
AVR32765: AVR32 DSPLib Reference Manual

1 Introduction

The AVR[®]32 DSP Library is a compilation of digital signal processing functions. All function availables in the DSP Library, from the AVR32 Software Framework. All the source code (C code and assembly optimized), software example and GCC and IAR[™] projects are released in the AVR32 UC3 Software Framework.

The Atmel AVR32 DSPLib is a library which provides a useful set of digital signal processing functions. All the source code is available and fully compatible with GCC and IAR. This library provides optimized functions for the AT32UC family but also target-independent functions. This document is a listing of the functions currently implemented in the DSP Library and available in the AVR32 Software Framework.

2 Reference

- **AT32UC3 Series Software Framework:**

http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4192

- **AVR32718 : AT32UC3 Series Software Framework DSPLib:**

http://www.atmel.com/dyn/resources/prod_documents/doc32076.pdf



**32-bit AVR[®]
Microcontrollers**

Application Note

Rev. 32120A-AVR32-04/09



3 Overview

3.1 What can I find in this document?

Following are the main sections available in this document.

- Basic APIs: Filtering, Operators, Signal generation, Transforms, Vectors and Windowing.
- Advanced APIs: Advanced.
- Data: Types, Macros and Constants.
- Configuration options: Optimization options, Q Format and Constant tables.
- A list of all functions available in the DSPLib (DSPLib functions' list).

3.2 Architecture

The DSPLib is divided in 2 parts:

- The basic library: containing the basic functions which are sorted in sub-groups: filters, windows, vector management functions, operators, transforms and signal generation functions.
- The advanced library: built on the first one and which contains more complex functions such as ADCPM encoder/decoder.

Here is the directory architecture of the DSPLib (See /SERVICES/DSPLIB in the AVR32 Software Framework):



Directory Architecture of the DSPLib

- The AT32UC directory includes all the source code of the optimized functions available for the AT32UC targets and provided by the DSPLib (mainly assembly coded).
- The DOC directory includes the main documentations of the whole library.
- The EXAMPLES directory regroups a lot of examples showing a way to use the functions provided by both advanced and basic libraries.
- The GENERIC directory regroups all the target-independent source code of the DSPLib.
- The INCLUDE directory includes all the common include files for the optimized functions and the generic functions.
- The UTILS directory regroups useful tools and scripts for the DSP library.
- The readme.html file is the entry point of the HTML documentation of the library. It gives accesses to doxygen documentations and many others useful data about functions, utilities, scripts, etc.

3.3 Fixed Point Format

All DSP types in the DSPLib include the notion of 16-bit and 32-bit fixed-point formats. It is important to understand this format in order to fastest and/or debug an application. See Q Format for more details on this format.

3.4 Naming Convention

Each function in the basic library of the DSPLib follows this naming convention rule: dspX_Y_Z(...) Where:

- X is the type used (16 or 32)
- Y is category of the library (op, vect, trans, ...)
- Z is the short name of the function (cos, mul, ...)

Example :

```
dsp16_vect_mul is a function part of the DSP library. It works with 16-bit values and is used to multiply vectors.
```

3.5 Compilation Options

The DSPLIB is made to fit the best the user needs. Therefore, many configuration constants can be set at the compilation of the library. Here is the list :

- **DSP_OPTIMIZATION** : used to optimize the algorithmic of the functions. This optimization can be for speed, size, accuracy or size and accuracy. It can be set in addition to the one provided by the compiler.
- **FORCE_ALL_GENERICS** : to use only the generic (target-independant) functions when you compile the library.
- **FORCE_GENERIC_DSP16_Z** : to use the 16-bit generic version of a function. The specified function is defined by replacing the Z letter with the short name of the function. See Naming Convention for more details on the short name of a function.
- **FORCE_GENERIC_DSP32_Z** : to use the 32-bit generic version of a function. The specified function is defined by replacing the Z letter with the short name of the function. See Naming Convention for more details on the short name of a function.
- **DSP16_FORMAT**: to specify the precision of the 16-bit Q format number used. (ie: to use Q3.13 numbers, define DSP16_FORMAT with the value 13).
- **DSP32_FORMAT**: to specify the precision of the 32-bit Q format number used. (ie: to use Q10.22 numbers, define DSP32_FORMAT with the value 22).

All of these defines can be passed to the preprocessor at the compilation.

Example:

```
use -D DSP16_FORMAT=12 in command line with GCC to use formatted Q4.12 numbers.
```





3.6 Deliveries

Here are the possibilities to use the DSP Library:

- The standalone library contained in the AVR32765.zip file.
- The source code of the inside the Software Framework.
- The AVR32 Studio Software Framework plug-in.

3.6.1 Library form

The AVR32765.zip contains the compiled library with the correct header file to integrate this one inside a custom application.

3.6.2 Source Code

The complete source code is available inside the zip package of the AVR32 Software Framework available to download here:

http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4192

3.6.3 AVR32 Studio Software Framework plug-in.

The DSPLib library can be easily incorporated inside a custom application through the AVR32 Studio Software Framework plug-in.

4 AVR32 – Fixed point DSP Library Modules

Here is a list of all modules:

- Basic
 - Filtering
 - Operators
 - Signal generation
 - Transforms
 - Vectors
 - Windowing
- Advanced
 - IMA/DVI ADPCM
 - Signal re-sampling



4.1 Filtering [Basic]

All the filtering functions currently supported by the DSP library.

- Finite Impulse Response Filter
- Infinite Impulse Response Filter
- Partial Infinite Impulse Response Filter
- Least Mean Square Filter
- Normalized Least Mean Square Filter
- Interpolation filter
- Lowpass FIR design

4.1.1 Standard Description

4.1.1.1 Finite Impulse Response Filter

This function computes a real FIR filter using the impulse response of the desire filter onto a fixed-length signal. It returns a signal of a length equals to (size - h_size + 1) elements.

Here is the formula of the FIR filter:

$$\text{vect1}(n) = \sum_{i=0}^N h(i) * \text{vect2}(N - i)$$

Note:

The impulse response of the filter has to be scaled to avoid overflowing values.

All the vectors have to be 32-bit aligned.

Relative functions:

- dsp16_filt_fir
- dsp32_filt_fir

4.1.1.2 Infinite Impulse Response Filter

This function computes a real IIR filter using the coefficients of the desire filter onto a fixed-length signal. It returns a signal of a length equals to size elements.

Here is the formula of the IIR filter:

$$\text{vect1}(n) = \sum_{i=0}^N \text{num}(i) * \text{vect2}(N - i) - \sum_{i=0}^M \text{den}(i) * \text{vect1}(M - i)$$

Note:

The data have to be scaled to avoid overflowing values.

All the vectors have to be 32-bit aligned.

Relative functions:

- dsp16_filt_iir
- dsp32_filt_iir

4.1.1.3 Partial Infinite Impulse Response Filter

This function computes a real IIR filter using the coefficients of the desire filter onto a fixed-length signal. It returns a signal of a length equals to (size - num_size + 1) elements.

Here is the formula of the IIR filter:

$$vect1(n) = \sum_{i=0}^N num(i) * vect2(N-i) - \sum_{i=0}^M den(i) * vect1(M-i)$$

Note:

The data have to be scaled to avoid overflowing values.
All the vectors have to be 32-bit aligned.

Relative functions:

- dsp16_filt_iirpart
- dsp32_filt_iirpart





4.1.1.4 Least Mean Square Filter

This function computes an adaptive LMS filter. It returns a (size)-length signal. Here is the formula of the LMS filter:

$$y(n) = (x^T w)(n)$$

$$e(n) = d(n) - y(n)$$

$$w(n+1) = w(n) + \mu e(n) x(n)$$

Note:

The data have to be scaled to avoid overflowing values.

All the vectors have to be 32-bit aligned.

You can change the value of the mu coefficient of the LMS filter by defining DSP_LMS_MU, at the compilation, with the desired value. This value defines mu as follow:

$$\mu = 2^{-(DSP_LMS_MU+1)}$$

Relative functions:

- dsp16_filt_lms
- dsp32_filt_lms

4.1.1.5 Normalized Least Mean Square Filter

This function computes an adaptive NLMS filter. It returns a (size)-length signal. Here is the formula of the NLMS filter:

$$y(n) = (x^* w)(n)$$

$$e(n) = d(n) - y(n)$$

$$w(n+1) = w(n) + \mu \frac{e^*(n) x(n)}{\sum_{i=0}^{N-1} x^2(n-i)}$$

Note:

The data have to be scaled to avoid overflowing values. All the vectors have to be 32-bit aligned. You can change the value of the mu coefficient of the NLMS filter by defining DSP_NLMS_MU, at the compilation, with the desired value. This value defines mu as follow:

$$\mu = 2^{-(DSP_NLMS_MU+1)}$$

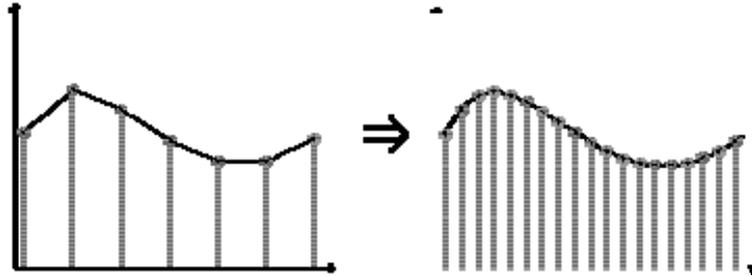
Relative functions:

- dsp16_filt_nlms
- dsp32_filt_nlms

4.1.1.6 Interpolation filter

This function performs an interpolation over the input signal. It returns a (`vect2_size * interpolation_ratio`)-length signal.

Here is the principle of the interpolation:



Note:

The data have to be scaled to avoid overflowing values.
All the vectors have to be 32-bit aligned.

Relative functions:

- `dsp16_filt_interpolation_coefsort`
- `dsp16_filt_interpolation`

4.1.1.7 Lowpass FIR design

This function uses a simple algorithm to calculate lowpass FIR filter's coefficients.

Note:

It does not take care of overflowing values.

