Common APIs

#### Header File

- `modules/common/include/dsp_common.h`

#### Functions

bool `dsp_is_power_of_two` (int *x*)

check power of two The function check if the argument is power of 2. The implementation use ANSI C and could be compiled and run on any platform

#### Return

- true if *x* is power of two
- false if no

int `dsp_power_of_two` (int *x*)

Power of two The function return power of 2 for values  $2^N$ . The implementation use ANSI C and could be compiled and run on any platform.

#### Return

- power of two

#### Macros

`dsp_get_cpu_cycle_count`

## C

CONFIG\_DSP\_MAX\_FFT\_SIZE (C macro), 14

## D

dsp\_get\_cpu\_cycle\_count (C macro), 34  
 dsp\_is\_power\_of\_two (C++ function), 34  
 dsp\_power\_of\_two (C++ function), 34  
 dspm\_mult\_3x3x1\_f32 (C macro), 33  
 dspm\_mult\_3x3x1\_f32\_ae32 (C++ function), 32  
 dspm\_mult\_3x3x3\_f32\_ae32 (C++ function), 32  
 dspm\_mult\_4x4x1\_f32\_ae32 (C++ function), 33  
 dspm\_mult\_4x4x4\_f32 (C macro), 34  
 dspm\_mult\_4x4x4\_f32\_ae32 (C++ function), 33  
 dspm\_mult\_f32 (C macro), 33  
 dspm\_mult\_f32\_ae32 (C++ function), 32  
 dspm\_mult\_f32\_aes3 (C++ function), 32  
 dspm\_mult\_f32\_ansi (C++ function), 31  
 dspm\_mult\_s16 (C macro), 33  
 dspm\_mult\_s16\_ae32 (C++ function), 32  
 dspm\_mult\_s16\_aes3 (C++ function), 32  
 dspm\_mult\_s16\_ansi (C++ function), 32  
 dsps\_add\_f32 (C macro), 22, 34  
 dsps\_add\_f32\_ae32 (C++ function), 22  
 dsps\_add\_f32\_ansi (C++ function), 22  
 dsps\_add\_s16 (C macro), 22  
 dsps\_add\_s16\_ae32 (C++ function), 22  
 dsps\_add\_s16\_ansi (C++ function), 22  
 dsps\_addc\_f32 (C macro), 25  
 dsps\_addc\_f32\_ae32 (C++ function), 25  
 dsps\_addc\_f32\_ansi (C++ function), 25  
 dsps\_biquad\_f32 (C macro), 21  
 dsps\_biquad\_f32\_ae32 (C++ function), 21  
 dsps\_biquad\_f32\_aes3 (C++ function), 21  
 dsps\_biquad\_f32\_ansi (C++ function), 21  
 dsps\_biquad\_gen\_allpass180\_f32 (C++ function), 19  
 dsps\_biquad\_gen\_allpass360\_f32 (C++ function), 19  
 dsps\_biquad\_gen\_bpf0db\_f32 (C++ function), 19  
 dsps\_biquad\_gen\_bpf\_f32 (C++ function), 18  
 dsps\_biquad\_gen\_highShelf\_f32 (C++ function), 20

dsps\_biquad\_gen\_hpf\_f32 (C++ function), 18  
 dsps\_biquad\_gen\_lowShelf\_f32 (C++ function), 20  
 dsps\_biquad\_gen\_lpf\_f32 (C++ function), 18  
 dsps\_biquad\_gen\_notch\_f32 (C++ function), 19  
 dsps\_biquad\_gen\_peakingEQ\_f32 (C++ function), 20  
 dsps\_bit\_rev2r\_fc32 (C++ function), 12  
 dsps\_bit\_rev\_fc32 (C macro), 14  
 dsps\_bit\_rev\_fc32\_ansi (C++ function), 12  
 dsps\_bit\_rev\_lookup\_fc32 (C macro), 14  
 dsps\_bit\_rev\_lookup\_fc32\_ae32 (C++ function), 13  
 dsps\_bit\_rev\_lookup\_fc32\_aes3 (C++ function), 13  
 dsps\_bit\_rev\_lookup\_fc32\_ansi (C++ function), 12  
 dsps\_bit\_rev\_sc16 (C macro), 14  
 dsps\_bit\_rev\_sc16\_ansi (C++ function), 12  
 dsps\_conv\_f32 (C macro), 27  
 dsps\_conv\_f32\_ae32 (C++ function), 26  
 dsps\_conv\_f32\_ansi (C++ function), 26  
 dsps\_corr\_f32 (C macro), 27  
 dsps\_corr\_f32\_ae32 (C++ function), 27  
 dsps\_corr\_f32\_ansi (C++ function), 27  
 dsps\_cplx2real256\_fc32\_ansi (C++ function), 14  
 dsps\_cplx2real\_sc16\_ansi (C++ function), 13  
 dsps\_cplx2reC\_fc32 (C macro), 14  
 dsps\_cplx2reC\_fc32\_ansi (C++ function), 13  
 dsps\_cplx2reC\_sc16 (C++ function), 13  
 dsps\_d\_gen\_f32 (C++ function), 28  
 dsps\_dct\_f32 (C++ function), 14  
 dsps\_dct\_f32\_ref (C++ function), 15  
 dsps\_dct\_inv\_f32 (C++ function), 14  
 dsps\_dct\_inverce\_f32\_ref (C++ function), 15  
 dsps\_dotprod\_f32 (C macro), 11  
 dsps\_dotprod\_f32\_ae32 (C++ function), 10  
 dsps\_dotprod\_f32\_aes3 (C++ function), 10  
 dsps\_dotprod\_f32\_ansi (C++ function), 10  
 dsps\_dotprod\_s16 (C macro), 11  
 dsps\_dotprod\_s16\_ae32 (C++ function), 10  
 dsps\_dotprod\_s16\_ansi (C++ function), 10  
 dsps\_dotprode\_f32 (C macro), 11  
 dsps\_dotprode\_f32\_ae32 (C++ function), 11  
 dsps\_dotprode\_f32\_ansi (C++ function), 10