Relative functions:

- `dsp16_filt_lpfirdesign`

4.1.2 C Description

4.1.2.1 Finite Impulse Response Filter

Description:	16-bit fixed point version of the FIR
Module:	FILTERING
Prototype:	<pre>void dsp16_filt_fir (dsp16_t *vect1, dsp16_t *vect2, int size, dsp16_t *h, dsp16_t h_size);</pre>
Arguments:	<p>vect1 : A pointer to a 16-bit fixed-point vector of (size - h_size + 1) elements corresponding to the output vector.</p> <p>vect2 : A pointer to a 16-bit fixed-point vector of size elements corresponding to the input vector.</p> <p>size: The length of the input vector (must be greater or equals to 4).</p> <p>h : A pointer to a 16-bit fixed-point vector of h_size elements corresponding to the vector containing the impulse response coefficients.</p> <p>h_size : The length of the impulse response of the filter (must be greater than 7).</p>
Return Value:	NONE
Notes:	<p>Due to its implementation, for the avr32-uc3 optimized version of the FIR, the output vector (vect1) have to have a length of 4*n elements to avoid overflows. You need the "Partial Convolution" module</p>



Description: 32-bit fixed point version of the FIR
Module: FILTERING
Prototype: <pre>void dsp32_filt_fir (dsp32_t *vect1, dsp32_t *vect2, int size, dsp32_t *h, dsp32_t h_size);</pre>
Arguments: <p>vect1 : A pointer to a 32-bit fixed-point vector of (size - h_size + 1) elements corresponding to the output vector.</p> <p>vect2 : A pointer to a 32-bit fixed-point vector of size elements corresponding to the input vector.</p> <p>size: The length of the input vector (must be greater or equals to 4).</p> <p>h : A pointer to a 32-bit fixed-point vector of h_size elements corresponding to the vector containing the impulse response coefficients.</p> <p>h_size : The length of the impulse response of the filter (must be greater than 7).</p>
Return Value: NONE
Remarks: Due to its implementation, for the avr32-uc3 optimized version of the FIR, the output vector (vect1) have to have a length of 4*n elements to avoid overflows. You need the "Partial Convolution" module

dsp32_filt_fir (See fir_example.c and fir_example.sce)

Example:

Using Scilab, compute of FIR low pass filter coefficients and filter signal:

```

fir_coef = [-0.01559148806
             -0.00894210585
             ...
             -0.00894210585
             -0.01559148806]
y =      [-0.18257484962
            -0.09404368788
            ...
            0.00000000000
            -0.03599992189]

```

C source code using the DSPLib with the pre-calculated coefficients:

```

A_ALIGNED dsp32_t fir_coef[FIR_COEF_SIZE] = {
    DSP32_Q(-0.01559148806),
    DSP32_Q(-0.00894210585),
    ...
    DSP32_Q(-0.00894210585),
    DSP32_Q(-0.01559148806)
}
dsp32_filt_fir(x, y, SIZE, fir_coef, FIR_COEF_SIZE);
y =      [- 0.1825
            - 0.0940
            ...
            0
            - 0.0359 ]

```



4.1.2.2 Infinite Impulse Response Filter

Description: 16-bit fixed point version of the IIR
Module: FILTERING
Prototype: <pre>void dsp16_filt_iir (dsp16_t *y, dsp16_t *x, int size, dsp16_t *num, int num_size, dsp16_t *den, int den_size, int num_prediv, int den_prediv);</pre>
Arguments: <p>y : A pointer to a 16-bit fixed-point vector of size elements corresponding to the output vector.</p> <p>x : A pointer to a 16-bit fixed-point vector of size elements corresponding to the input vector.</p> <p>size : The length of the input vector.</p> <p>num : A pointer to a 16-bit fixed-point vector of num_size elements corresponding to the vector containing the numerator's coefficients of the filter.</p> <p>num_size : The length of the numerator's coefficients of the filter (must be a multiple of 2).</p> <p>den : A pointer to a 16-bit fixed-point vector of den_size elements corresponding to the vector containing the denominator's coefficients of the filter.</p> <p>den_size : The length of the denominator's coefficients of the filter (must be a multiple of 2).</p> <p>num_prediv : The predivisor used to scale down the numerator's coefficients of the filter in order to avoid overflow values.</p> <p>den_prediv : The predivisor used to scale down the denominator's coefficients of the filter in order to avoid overflow values.</p>
Return Value: NONE
Remarks: Due to its implementation, for the avr32-uc3 optimized version of the IIR, the output vector (vect1) have to have a length of 6*n elements to avoid overflows.

dsp16_filt_iir (See iir_example.c and iir_example.sce)

Example:

Using Scilab, compute of butterworth coefficients and filter signal:

```

num = [0.6537018
        -3.2685088
        ...
        3.2685088
        -0.6537018]
den = [1
        -4.1534907
        ...
        2.498365
        -0.427326]
y = [ ... 0.0286757788249646 0.0456197560223300
       0.0505215086706680 0.0420876554669111 0.0226018579517016]

```

C source code using the DSPLib with the pre-calculated coefficients:

```

A_ALIGNED dsp16_t num[NUM_SIZE] = {
    DSP16_Q(0.6537018/ (1 << NUM_PREDIV)),
    DSP16_Q(-3.2685088/ (1 << NUM_PREDIV)),
    ...
}
A_ALIGNED dsp16_t den[DEN_SIZE] = {
    DSP16_Q(-4.1534907/ (1 << DEN_PREDIV)),
    DSP16_Q(6.9612765 / (1 << DEN_PREDIV)),
    ...
};
dsp16_filt_iir(&y[DEN_SIZE], &x[NUM_SIZE-1], SIZE, num, NUM_SIZE,
den, DEN_SIZE, NUM_PREDIV, DEN_PREDIV);

y = [... 0.00639704989 0.02903778954
        0.04445382457 0.04819074617
        0.03898097309 0.01909797642];

```



<p>Description: 32-bit fixed point version of the IIR</p>
<p>Module: FILTERING</p>
<p>Prototype:</p> <pre>void dsp32_filt_iir (dsp32_t *y, dsp32_t *x, int size, dsp32_t *num, int num_size, dsp32_t *den, int den_size, int num_prediv, int den_prediv);</pre>
<p>Arguments:</p> <p>y : A pointer to a 32-bit fixed-point vector of size elements corresponding to the output vector.</p> <p>x : A pointer to a 32-bit fixed-point vector of size elements corresponding to the input vector.</p> <p>size : The length of the input vector.</p> <p>num : A pointer to a 32-bit fixed-point vector of num_size elements corresponding to the vector containing the numerator's coefficients of the filter.</p> <p>num_size : The length of the numerator's coefficients of the filter (must be a multiple of 2).</p> <p>den : A pointer to a 32-bit fixed-point vector of den_size elements corresponding to the vector containing the denominator's coefficients of the filter.</p> <p>den_size : The length of the denominator's coefficients of the filter (must be a multiple of 2).</p> <p>num_prediv : The pre-divisor used to scale down the numerator's coefficients of the filter in order to avoid overflow values.</p> <p>den_prediv : The pre-divisor used to scale down the denominator's coefficients of the filter in order to avoid overflow values.</p>
<p>Return Value: NONE</p>
<p>Remarks:</p> <p>Due to its implementation, for the avr32-uc3 optimized version of the IIR, the output vector (vect1) have to have a length of 6*n elements to avoid overflows.</p>

4.1.2.3 Partial Infinite Impulse Response Filter

Description: 16-bit fixed point version of the IIR
Module: FILTERING
Prototype: <pre>void dsp16_filt_iir_part (dsp16_t *vect1, dsp16_t *vect2, int size, dsp16_t *num, int num_size, dsp16_t *den, int den_size, int num_prediv, int den_prediv);</pre>
Arguments: <p>vect1 : A pointer to a 16-bit fixed-point vector of (size - num_size + 1) elements corresponding to the output vector..</p> <p>vect2 A pointer to a 16-bit fixed-point vector of size elements corresponding to the input vector.</p> <p>size The length of the input vector.</p> <p>num A pointer to a 16-bit fixed-point vector of num_size elements corresponding to the vector containing the numerator's coefficients of the filter.</p> <p>num_size The length of the numerator's coefficients of the filter.</p> <p>den A pointer to a 16-bit fixed-point vector of den_size elements corresponding to the vector containing the denominator's coefficients of the filter.</p> <p>den_size The length of the denominator's coefficients of the filter.</p> <p>num_prediv The pre-divisor used to scale down the numerator's coefficients of the filter in order to avoid overflow values.</p> <p>den_prediv The pre-divisor used to scale down the denominator's coefficients of the filter in order to avoid overflow values.</p>
Return Value: NONE
Remarks: Due to its implementation, for the avr32-uc3 optimized version of the IIR, the output vector (vect1) have to have a length of 6*n elements to avoid overflows.





Description: 32-bit fixed point version of the IIR
Module: FILTERING
Prototype: <pre>void dsp32_filt_iir_part (dsp32_t *vect1, dsp32_t *vect2, int size, dsp32_t *num, int num_size, dsp32_t *den, int den_size, int num_prediv, int den_prediv);</pre>
Arguments: <p>vect1 : A pointer to a 32-bit fixed-point vector of (size - num_size + 1) elements corresponding to the output vector..</p> <p>vect2 A pointer to a 32-bit fixed-point vector of size elements corresponding to the input vector.</p> <p>size The length of the input vector.</p> <p>num A pointer to a 32-bit fixed-point vector of num_size elements corresponding to the vector containing the numerator's coefficients of the filter.</p> <p>num_size The length of the numerator's coefficients of the filter.</p> <p>den A pointer to a 32-bit fixed-point vector of den_size elements corresponding to the vector containing the denominator's coefficients of the filter.</p> <p>den_size The length of the denominator's coefficients of the filter.</p> <p>num_prediv The pre-divisor used to scale down the numerator's coefficients of the filter in order to avoid overflow values.</p> <p>den_prediv The pre-divisor used to scale down the denominator's coefficients of the filter in order to avoid overflow values.</p>
Return Value: NONE
Remarks: Due to its implementation, for the avr32-uc3 optimized version of the IIR, the output vector (vect1) have to have a length of 6*n elements to avoid overflows.

dsp32_filt_iir_part (See iirpart_example.c and iirpart_example.sce files)

Example:

Using Scilab, compute of butterworth coefficients and filter signal:

```

num = [0.0000274
        0.0004939
        ...
        0.0004939
        0.0000274 ]
den = [0.00000000000000001427
        2.4446755
        ...
        -0.00000000000000000000
        0.000000001506]
y = [ ...
        -0.00998044237
        -0.00929886725
        -0.00770704662]

```

C source code using the DSPLib with the pre-calculated coefficients:

```

A_ALIGNED dsp32_t num[NUM_SIZE] = {
    DSP16_Q(0.0000274/ (1 << PREDIV)),
    DSP16_Q(0.0004939/ (1 << PREDIV)),
    ...
}
A_ALIGNED dsp32_t den[DEN_SIZE] = {
};
dsp32_filt_iir_part(&y[DEN_SIZE], &x[NUM_SIZE-1], SIZE, num,
NUM_SIZE, den, DEN_SIZE, NUM_PREDIV, DEN_PREDIV);

y = [...
    -0.00998368071,
    -0.00929394507,
    -0.00770432784];

```





4.1.2.4 Least Mean Square Filter

Description: 16-bit fixed point version of the LMS filter
Module: FILTERING
Prototype: <pre>void dsp16_filt_lms (dsp16_t *x, dsp16_t *w, int size, dsp16_t new_x, dsp16_t d, dsp16_t *y, dsp16_t *e);</pre>
Arguments: <p>x A pointer to a 16-bit fixed-point vector of (size) elements that acts as a circular vector, filled with the input samples. The elements have to be initialized to zero and then you just need to insert this vector each time you call this functions without filling any of its values.</p> <p>w A pointer to a 16-bit fixed-point vector of size elements corresponding to the coefficients of the filter. Just initialize The elements to zero and after several iterations, this vector will be filled with the actual coefficients of the filter.</p> <p>size The length of the circular vector (x) and of the coefficient's vector (w). It must be a multiple of 4.</p> <p>new_x A 16-bit fixed-point value which contains a new input sample signal.</p> <p>d A 16-bit fixed-point value which contains the current sample of the reference's signal.</p> <p>y A pointer to a 16-bit fixed-point value corresponding to the current sample of the output signal.</p> <p>e A pointer to a 16-bit fixed-point value corresponding to the current sample of the output error signal.</p>
Return Value: NONE
Remarks: NONE

Description:	32-bit fixed point version of the LMS filter
Module:	FILTERING
Prototype:	<pre> void dsp32_filt_lms (dsp32_t *x, dsp32_t *w, int size, dsp32_t new_x, dsp32_t d, dsp32_t *y, dsp32_t *e); </pre>
Arguments:	<p>x A pointer to a 32-bit fixed-point vector of (size) elements that acts as a circular vector, filled with the input samples. The elements have to be initialized to zero and then you just need to insert this vector each time you call this functions without filling any of its values.</p> <p>w A pointer to a 16-bit fixed-point vector of size elements corresponding to the coefficients of the filter. Just initialize The elements to zero and after several iterations, this vector will be filled with the actual coefficients of the filter.</p> <p>size The length of the circular vector (x) and of the coefficient's vector (w). It must be a multiple of 4.</p> <p>new_x A 32-bit fixed-point value which contains a new input sample signal.</p> <p>d A 16-bit fixed-point value which contains the current sample of the reference's signal.</p> <p>y A pointer to a 16-bit fixed-point value corresponding to the current sample of the output signal.</p> <p>e A pointer to a 16-bit fixed-point value corresponding to the current sample of the output error signal.</p>
Return Value:	NONE
Remarks:	



dsp16_filt_lms (See lms_example.c and lms_example.m files)

Example:

Using Matlab,

Generate a **x** input signal made of a combination of 2 sinus.

Filter this signal with an unknown process. The resulting signal is called **d**.

Compute a LMS filter and check the output coefficients converge with the unknown process.

The unknown process model is

```
fir_coef = [ 0.0132,  
            0.0254),  
            0.0579),  
            0.1006),  
            0.1398),  
            0.1632),  
            0.1632),  
            0.1398),  
            0.1006),  
            0.0579),  
            0.0254),  
            0.0132) ]
```

C source code using the DSPLib with the pre-calculated **x** and **d** signals.

```
w      = [ 0.024536,  
          0.050109,  
          0.076080,  
          0.099517,  
          0.117767,  
          0.128936,  
          0.131835,  
          0.126739,  
          0.114013,  
          0.095642,  
          0.073516,  
          0.049316];
```

4.1.2.5 Normalized Least Mean Square Filter

Description: 16-bit fixed point version of the Normalized LMS filter
Module: FILTERING
Prototype: <pre>void dsp16_filt_nlms (dsp16_t *x, dsp16_t *w, int size, dsp16_t new_x, dsp16_t d, dsp16_t *y, dsp16_t *e);</pre>
Arguments: <p>x A pointer to a 16-bit fixed-point vector of (size) elements that acts as a circular vector, filled with the input samples. The elements have to be initialized to zero and then you just need to insert this vector each time you call this functions without filling any of its values.</p> <p>w A pointer to a 16-bit fixed-point vector of size elements corresponding to the coefficients of the filter. Just initialize The elements to zero and after several iterations, this vector will be filled with the actual coefficients of the filter.</p> <p>size The length of the circular vector (x) and of the coefficient's vector (w). It must be a multiple of 4.</p> <p>new_x A 16-bit fixed-point value which contains a new input sample signal.</p> <p>d A 16-bit fixed-point value which contains the current sample of the reference's signal.</p> <p>y A pointer to a 16-bit fixed-point value corresponding to the current sample of the output signal.</p> <p>e A pointer to a 16-bit fixed-point value corresponding to the current sample of the output error signal.</p>
Return Value: NONE
Remarks:

Description: 32-bit fixed point version of the Normalized LMS filter
Module: FILTERING
Prototype: <pre> void dsp32_filt_nlms (dsp32_t *x, dsp32_t *w, int size, dsp32_t new_x, dsp32_t d, dsp32_t *y, dsp32_t *e); </pre>
Arguments: <p>x A pointer to a 32-bit fixed-point vector of (size) elements that acts as a circular vector, filled with the input samples. The elements have to be initialized to zero and then you just need to insert this vector each time you call this functions without filling any of its values.</p> <p>w A pointer to a 16-bit fixed-point vector of size elements corresponding to the coefficients of the filter. Just initialize The elements to zero and after several iterations, this vector will be filled with the actual coefficients of the filter.</p> <p>size The length of the circular vector (x) and of the coefficient's vector (w). It must be a multiple of 4.</p> <p>new_x A 32-bit fixed-point value which contains a new input sample signal.</p> <p>d A 16-bit fixed-point value which contains the current sample of the reference's signal.</p> <p>y A pointer to a 16-bit fixed-point value corresponding to the current sample of the output signal.</p> <p>e A pointer to a 16-bit fixed-point value corresponding to the current sample of the output error signal.</p>
Return Value: NONE
Remarks:

dsp32_filt_nlms (See nlms_example.c and nlms_example.m files)

Example:

Using Matlab,

Generate a **x** input signal made of a combination of 2 sinus.

Filter this signal with an unknown process. The resulting signal is called **d**.

Compute a LMS filter and check the output coefficients converge with the unknown process.

The unknown process model is

```
fir_coef = [ 0.0132,
             0.0254),
            0.0579),
            0.1006),
            0.1398),
            0.1632),
            0.1632),
            0.1398),
            0.1006),
            0.0579),
            0.0254),
            0.0132) ]
```

C source code using the DSPLib with the pre-calculated **x** and **d** signals.

```
w      = [0.08656555586,
          0.11905200410,
          0.17031530270,
          0.20344278152,
          0.22639864860,
          0.22585131265,
          0.20688786015,
          0.16399965505,
          0.10478496320,
          0.02905859762,
          -0.05285281242,
          -0.13855776903]
```





4.1.2.6 Interpolation filter

Description: This function resort the coefficients of a FIR filter to be used with the function <code>dsp16_filt_interpolation</code>
Module: FILTERING
Prototype: <pre>void dsp16_filt_interpolation (dsp16_t *vect1, dsp16_t *vect2, int vect2_size, dsp16_t *h, int h_size, int interpolation_ratio);</pre>
Arguments: <p>vect1 A pointer to a 16-bit fixed-point vector where to store the result. It must be of a size (in sample) equals to the size of the input vector multiplied by the interpolation factor.</p> <p>vect2 A pointer to a 16-bit fixed-point vector containing the input samples.</p> <p>vect2_size The size of the input vector.</p> <p>h A pointer to a 16-bit fixed-point vector which contains the coefficients of the filter. These coefficients must be reorder with the function <code>dsp16_filt_interpolation_coefsor</code> before being used.</p> <p>h_size The size of this vector.</p> <p>interpolation_ratio The interpolation factor desired for this interpolation.</p>
Return Value: NONE
Remarks: NONE

Description:	This function interpolates a vector.
Module:	FILTERING
Prototype:	<pre> void dsp16_filt_interpolation (dsp16_t *vect1, dsp16_t *vect2, int vect2_size, dsp16_t *h, int h_size, int interpolation_ratio); </pre>
Arguments:	<p>vect1 A pointer to a 16-bit fixed-point vector where to store the result. It must be of a size (in sample) equals to the size of the input vector multiplied by the interpolation factor.</p> <p>vect2 A pointer to a 16-bit fixed-point vector containing the input samples.</p> <p>vect2_size The size of the input vector.</p> <p>h A pointer to a 16-bit fixed-point vector which contains the coefficients of the filter. These coefficients must be reorder with the function <code>dsp16_filt_interpolation_coefsrt</code> before being used.</p> <p>h_size The size of this vector.</p> <p>interpolation_ratio The interpolation factor desired for this interpolation.</p>
Return Value:	NONE
Remarks:	NONE



4.1.2.7 Lowpass FIR design

Description: 16-bit fixed point version of the low-pass FIR filter design.
Module: FILTERING
Prototype: <pre>void dsp16_filt_lpfirdesign (dsp16_t *c, int fc, int fs, int order, dsp16_win_fct_t dsp16_win_fct, dsp_filt_design_options_t options);</pre>
Arguments: c A pointer to a 16-bit fixed-point vector of "order" size, used to store the coefficients of the filter designed. fc Cutoff frequency of the low-pass filter. fs Sample rate of the signal to filter. order Order of the filter to design. dsp16_win_fct A window to apply to the coefficients. If this parameter is NULL, then no window is applied. options Specific options for the design.
Return Value: NONE
Remarks: NONE

4.2 Operators [Basic]

All the fixed point operators functions currently implemented in the DSP library.

- Multiplication
- Division
- Sine
- Cosine
- Arc sine
- Arc cosine
- Absolute
- Square root
- Natural logarithm
- Binary logarithm or base 2 logarithm
- Common logarithm or base 10 logarithm
- Exponential
- Power
- Random

4.2.1 Standard description

4.2.1.1 Multiplication

These functions multiply two fixed-point numbers.

Relative functions:

- dsp16_op_mul
- dsp32_op_mul

4.2.1.2 Division

These functions divide two fixed-point numbers. Relative functions:

- dsp16_op_div
- dsp32_op_div

4.2.1.3 Sine

These functions compute the sine of the argument 'angle'.

Note:

Angle is scaled to fit in the range [-1; 1], where -1 corresponds to -pi and 1 to pi.

Relative functions:

- dsp16_op_sin
- dsp32_op_sin



4.2.1.4 Cosine

These functions compute the cosine of the argument 'angle'. Angles are scale to fit in the range [-1; 1], where -1 corresponds to -pi and 1 to pi. Relative functions:

- dsp16_op_cos
- dsp32_op_cos

4.2.1.5 Arc sine

These functions compute the arc sine of the argument.

Note:

The output is scaled to fit in the range [-0.5; 0.5], where -0.5 corresponds to -pi/2 and 0.5 to pi/2.

Relative functions:

- dsp16_op_asin
- dsp32_op_asin

4.2.1.6 Arc cosine

These functions compute the arc cosine of the argument.

Note:

The output is scaled to fit in the range [0.; 1.], where 0. corresponds to 0. and 1. to pi.

Relative functions:

- dsp16_op_acos
- dsp32_op_acos

4.2.1.7 Absolute

These functions compute the absolute value of the argument. Relative functions:

- dsp16_op_abs
- dsp32_op_abs

4.2.1.8 Square root

These functions compute the square root of the argument. Relative functions:

- dsp16_op_sqrt
- dsp32_op_sqrt

4.2.1.9 Natural logarithm

These functions compute the natural logarithm of the argument. Relative functions:

- dsp16_op_ln
- dsp32_op_ln

Note:

The output will be limit in the range of the fixed point format used.