dsps\_fft2r\_deinit\_fc32 (C++ function), 12  
 dsps\_fft2r\_deinit\_sc16 (C++ function), 12  
 dsps\_fft2r\_fc32 (C macro), 14  
 dsps\_fft2r\_fc32\_ae32 (C macro), 14  
 dsps\_fft2r\_fc32\_ae32\_ (C++ function), 12  
 dsps\_fft2r\_fc32\_aes3 (C macro), 14  
 dsps\_fft2r\_fc32\_aes3\_ (C++ function), 12  
 dsps\_fft2r\_fc32\_ansi (C macro), 14  
 dsps\_fft2r\_fc32\_ansi\_ (C++ function), 12  
 dsps\_fft2r\_init\_fc32 (C++ function), 11  
 dsps\_fft2r\_init\_sc16 (C++ function), 11  
 dsps\_fft2r\_sc16\_ae32 (C macro), 14  
 dsps\_fft2r\_sc16\_ae32\_ (C++ function), 12  
 dsps\_fft2r\_sc16\_aes3 (C macro), 14  
 dsps\_fft2r\_sc16\_aes3\_ (C++ function), 12  
 dsps\_fft2r\_sc16\_ansi (C macro), 14  
 dsps\_fft2r\_sc16\_ansi\_ (C++ function), 12  
 dsps\_fir\_f32 (C macro), 17  
 dsps\_fir\_f32\_ae32 (C++ function), 16  
 dsps\_fir\_f32\_aes3 (C++ function), 16  
 dsps\_fir\_f32\_ansi (C++ function), 15  
 dsps\_fir\_init\_f32 (C++ function), 16  
 dsps\_fird\_f32 (C macro), 17  
 dsps\_fird\_f32\_ae32 (C++ function), 16  
 dsps\_fird\_f32\_ansi (C++ function), 16  
 dsps\_fird\_init\_f32 (C++ function), 16  
 dsps\_gen\_bitrev2r\_table (C++ function), 14  
 dsps\_gen\_w\_r2\_fc32 (C++ function), 13  
 dsps\_gen\_w\_r2\_sc16 (C++ function), 13  
 dsps\_h\_gen\_f32 (C++ function), 28  
 dsps\_mul\_f32 (C macro), 24  
 dsps\_mul\_f32\_ae32 (C++ function), 24  
 dsps\_mul\_f32\_ansi (C++ function), 23  
 dsps\_mul\_s16 (C macro), 24  
 dsps\_mul\_s16\_ansi (C++ function), 24  
 dsps\_mulc\_f32 (C macro), 26  
 dsps\_mulc\_f32\_ae32 (C++ function), 26  
 dsps\_mulc\_f32\_ansi (C++ function), 25  
 dsps\_mulc\_s16 (C macro), 26  
 dsps\_mulc\_s16\_ae32 (C++ function), 26  
 dsps\_mulc\_s16\_ansi (C++ function), 26  
 dsps\_sfdr\_f32 (C++ function), 30  
 dsps\_sfdr\_fc32 (C++ function), 30  
 dsps\_snr\_f32 (C++ function), 30  
 dsps\_snr\_fc32 (C++ function), 30  
 dsps\_sub\_f32 (C macro), 23, 33  
 dsps\_sub\_f32\_ae32 (C++ function), 23  
 dsps\_sub\_f32\_ansi (C++ function), 23  
 dsps\_tone\_gen\_f32 (C++ function), 28  
 dsps\_view (C++ function), 29  
 dsps\_view\_s16 (C++ function), 29  
 dsps\_view\_spectrum (C++ function), 29  
 dsps\_wind\_hann\_f32 (C++ function), 31

## F

fir\_f32\_s (C++ class), 17  
 fir\_f32\_s::coeffs (C++ member), 17  
 fir\_f32\_s::d\_pos (C++ member), 17  
 fir\_f32\_s::decim (C++ member), 17  
 fir\_f32\_s::delay (C++ member), 17  
 fir\_f32\_s::N (C++ member), 17  
 fir\_f32\_s::pos (C++ member), 17  
 fir\_f32\_t (C++ type), 17