4.2.1.10 Binary logarithm or base 2 logarithm

These functions compute the natural logarithm of the argument. Relative functions:

- dsp16_op_log2
- dsp32_op_log2

Note:

The output will be limit in the range of the fixed point format used.

4.2.1.11 Common logarithm or base 10 logarithm

These functions compute the common logarithm of the argument.

Note:

The output will be limit in the range of the fixed point format used.

Relative functions:

- dsp16_op_log10
- dsp32_op_log10

4.2.1.12 Exponential

These functions compute the exponential of the argument.

Note:

The output will be limit in the range of the fixed point format used.

Relative functions:

- dsp16_op_exp
- dsp32_op_exp

4.2.1.13 Power

These functions compute x^y .

Note:

The output will be limit in the range of the fixed point format used.

Relative functions:

- dsp16_op_pow
- dsp32_op_pow





4.2.1.14 *Random*

These functions generate a pseudo-randomized number Relative functions:

- `dsp_op_srand`
- `dsp16_op_rand`
- `dsp32_op_rand`

4.2.2 C description

4.2.2.1 Multiplication

Description:	16-bit fixed point version of the multiplication function.
Module:	OPERATORS
Prototype:	<pre>inline static dsp16_t dsp16_op_mul (dsp16_t num1, dsp16_t num2)</pre>
Arguments:	<p>num1 The first argument of the multiplication.</p> <p>num2 The second argument of the multiplication.</p>
Return Value:	The result of the multiplication.
Remarks:	NONE

Description:	32-bit fixed point version of the multiplication function.
Module:	OPERATORS
Prototype:	<pre>inline static dsp32_t dsp32_op_mul (dsp32_t num1, dsp32_t num2)</pre>
Arguments:	<p>num1 The first argument of the multiplication.</p> <p>num2 The second argument of the multiplication.</p>
Return Value:	The result of the multiplication.
Remarks:	NONE



4.2.2.2 Division

Description: 16-bit fixed point version of the division function.
Module: OPERATORS
Prototype: <pre>inline static dsp16_t dsp16_op_div (dsp16_t num1, dsp16_t num2)</pre>
Arguments: num1 The first argument of the division. num2 The second argument of the division.
Return Value: The result of the division.
Remarks: NONE

Description: 32-bit fixed point version of the division function.
Module: OPERATORS
Prototype: <pre>inline static dsp32_t dsp32_op_div (dsp32_t num1, dsp32_t num2)</pre>
Arguments: num1 The first argument of the division. num2 The second argument of the division.
Return Value: The result of the division.
Remarks: NONE

4.2.2.3 Sine

Description: 16-bit fixed point version of the sine function.
Module: OPERATORS
Prototype: <pre>inline static dsp16_t dsp16_op_sin (dsp16_t angle)</pre>
Arguments: angle The angle to compute
Return Value: The sine of 'angle' is returned.
Remarks: If you are using a number greater than 1 (or lower than -1) for the angle, the function will take the modulo 1 of this angle.

Description: 32-bit fixed point version of the sine function.
Module: OPERATORS
Prototype: <pre>inline static dsp32_t dsp32_op_sin (dsp32_t angle)</pre>
Arguments: angle The angle to compute
Return Value: The sine of 'angle' is returned.
Remarks: If you are using a number greater than 1 (or lower than -1) for the angle, the function will take the modulo 1 of this angle.



4.2.2.4 Cosine

Description: 16-bit fixed point version of the cosine function.
Module OPERATORS
Prototype: <pre>inline static dsp16_t dsp16_op_cos (dsp16_t angle)</pre>
Arguments: angle The angle to compute
Return Value: The cosine of 'angle' is returned.
Remarks: If you are using a number greater than 1 (or lower than -1) for the angle, the function will take the modulo 1 of this angle.

Description: 32-bit fixed point version of the cosine function.
Module OPERATORS
Prototype: <pre>inline static dsp32_t dsp32_op_sin (dsp32_t angle)</pre>
Arguments: angle The angle to compute
Return Value: The cosine of 'angle' is returned.
Remarks: If you are using a number greater than 1 (or lower than -1) for the angle, the function will take the modulo 1 of this angle.

4.2.2.5 Arc sine

Description:	16-bit fixed point version of the arc sine function.
Module:	OPERATORS
Prototype:	<pre>inline static dsp16_t dsp16_op_asin (dsp16_t number)</pre>
Arguments:	number The input argument
Return Value:	The arc sine of 'number' is returned.
Remarks:	If you are using a number greater than 1 (or lower than -1) for the 'number', the function will limit the output to the range [-0.5; 0.5].

Description:	32-bit fixed point version of the arc sine function.
Module:	OPERATORS
Prototype:	<pre>inline static dsp32_t dsp32_op_asin (dsp32_t number)</pre>
Arguments:	number The input argument
Return Value:	The arc sine of 'number' is returned.
Remarks:	If you are using a number greater than 1 (or lower than -1) for the 'number', the function will limit the output to the range [-0.5; 0.5].



4.2.2.6 Arc cosine

Description: 16-bit fixed point version of the arc cosine function.
Module: OPERATORS
Prototype: <pre>inline static dsp16_t dsp16_op_acos (dsp16_t number);</pre>
Arguments: number The input argument
Return Value: The arc cosine of 'number' is returned.
Remarks: If you are using a number greater than 1 (or lower than -1) for the 'number', the function will limit the output to the range [0.; 1.].

Description: 32-bit fixed point version of the arc cosine function.
Module: OPERATORS
Prototype: <pre>inline static dsp32_t dsp32_op_acos (dsp16_t number);</pre>
Arguments: number The input argument
Return Value: The arc cosine of 'number' is returned.
Remarks: If you are using a number greater than 1 (or lower than -1) for the 'number', the function will limit the output to the range [0.; 1.].

4.2.2.7 Absolute

Description: 16-bit fixed point version of the absolute function.
Module: OPERATORS
Prototype: <pre>inline static dsp16_t dsp16_op_abs (dsp16_t number);</pre>
Arguments: number The input argument
Return Value: The absolute value of the number.
Remarks: NONE

Description: 32-bit fixed point version of the absolute function.
Module: OPERATORS
Prototype: <pre>inline static dsp32_t dsp32_op_abs (dsp32_t number);</pre>
Arguments: number The input argument
Return Value: The absolute value of the number.
Remarks: NONE





4.2.2.8 Square root

Description: 16-bit fixed point version of the square root function.
Module: OPERATORS
Prototype: <pre>dsp16_t dsp16_op_sqrt (dsp16_t number);</pre>
Arguments: number The input argument
Return Value: The square root of 'number' is returned.
Remarks: If you are using a number lower than 0 for the number, the function will return 0.

Description: 32-bit fixed point version of the square root function.
Module: OPERATORS
Prototype: <pre>dsp32_t dsp32_op_sqrt (dsp32_t number);</pre>
Arguments: number The input argument
Return Value: The square root of 'number' is returned.
Remarks: If you are using a number lower than 0 for the number, the function will return 0.

4.2.2.9 Natural logarithm

Description: 16-bit fixed point version of the natural logarithm function.
Module: OPERATORS
Prototype: <pre>dsp16_t dsp16_op_ln (dsp16_t number);</pre>
Arguments: number The input argument
Return Value: The natural logarithm of 'number' is returned.
Remarks: If you are using a number lower than 0 for the number, the function will return DSP_Q_MIN(DSP16_QA, DSP16_QB).

Description: 32-bit fixed point version of the natural logarithm function.
Module: OPERATORS
Prototype: <pre>dsp32_t dsp32_op_ln (Dsp32_t number);</pre>
Arguments: number The input argument
Return Value: The natural logarithm of 'number' is returned.
Remarks: If you are using a number lower than 0 for the number, the function will return DSP_Q_MIN(DSP32_QA, DSP32_QB).



4.2.2.10 Binary logarithm or base 2 logarithm

Description: 16-bit fixed point version of the binary logarithm function.
Module: OPERATORS
Prototype: <pre>dsp16_t dsp16_op_log2 (dsp16_t number);</pre>
Arguments: number The input argument
Return Value: The binary logarithm of 'number' is returned.
Remarks: If you are using a number lower than 0 for the number, the function will return DSP_Q_MIN(DSP16_QA, DSP16_QB).

Description: 32-bit fixed point version of the binary logarithm function.
Module: OPERATORS
Prototype: <pre>dsp32_t dsp32_op_log2 (dsp32_t number);</pre>
Arguments: number The input argument
Return Value: The binary logarithm of 'number' is returned.
Remarks: If you are using a number lower than 0 for the number, the function will return DSP_Q_MIN(DSP32_QA, DSP32_QB).

4.2.2.11 Common logarithm or base 10 logarithm

Description: 16-bit fixed point version of the common logarithm function.
Module: OPERATORS
Prototype: <pre>dsp16_t dsp16_op_log10 (dsp16_t number);</pre>
Arguments: number The input argument
Return Value: The common logarithm of 'number' is returned.
Remarks: If you are using a number lower than 0 for the number, the function will return DSP_Q_MIN(DSP16_QA, DSP16_QB).

Description: 32-bit fixed point version of the common logarithm function.
Module: OPERATORS
Prototype: <pre>dsp32_t dsp32_op_log10 (dsp32_t number);</pre>
Arguments: number The input argument
Return Value: The common logarithm of 'number' is returned.
Remarks: If you are using a number lower than 0 for the number, the function will return DSP_Q_MIN(DSP32_QA, DSP32_QB).



4.2.2.12 Exponential

Description: 16-bit fixed point version of the exponential function.
Module: OPERATORS
Prototype: <pre>dsp16_t dsp16_op_exp (dsp16_t number);</pre>
Arguments: number The input argument
Return Value: The exponential of 'number' is returned.
Remarks: NONE

Description: 32-bit fixed point version of the exponential function.
Module: OPERATORS
Prototype: <pre>dsp32_t dsp32_op_exp (dsp32_t number);</pre>
Arguments: number The input argument
Return Value: The exponential of 'number' is returned.
Remarks: NONE

4.2.2.13 Power

Description: 16-bit fixed point version of the power function.
Module: PROTOTYPE
Prototype: <pre>dsp16_t dsp16_op_pow (dsp16_t x, dsp16_t y);</pre>
Arguments: x The number from which the power has to be applied. y The power.
Return Value: The x power y is returned.
Remarks: if the argument x is negative: this would result in a complex number.

Description: 32-bit fixed point version of the power function.
Module: PROTOTYPE
Prototype: <pre>dsp32_t dsp32_op_pow (dsp32_t x, dsp32_t y);</pre>
Arguments: x The number from which the power has to be applied. y The power.
Return Value: The x power y is returned.
Remarks: if the argument x is negative: this would result in a complex number.



4.2.2.14 Random

Description: 16-bit fixed point version of the random operator.
Module: OPERATORS
Prototype: <code>dsp16_t dsp16_op_rand();</code>
Arguments: NONE
Return Value: The random value
Remarks: The number generated can be any 16-bit value in the range [-1; 1[.

Description: 32-bit fixed point version of the random operator.
Module: OPERATORS
Prototype: <code>dsp32_t dsp32_op_rand();</code>
Arguments: NONE
Return Value: The random value
Remarks: The number generated can be any 32-bit value in the range [-1; 1[.

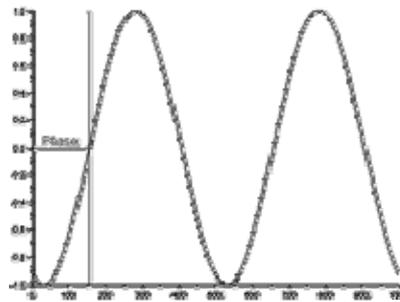
4.3 Signal generation [Basic]

All the signal generation functions currently supported by the DSP library.

- Periodic
 - Cosinusoidal
 - Sinusoidal
 - Square
 - Rectangular
 - Saw tooth
 - Dirac comb
- Non-periodic
 - Noise
 - Ramp
 - Step
 - Dirac

4.3.1 Standard description

4.3.1.1 Sinusoidal



This function generates a sinusoidal signal with a specified frequency, sampling frequency and phase.

Note:

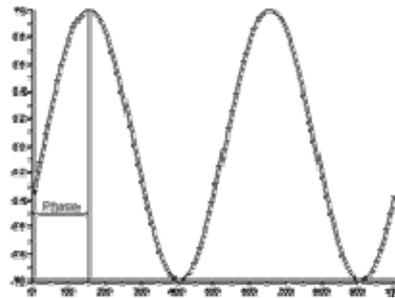
The amplitude of the signal fits in the range $[-1; 1]$.

The phase is scaled to fit in the range $[-1; 1]$, where -1 corresponds to $-\pi$ and 1 to π .

Relative functions:

- `dsp16_gen_sin`
- `dsp32_gen_sin`

4.3.1.2 Cosinusoidal



This function generates a cosinusoidal signal with a specified frequency, sampling frequency and phase.

Note:

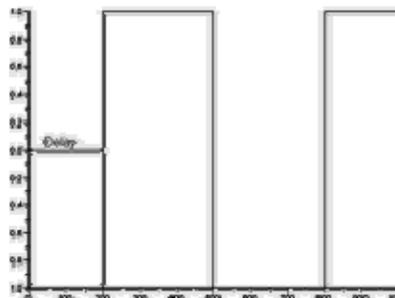
The amplitude of the signal fits in the range [-1; 1].

The phase is scaled to fit in the range [-1; 1], where -1 corresponds to $-\pi$ and 1 to π .

Relative functions:

- `dsp16_gen_cos`
- `dsp32_gen_cos`

4.3.1.3 Square



This function generates a square signal.

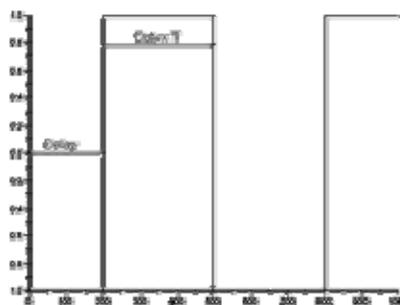
Note:

The amplitude of the signal fits in the range [-1; 1].

Relative functions:

- `dsp16_gen_sqr`
- `dsp32_gen_sqr`

4.3.1.4 Rectangular



This function generates a rectangular signal.

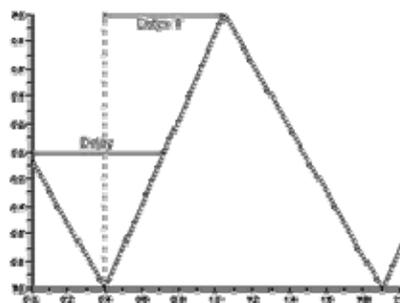
Note:

The amplitude of the signal fits in the range [-1; 1].

Relative functions:

- `dsp16_gen_rect`
- `dsp32_gen_rect`

4.3.1.5 Saw Tooth



This function generates a saw tooth signal.

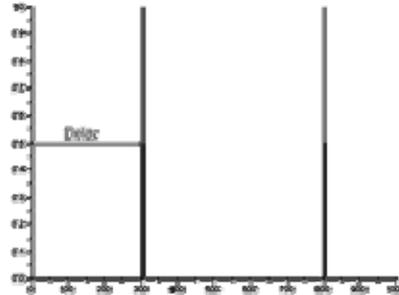
Note:

The amplitude of the signal fits in the range [-1; 1].

Relative functions:

- `dsp16_gen_saw`
- `dsp32_gen_saw`

4.3.1.6 Dirac Comb



This function generates a dirac comb signal.

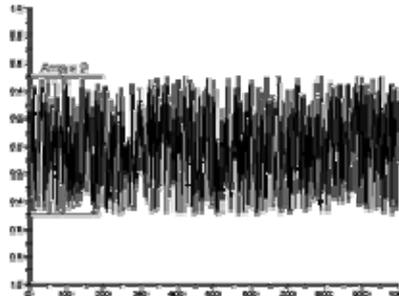
Note:

The amplitude of the signal fits in the range [0; 1].

Relative functions:

- `dsp16_gen_dcomb`
- `dsp32_gen_dcomb`

4.3.1.7 Noise



This function generates a noise with a specified amplitude.

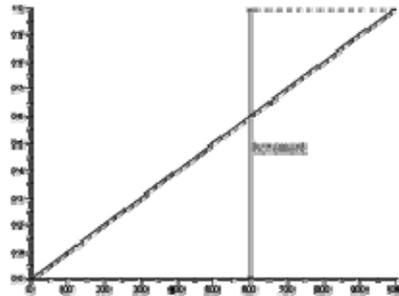
Note:

The amplitude of the signal fits in the range [-1; 1].

Relative functions:

- `dsp16_gen_noise`
- `dsp32_gen_noise`

4.3.1.8 Ramp



This function generates a ramp.

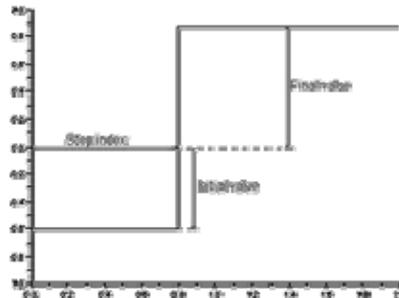
Note:

The amplitude of the signal fits in the range [0; 1].

Relative functions:

- dsp16_gen_ramp
- dsp32_gen_ramp

4.3.1.9 Step

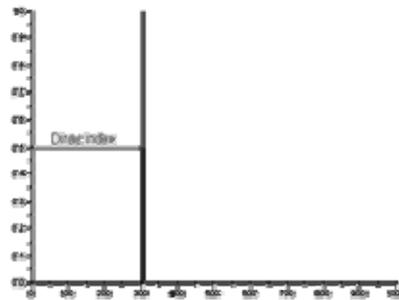


This function generates a step.

Relative functions:

- dsp16_gen_step
- dsp32_gen_step

4.3.1.10 Dirac



This function generates a dirac.

Note:

The amplitude of the signal fits in the range [0; 1].

Relative functions:

- `dsp16_gen_dirac`
- `dsp32_gen_dirac`

4.3.2 C description

4.3.2.1 Sinusoidal

Description: 16-bit fixed point version of the sinusoidal signal generation.
Module: SIGNAL_GENERATION
Prototype: <pre>void dsp16_gen_sin (dsp16_t *vect1, int size, int f, int fs, dsp16_t phase);</pre>
Arguments: vect1 A pointer to a 16-bit vector which is the output vector of this function. size The number of elements in the output vector. f The frequency of the output signal. fs The sampling frequency of the output signal. phase The phase of the output signal.
Return Value: NONE
Remarks: NONE



Description: 32-bit fixed point version of the sinusoidal signal generation.
Module: SIGNAL_GENERATION
Prototype: <pre>void dsp32_gen_sin (dsp32_t *vect1, int size, int f, int fs, dsp32_t phase);</pre>
Arguments: vect1 A pointer to a 32-bit vector which is the output vector of this function. size The number of elements in the output vector. f The frequency of the output signal. fs The sampling frequency of the output signal. phase The phase of the output signal.
Return Value: NONE
Remarks: NONE

4.3.2.2 Cosinusoidal

Description: 16-bit fixed point version of the cosine sinusoidal signal generation.
Module: SIGNAL_GENERATION
Prototype: <pre>void dsp16_gen_cos (dsp16_t *vect1, int size, int f, int fs, dsp16_t phase);</pre>
Arguments: vect1 A pointer to a 16-bit vector which is the output vector of this function. size The number of elements in the output vector. f The frequency of the output signal. fs The sampling frequency of the output signal. phase The phase of the output signal.
Return Value: NONE.
Remarks: NONE.



Description: 32-bit fixed point version of the co sinusoidal signal generation.
Module: SIGNAL_GENERATION
Prototype: <pre>void dsp32_gen_cos (dsp32_t *vect1, int size, int f, int fs, dsp16_t phase);</pre>
Arguments: vect1 A pointer to a 32-bit vector which is the output vector of this function. size The number of elements in the output vector. f The frequency of the output signal. fs The sampling frequency of the output signal. phase The phase of the output signal.
Return Value: NONE.
Remarks: NONE.

4.3.2.3 Square

Description:	16-bit fixed point version of the square signal generation.
Module:	SIGNAL_GENERATION
Prototype:	<pre> inline static void dsp16_gen_sqr (dsp16_t *vect1, int size, int f, int fs, dsp16_t delay) </pre>
Arguments:	<p>vect1 A pointer to a 16-bit vector which is the output vector of this function.</p> <p>size The number of elements in the output vector.</p> <p>f The frequency of the output signal.</p> <p>fs The sampling frequency of the output signal.</p> <p>delay The delay of the periodic waveform. The delay must fit in the range [0; 1] where 1 to T.</p>
Return Value:	NONE
Remarks:	NONE



Description: 32-bit fixed point version of the square signal generation.
Module: SIGNAL_GENERATION
Prototype: <pre>inline static void dsp32_gen_sqr (dsp32_t *vect1, int size, int f, int fs, dsp32_t delay)</pre>
Arguments: vect1 A pointer to a 32-bit vector which is the output vector of this function. size The number of elements in the output vector. f The frequency of the output signal. fs The sampling frequency of the output signal. delay The delay of the periodic waveform. The delay must fit in the range [0; 1] where 1 to T.
Return Value: NONE
Remarks: NONE

4.3.2.4 Rectangular

Description: 16-bit fixed point version of the rectangular signal generation.
Module: SIGNAL_GENERATION
Prototype: <pre>void dsp16_gen_rect (dsp16_t *vect1, int size, int f, int fs, dsp16_t duty, dsp16_t delay);</pre>
Arguments: <p>vect1 A pointer to a 16-bit vector which is the output vector of this function.</p> <p>size The number of elements in the output vector.</p> <p>f The frequency of the output signal.</p> <p>fs The sampling frequency of the output signal.</p> <p>duty The duty cycle of the output signal. The duty cycle is a number in the range]0; 1] which is the ratio between the pulse duration and the period of the waveform.</p> <p>delay The delay of the periodic waveform. The delay must fit in the range [0; 1] where 1 to T.</p>
Return Value: NONE
Remarks: NONE



<p>Description: 32-bit fixed point version of the rectangular signal generation.</p>
<p>Module: SIGNAL_GENERATION</p>
<p>Prototype:</p> <pre>void dsp32_gen_rect (dsp32_t *vect1, int size, int f, int fs, dsp32_t duty, dsp32_t delay);</pre>
<p>Arguments:</p> <p>vect1 A pointer of a 32-bit vector which is the output vector of this function.</p> <p>size The number of elements in the output vector.</p> <p>f The frequency of the output signal.</p> <p>fs The sampling frequency of the output signal.</p> <p>duty The duty cycle of the output signal. The duty cycle is a number in the range]0; 1] which is the ratio between the pulse duration and the period of the waveform.</p> <p>delay The delay of the periodic waveform. The delay must fit in the range [0; 1] where 1 to T.</p>
<p>Return Value: NONE</p>
<p>Remarks: NONE</p>

4.3.2.5 Saw Tooth

Description: 16-bit fixed point version of the saw tooth signal generation.
Module: SIGNAL_GENERATION
Prototype: <pre>void dsp16_gen_saw (dsp16_t *vect1, int size, int f, int fs, dsp16_t duty, dsp16_t delay);</pre>
Arguments: <p>vect1 A pointer to a 16-bit vector which is the output vector of this function.</p> <p>size The number of elements in the output vector.</p> <p>f The frequency of the output signal.</p> <p>fs The sampling frequency of the output signal.</p> <p>duty The duty cycle of the output signal. The duty cycle is a number in the range]0; 1] which is the ratio between the pulse duration and the period of the waveform.</p> <p>delay The delay of the periodic waveform. The delay must fit in the range [0; 1] where 1 to T.</p>
Return Value: NONE
Remarks: NONE





Description: 32-bit fixed point version of the saw tooth signal generation.
Module: SIGNAL_GENERATION
Prototype: <pre>void dsp32_gen_saw (dsp32_t *vect1, int size, int f, int fs, dsp32_t duty, dsp32_t delay);</pre>
Arguments: vect1 A pointer to a 32-bit vector which is the output vector of this function. size The number of elements in the output vector. f The frequency of the output signal. fs The sampling frequency of the output signal. duty The duty cycle of the output signal. The duty cycle is a number in the range]0; 1] which is the ratio between the pulse duration and the period of the waveform. delay The delay of the periodic waveform. The delay must fit in the range [0; 1] where 1 to T.
Return Value: NONE
Remarks: NONE

4.3.2.6 Dirac Comb

Description: 16-bit fixed point version of the dirac comb signal generation.
Module: SIGNAL_GENERATION
Prototype: <pre>void dsp16_gen_dcomb (dsp16_t *vect1, int size, int f, int fs, dsp16_t delay);</pre>
Arguments: <p>vect1 A pointer to a 16-bit vector which is the output vector of this function.</p> <p>size The number of elements in the output vector.</p> <p>f The frequency of the output signal.</p> <p>fs The sampling frequency of the output signal.</p> <p>delay The delay of the periodic waveform. The delay must fit in the range [0; 1] where 1 to T.</p>
Return Value: NONE
Remarks: NONE



Description: 32-bit fixed point version of the dirac comb signal generation.
Module: SIGNAL_GENERATION
Prototype: <pre> void dsp32_gen_dcomb (dsp32_t *vect1, int size, int f, int fs, dsp32_t delay); </pre>
Arguments: <p>vect1 A pointer to a 32-bit vector which is the output vector of this function.</p> <p>size The number of elements in the output vector.</p> <p>f The frequency of the output signal.</p> <p>fs The sampling frequency of the output signal.</p> <p>delay The delay of the periodic waveform. The delay must fit in the range [0; 1] where 1 to T.</p>
Return Value: NONE
Remarks: NONE

4.3.2.7 Noise

Description:	16-bit fixed point version of the noise generation.
Module:	SIGNAL_GENERATION
Prototype:	<pre>void dsp16_gen_noise (dsp16_t *vect1, int size, dsp16_t amp);</pre>
Arguments:	<p>vect1 A pointer to a 16-bit vector which is the output vector of this function.</p> <p>size The number of elements in the output vector.</p> <p>amp The amplitude of the signal.</p>
Return Value:	NONE
Remarks:	NONE

Description:	32-bit fixed point version of the noise generation.
Module:	SIGNAL_GENERATION
Prototype:	<pre>void dsp32_gen_noise (dsp32_t *vect1, int size, dsp32_t amp);</pre>
Arguments:	<p>vect1 A pointer to a 32-bit vector which is the output vector of this function.</p> <p>size The number of elements in the output vector.</p> <p>amp The amplitude of the signal.</p>
Return Value:	NONE
Remarks:	NONE





4.3.2.8 Ramp

Description: 16-bit fixed point version of the ramp signal generation.
Module: SIGNAL_GENERATION
Prototype: <pre>void dsp16_gen_ramp (dsp16_t *vect1, int size, dsp16_t increment);</pre>
Arguments: vect1 A pointer to a 16-bit vector which is the output vector of this function. size The number of elements in the output vector. increment The incrementation of the signal per vector's element.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the ramp signal generation.
Module: SIGNAL_GENERATION
Prototype: <pre>void dsp32_gen_ramp (dsp32_t *vect1, int size, dsp32_t increment);</pre>
Arguments: vect1 A pointer to a 32-bit vector which is the output vector of this function. size The number of elements in the output vector. increment The incrementation of the signal per vector's element.
Return Value: NONE
Remarks: NONE

4.3.2.9 Step

Description: 16-bit fixed point version of the dirac comb signal generation.
Module: SIGNAL_GENERATION
Prototype: <pre>void dsp16_gen_step (dsp16_t *vect1, int size, dsp16_t intial_value, dsp16_t final_value, int step_index);</pre>
Arguments: vect1 A pointer to a 16-bit vector which is the output vector of this function. size The number of elements in the output vector. intial_value The initial value of the signal. final_value The final value of the signal. step_index The index in the vector, where the step appears.
Return Value: NONE
Remarks: NONE



Description: 32-bit fixed point version of the dirac comb signal generation.
Module: SIGNAL_GENERATION
Prototype: <pre>void dsp32_gen_step (dsp32_t *vect1, int size, dsp32_t intial_value, dsp32_t final_value, int step_index);</pre>
Arguments: vect1 A pointer to a 32-bit vector which is the output vector of this function. size The number of elements in the output vector. intial_value The initial value of the signal. final_value The final value of the signal. step_index The index in the vector, where the step appears.
Return Value: NONE
Remarks: NONE

4.3.2.10 Dirac

Description: 16-bit fixed point version of the dirac comb signal generation.
Module: SIGNAL_GENERATION
Prototype: <pre>void dsp16_gen_dirac (dsp16_t *vect1, int size, int dirac_index);</pre>
Arguments: vect1 A pointer to a 16-bit vector which is the output vector of this function. size The number of elements in the output vector. dirac_index The index in the vector, where the dirac appears.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the dirac comb signal generation.
Module: SIGNAL_GENERATION
Prototype: <pre>void dsp32_gen_dirac (dsp32_t *vect1, int size, int dirac_index);</pre>
Arguments: vect1 A pointer to a 32-bit vector which is the output vector of this function. size The number of elements in the output vector. dirac_index The index in the vector, where the dirac appears.
Return Value: NONE
Remarks: NONE



4.4 Transforms [Basic]

All the transforms functions currently supported by the DSP library.

- Complex Fast Fourier Transform
- Complex inverse Fast Fourier Transform
- Real to complex Fast Fourier Transform

4.4.1 Standard description

4.4.1.1 Complex Fast Fourier Transform

This function computes a complex FFT from an input signal. It uses the Radix-4 "Decimate In Time" algorithm and does not perform a calculation "in place" which means that the input vector has to be different from the output vector. Here is the formula of the FFT:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-\frac{2\pi j n k}{N}}$$

Relative functions:

- dsp16_trans_complexfft

4.4.1.2 Complex Inverse Fast Fourier Transform

This function computes a complex inverse FFT from an input signal. It uses the Radix-4 "Decimate In Time" algorithm and does not perform a calculation "in place" which means that the input vector has to be different from the output vector. Here is the formula of the iFFT:

$$x(k) = \sum_{n=0}^{N-1} X(n) e^{\frac{2\pi j n k}{N}}$$

Relative functions:

- dsp16_trans_complexifft

4.4.1.3 Real to Complex Fast Fourier Transform

This function computes a complex FFT from a real input signal. It uses the Radix-4 "Decimate In Time" algorithm and does not perform a calculation "in place" which means that the input vector has to be different from the output vector. Here is the formula of the FFT:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-\frac{2\pi j n k}{N}}$$

Relative functions:

- dsp16_trans_realcomplexfft
- dsp32_trans_realcomplexfft

4.4.2 C description

4.4.2.1 Complex Fast Fourier Transform

Description: 16-bit fixed point version of the complex FFT algorithm.
Module: TRANSFORMATION
Prototype: <pre>void dsp16_trans_complexfft (dsp16_complex_t *vect1, dsp16_complex_t *vect2, int nlog);</pre>
Arguments: <p>vect1 A pointer to a 16-bit complex vector which is the output vector of this function.</p> <p>vect2 A pointer to a 16-bit complex vector which is the input vector of this function.</p> <p>nlog It is the base-2-logarithm of the size of the input/output vector. Due to its implementation, this function computes only 4ⁿ-point complex FFT. Therefore, the \b nlog argument has to be even.</p>
Return Value: NONE
Remarks: The size of the output vector has to be the same as the size of the input vector. To avoid overflowing values, the resulting vector amplitude is scaled by 2 ^{nlog} . This function uses a static twiddle factors table.



4.4.2.2 Complex Inverse Fast Fourier Transform

Description: 16-bit fixed point version of the complex iFFT algorithm.
Module: TRANSFORMATION
Prototype: <pre>void dsp16_trans_complexifft (dsp16_complex_t *vect1, dsp16_complex_t *vect2, int nlog);</pre>
Arguments: <p>vect1 A pointer to a 16-bit complex vector which is the output vector of this function.</p> <p>vect2 A pointer to a 16-bit complex vector which is the input vector of this function.</p> <p>nlog It is the base-2-logarithm of the size of the input/output vector. Due to its implementation, this function computes only 4^n-point complex iFFT. Therefore, the nlog argument has to be even.</p>
Return Value: NONE
Remarks: The size of the output vector has to be the same as the size of the input vector. To avoid overflowing values, the resulting vector amplitude is scaled by 2^{nlog} . This function uses a static twiddle factors table.

4.4.2.3 Real to Complex Fast Fourier Transform

Description: 16-bit fixed point version of the real to complex FFT algorithm.
Module: TRANSFORMATION
Prototype: <pre>void dsp16_trans_realcomplexfft (dsp16_complex_t *vect1, dsp16_t *vect2, int nlog);</pre>
Arguments: <p>vect1 A pointer to a 16-bit complex vector which is the output vector of this function.</p> <p>vect2 A pointer to a 16-bit real vector which is the input vector of this function.</p> <p>nlog It is the base-2-logarithm of the size of the input/output vector. Due to its implementation, this function computes only 4ⁿ-point complex FFT. Therefore, the nlog argument has to be even.</p>
Return Value: NONE
Remarks: The size of the output vector has to be the same as the size of the input vector. To avoid overflowing values, the resulting vector amplitude is scaled by 2 ^{nlog} . This function uses a static twiddle factors table.

dsp16_trans_realcomplexfft (See complex_fft_example.c and complex_fft_example.sce files)**Example:**

Using Scilab,

Generate a **x** input signal made of a combinaison of 2 sinus.

Compute FFT Transformation and store value in **y** vector:

```
y      = [-0.048306      0.000000 i
          ...
          -0.138689      0.010377 i
          -0.058407      0.000356 i]
```

C source code using the DSPLib with the pre-calculated **x** signal.

```
vect1 = [-0.048370 + 0.i77563
          ...
          -0.138641 + 0.010375i
          -0.058349 + 0.000335i];
```



Description: 32-bit fixed point version of the real to complex FFT algorithm.
Module: TRANSFORMATION
Prototype: <pre>void dsp32_trans_realcomplexfft (dsp32_complex_t *vect1, dsp32_t *vect2, int nlog);</pre>
Arguments: <p>vect1 A pointer to a 32-bit complex vector which is the output vector of this function.</p> <p>vect2 A pointer to a 32-bit real vector which is the input vector of this function.</p> <p>nlog It is the base-2-logarithm of the size of the input/output vector. Due to its implementation, this function computes only 4^n-point complex FFT. Therefore, the nlog argument has to be even.</p>
Return Value: NONE
Remarks: The size of the output vector has to be the same as the size of the input vector. To avoid overflowing values, the resulting vector amplitude is scaled by 2^{nlog} . This function uses a static twiddle factors table.

4.5 Vectors [Basic]

All the vector management functions currently supported by the DSP library.

- Real
 - Addition
 - Addition with a real
 - Subtraction
 - Subtraction with a real
 - Multiplication with a real
 - Division with a real
 - Dot multiplication
 - Dot division
 - Multiplication with an integer
 - Division with an integer
 - Power
 - Minimum
 - Maximum
 - Negate
 - Zero padding
 - Copy
 - Partial Convolution
 - Convolution
- Complex
 - Complex addition
 - Complex subtraction
 - Complex absolute
 - Complex conjugate

4.5.1 Standard description

4.5.1.1 Addition

This function adds two vectors and store the result into another one.

Relative functions:

- `dsp16_vect_add`
- `dsp32_vect_add`

4.5.1.2 Addition with a real

This function adds each items of a vector with a real number and store the result into another vector.

Relative functions:

- `dsp16_vect_realadd`
- `dsp32_vect_realadd`





4.5.1.3 Subtraction

This function subtracts two vectors and store the result into another one.

Relative functions:

- dsp16_vect_sub
- dsp32_vect_sub

4.5.1.4 Subtraction with a real

This function subtracts each items of a vector with a real number and store the result into another vector.

Relative functions:

- dsp16_vect_realsub
- dsp32_vect_realsub

4.5.1.5 Multiplication with a real

This function multiplies one vector with a real number and store the result into another vector.

Relative functions:

- dsp16_vect_realmul
- dsp32_vect_realmul

4.5.1.6 Division with a real

This function divides one vector with a real number and store the result into another vector.

Relative functions:

- dsp16_vect_realdiv
- dsp32_vect_realdiv

4.5.1.7 Multiplication with an integer

This function multiplies one vector with an integer and store the result into another vector.

Relative functions:

- dsp16_vect_intmul
- dsp32_vect_intmul

4.5.1.8 Division with an integer

This function divides one vector with an integer and store the result into another vector.

Relative functions:

- `dsp16_vect_intdiv`
- `dsp32_vect_intdiv`

4.5.1.9 Dot multiplication

This function multiplies two vectors point per point and store the result into another one.

Relative functions:

- `dsp16_vect_dotmul`
- `dsp32_vect_dotmul`

4.5.1.10 Dot division

This function divides two vectors point per point and store the result into another one.

Relative functions:

- `dsp16_vect_dotdiv`
- `dsp32_vect_dotdiv`

4.5.1.11 Power

These functions compute $\text{vect}^2^{\text{real}}$.

Note:

The output will be limit in the range of the fixed point format used.

Relative functions:

- `dsp16_vect_pow`
- `dsp32_vect_pow`

4.5.1.12 Minimum

This function retrieves the minimum of a vector.

Relative functions:

- `dsp16_vect_min`
- `dsp32_vect_min`



4.5.1.13 Maximum

This function retrieves the maximum of a vector.

Relative functions:

- `dsp16_vect_max`
- `dsp32_vect_max`

4.5.1.14 Negate

This function negate a vector.

Relative functions:

- `dsp16_vect_neg`
- `dsp32_vect_neg`

4.5.1.15 Zero padding

This function zero pads the tail of the vector.

Relative functions:

- `dsp16_vect_zeropad`
- `dsp32_vect_zeropad`

4.5.1.16 Copy

This function copy a vector into another vector.

Relative functions:

- `dsp16_vect_copy`
- `dsp32_vect_copy`

4.5.1.17 Partial Convolution

This function performs a linear partial convolution between two discrete sequences. It returns a signal of a length equals to $(\text{vect2_size} - \text{vect3_size} + 1)$ elements.

Note:

The two discrete sequences have to be scaled to avoid overflowing values.
All the vectors have to be 32-bits aligned.

Relative functions:

- `dsp16_vect_convpart`
- `dsp32_vect_convpart`

4.5.1.18 Convolution

This function performs a linear convolution between two discrete sequences. It returns a signal of a length equals to $(\text{vect2_size} + \text{vect3_size} - 1)$ elements.

Note:

The two discrete sequences have to be scaled to avoid overflowing values. All the vectors have to be 32-bits aligned.

Warning:

The output vector of the function has to have at least a length of $N + 2 * M$  2 elements because of internal computations, where N is the length of the largest input vector and M, the length of the smallest input vector.

Relative functions:

- `dsp16_vect_conv`
- `dsp32_vect_conv`

4.5.1.19 Complex addition

This function adds two complex vectors and store the result into another one.

Relative functions:

- `dsp16_vect_complex_add`
- `dsp32_vect_complex_add`

4.5.1.20 Complex subtraction

This function sub two complex vectors and store the result into another one.

Relative functions:

- `dsp16_vect_complex_sub`
- `dsp32_vect_complex_sub`

4.5.1.21 Complex absolute

This function returns the absolute value of a complex vector.

Relative functions:

- `dsp16_vect_complex_abs`
- `dsp32_vect_complex_abs`

4.5.1.22 Complex conjugate

This function returns the conjugate complex vector of the input.

Relative functions:

- `dsp16_vect_complex_conj`





4.5.2 C description

4.5.2.1 Addition

Description: 16-bit fixed point version of the vector addition function.
Module: VECTORS
Prototype: <pre>void dsp16_vect_add (dsp16_t *vect1, dsp16_t *vect2, dsp16_t *vect3, int size);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the result of the addition. vect2 A pointer to the 16-bit real vector that will be added with the other input vector (vect3). vect3 A pointer to the 16-bit real vector that will be added with the other input vector (vect2). size The size of the input vectors.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the vector addition function.
Module: VECTORS
Prototype: <pre>void dsp32_vect_add (dsp32_t *vect1, dsp32_t *vect2, dsp32_t *vect3, int size);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the result of the addition. vect2 A pointer to the 32-bit real vector that will be added with the other input vector (vect3). vect3 A pointer to the 32-bit real vector that will be added with the other input vector (vect2). size The size of the input vectors.
Return Value: NONE
Remarks: NONE





4.5.2.2 Addition with a real

Description: 16-bit fixed point version of the vector addition with a real number.
Module: VECTORS
Prototype: <pre>void dsp16_vect_realadd (dsp16_t *vect1, dsp16_t *vect2, int size, dsp16_t real);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the result of the operation. vect2 A pointer to the 16-bit real vector that will be added with the real number. size The size of the vectors. real The real number to be added with the vector (vect2).
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the vector addition with a real number.
Module: VECTORS
Prototype: <pre>void dsp32_vect_realadd (dsp32_t *vect1, dsp32_t *vect2, int size, dsp16_t real);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the result of the operation. vect2 A pointer to the 32-bit real vector that will be added with the real number. size The size of the vectors. real The real number to be added with the vector (vect2).
Return Value: NONE
Remarks: NONE



4.5.2.3 Subtraction

Description: 16-bit fixed point version of the vector subtraction function.
Module: VECTORS
Prototype: <pre>void dsp16_vect_sub (dsp16_t *vect1, dsp16_t *vect2, dsp16_t *vect3, int size);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the result of the subtraction. vect2 A pointer to the 16-bit real vector that will be subtracted with the other input vector (vect3). vect3 A pointer to the 16-bit real vector that will be subtracted with the other input vector (vect2). size The size of the input vectors.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the vector subtraction function.
Module: VECTORS
Prototype: <pre>void dsp32_vect_sub (dsp32_t *vect1, dsp32_t *vect2, dsp32_t *vect3, int size);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the result of the subtraction. vect2 A pointer to the 32-bit real vector that will be subtracted with the other input vector (vect3). vect3 A pointer to the 32-bit real vector that will be subtracted with the other input vector (vect2). size The size of the input vectors.
Return Value: NONE
Remarks: NONE



4.5.2.4 Subtraction with a real

Description: 16-bit fixed point version of the vector subtraction with a real number.
Module: VECTORS
Prototype: <pre>void dsp16_vect_realsub (dsp16_t *vect1, dsp16_t *vect2, int size, dsp16_t real);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the result of the operation. vect2 A pointer to the 16-bit real vector that will be subtracted with the real number. size The size of the vectors. real The real number to be subtracted with the vector (vect2).
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the vector subtraction with a real number.
Module: VECTORS
Prototype: <pre>void dsp32_vect_realsub (dsp32_t *vect1, dsp32_t *vect2, int size, dsp32_t real);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the result of the operation. vect2 A pointer to the 32-bit real vector that will be subtracted with the real number. size The size of the vectors. real The real number to be subtracted with the vector (vect2).
Return Value: NONE
Remarks: NONE



4.5.2.5 Multiplication with a real

Description: 16-bit fixed point version of the vector multiplication with a real number.
Module: VECTORS
Prototype: <pre>void dsp16_vect_realmul (dsp16_t *vect1, dsp16_t *vect2, int size, dsp16_t real);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the result of the operation. vect2 A pointer to the 16-bit real vector that will be multiplied with the real number. size The size of the vectors. real The real number to be multiplied with the vector (vect2).
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the vector multiplication with a real number.
Module: VECTORS
Prototype: <pre>void dsp32_vect_realmul (dsp32_t *vect1, dsp32_t *vect2, int size, dsp32_t real);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the result of the operation. vect2 A pointer to the 32-bit real vector that will be multiplied with the real number. size The size of the vectors. real The real number to be multiplied with the vector (vect2).
Return Value: NONE
Remarks: NONE





4.5.2.6 Division with a real

Description: 16-bit fixed point version of the vector division with a real number.
Module: VECTORS
Prototype: <pre>void dsp16_vect_realdiv (dsp16_t *vect1, dsp16_t *vect2, int size, dsp16_t real);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the result of the operation. vect2 A pointer to the 16-bit real vector that will be divided with the real number. size The size of the vectors. real The real number to be divided with the vector (vect2).
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the vector division with a real number.
Module: VECTORS
Prototype: <pre>void dsp32_vect_realdiv (dsp32_t *vect1, dsp32_t *vect2, int size, dsp32_t real);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the result of the operation. vect2 A pointer to the 32-bit real vector that will be divided with the real number. size The size of the vectors. real The real number to be divided with the vector (vect2).
Return Value: NONE
Remarks: NONE





4.5.2.7 Multiplication with an integer

Description: 16-bit fixed point version of the vector multiplication with an integer.
Module: VECTORS
Prototype: <pre>void dsp16_vect_intmul (dsp16_t *vect1, dsp16_t *vect2, int size, int integer);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the result of the operation. vect2 A pointer to the 16-bit real vector that will be multiplied with the integer. size The size of the vectors. integer The integer to be multiplied with the vector (vect2).
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the vector multiplication with an integer.
Module: VECTORS
Prototype: <pre>void dsp32_vect_intmul (dsp32_t *vect1, dsp32_t *vect2, int size, int integer);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the result of the operation. vect2 A pointer to the 32-bit real vector that will be multiplied with the integer. size The size of the vectors. integer The integer to be multiplied with the vector (vect2).
Return Value: NONE
Remarks: NONE



4.5.2.8 Division with an integer

Description: 16-bit fixed point version of the vector division with an integer.
Module: VECTORS
Prototype: <pre>void dsp16_vect_intdiv (dsp16_t *vect1, dsp16_t *vect2, int size, int integer);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the result of the operation. vect2 A pointer to the 16-bit real vector that will be divided with the integer. size The size of the vectors. integer The integer to be divided with the vector (vect2).
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the vector division with an integer.
Module: VECTORS
Prototype: <pre>void dsp32_vect_intdiv (dsp32_t *vect1, dsp132_t *vect2, int size, int integer);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the result of the operation. vect2 A pointer to the 32-bit real vector that will be divided with the integer. size The size of the vectors. integer The integer to be divided with the vector (vect2).
Return Value: NONE
Remarks: NONE





4.5.2.9 Dot multiplication

Description: 16-bit fixed point version of the vector dot multiplication function.
Module: VECTORS
Prototype: <pre>void dsp16_vect_dotmul (dsp16_t *vect1, dsp16_t *vect2, dsp16_t *vect3, int size);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will received the result of the dot multiplication. vect2 A pointer to the 16-bit real vector that will be multiplied with the other input vector (vect3). vect3 A pointer to the 16-bit real vector that will be multiplied with the other input vector (vect2). size The size of the input vectors.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the vector dot multiplication function.
Module: VECTORS
Prototype: <pre>void dsp32_vect_dotmul (dsp32_t *vect1, dsp32_t *vect2, dsp32_t *vect3, int size);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the result of the dot multiplication. vect2 A pointer to the 32-bit real vector that will be multiplied with the other input vector (vect3). vect3 A pointer to the 32-bit real vector that will be multiplied with the other input vector (vect2). size The size of the input vectors.
Return Value: NONE
Remarks: NONE





4.5.2.10 Dot division

Description: 16-bit fixed point version of the vector dot division function.
Module: VECTORS
Prototype: <pre>void dsp16_vect_dotdiv (dsp16_t *vect1, dsp16_t *vect2, dsp16_t *vect3, int size);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the result of the dot division. vect2 A pointer to the 16-bit real vector that will be divided with the other input vector (vect3). vect3 A pointer to the 16-bit real vector that will be divided with the other input vector (vect2). size The size of the input vectors.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the vector dot division function.
Module: VECTORS
Prototype: <pre>void dsp32_vect_dotdiv (dsp32_t *vect1, dsp32_t *vect2, dsp32_t *vect3, int size);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the result of the dot division. vect2 A pointer to the 32-bit real vector that will be divided with the other input vector (vect3). vect3 A pointer to the 32-bit real vector that will be divided with the other input vector (vect2). size The size of the input vectors.
Return Value: NONE
Remarks: NONE



4.5.2.11 Power

Description: 16-bit fixed point version of the power function.
Module: VECTORS
Prototype: <pre>void dsp16_vect_pow (dsp16_t *vect1, dsp16_t *vect2, int size, dsp16_t real);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the result of the operation. vect2 A pointer to the 16-bit real vector that will be raised to the power 'real'. size The size of the vectors. real The real number used to raised to the power 'vect2'.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the power function.
Module: VECTORS
Prototype: <pre>void dsp32_vect_pow (dsp32_t *vect1, dsp32_t *vect2, int size, dsp32_t real);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the result of the operation. vect2 A pointer to the 32-bit real vector that will be raised to the power 'real'. size The size of the vectors. real The real number used to raised to the power 'vect2'.
Return Value: NONE
Remarks: NONE

4.5.2.12 Minimum

Description: 16-bit fixed point version of the vector minimum function.
Module: VECTORS
Prototype: <pre>dsp16_t dsp16_vect_min (dsp16_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that is used to find its minimum. size The size of the input vector.
Return Value: The minimum of the vector (vect1).
Remarks: NONE

Description: 32-bit fixed point version of the vector minimum function.
Module: VECTORS
Prototype: <pre>dsp32_t dsp32_vect_min (dsp32_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that is used to find its minimum. size The size of the input vector.
Return Value: The minimum of the vector (vect1).
Remarks: NONE



4.5.2.13 Maximum

Description: 16-bit fixed point version of the vector maximum function.
Module: VECTORS
Prototype: <pre>dsp16_t dsp16_vect_max (dsp16_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that is used to find its maximum. size The size of the input vector.
Return Value: The maximum of the vector (vect1).
Remarks: NONE

Description: 32-bit fixed point version of the vector maximum function.
Module: VECTORS
Prototype: <pre>dsp32_t dsp32_vect_max (dsp32_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that is used to find its maximum. size The size of the input vector.
Return Value: The maximum of the vector (vect1).
Remarks: NONE

4.5.2.14 Negate

Description: 16-bit fixed point version of the vector negate function.
Module: VECTORS
Prototype: <pre>void dsp16_vect_neg (dsp16_t *vect1, dsp16_t *vect2, int size);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the result. vect2 A pointer to the 16-bit real vector that will be negate. size The size of the input vector.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the vector negate function.
Module: VECTORS
Prototype: <pre>void dsp32_vect_neg (dsp32_t *vect1, dsp32_t *vect2, int size);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the result. vect2 A pointer to the 32-bit real vector that will be negate. size The size of the input vector.
Return Value: NONE
Remarks: NONE





4.5.2.15 Zero padding

Description: 16-bit fixed point version of the zero padding function.
Module: VECTORS
Prototype: <pre>inline static void dsp16_vect_zeropad (dsp16_t *vect1, int size, int num_zero);</pre>
Arguments: vect1 A pointer to the 16 bits real vector that has to be padded with zeros. size The size of this vector. num_zero The number of zeros to pad at the end of the vector.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the zero padding function.
Module: VECTORS
Prototype: <pre>inline static void dsp32_vect_zeropad (dsp32_t *vect1, int size, int num_zero);</pre>
Arguments: vect1 A pointer to the 32 bits real vector that has to be padded with zeros. size The size of this vector. num_zero The number of zeros to pad at the end of the vector.
Return Value: NONE
Remarks: NONE

4.5.2.16 Copy

Description:	16-bit fixed point version of the copy function.
Module:	VECTORS
Prototype:	<pre> inline static void dsp16_vect_copy (dsp16_t *vect1, dsp16_t *vect2, int size) </pre>
Arguments:	<p>vect1 A pointer to the 16 bits real vector that contains the data.</p> <p>vect2 A pointer to the 16 bits real vector to be copied to.</p> <p>size The size of those vectors.</p>
Return Value:	NONE
Remarks:	NONE

Description:	32-bit fixed point version of the copy function.
Module:	VECTORS
Prototype:	<pre> inline static void dsp32_vect_copy (dsp32_t *vect1, dsp32_t *vect2, int size) </pre>
Arguments:	<p>vect1 A pointer to the 32 bits real vector that contains the data.</p> <p>vect2 A pointer to the 32 bits real vector to be copied to.</p> <p>size The size of those vectors.</p>
Return Value:	NONE
Remarks:	NONE

4.5.2.17 Partial Convolution

Description: 16-bit fixed point version of the Partial Convolution.
Module: VECTORS
Prototype: <pre>void dsp16_vect_convpart (dsp16_t *vect1, dsp16_t *vect2, int vect2_size, dsp16_t *vect3, int vect3_size);</pre>
Arguments: vect1 A pointer to a 16-bits fixed-point output vector. vect2 A pointer to a 16-bits fixed-point input vector. vect2_size The length of the first input vector (must be greater or equal to 4). vect3 A pointer to a 16-bits fixed-point input vector. vect3_size The length of the second input vector (must be greater or equal to 8)
Return Value: NONE
Remarks: Due to its implementation, for the avr32-uc3 optimized version of the FIR, the output vector (vect1) has to have a length of 4*n elements to avoid overflows.

4.5.2.18 Convolution

Description: 16-bit fixed point version of the Convolution.
Module: VECTORS
Prototype: <pre>void dsp16_vect_conv (dsp16_t *vect1, dsp16_t *vect2, int vect2_size, dsp16_t *vect3, int vect3_size);</pre>
Arguments: <p>vect1 A pointer to a 16-bits fixed-point output vector.</p> <p>vect2 A pointer to a 16-bits fixed-point inout vector.</p> <p>vect2_size The length of the first input vector (must be greater or equal to 8).</p> <p>vect3 A pointer to a 16-bits fixed-point input vector.</p> <p>vect3_size The length of the second input vector (must be greater or equal to 8)</p>
Return Value: NONE
Remarks: Due to its implementation, for the avr32-uc3 optimized version of the FIR, the output vector (vect1) has to have a length of 4*n elements to avoid overflows.



Description: 32-bit fixed point version of the Convolution.
Module: VECTORS
Prototype: <pre>void dsp32_vect_conv (dsp32_t *vect1, dsp32_t *vect2, int vect2_size, dsp32_t *vect3, int vect3_size);</pre>
Arguments: vect1 A pointer to a 32-bits fixed-point output vector. vect2 A pointer to a 32-bits fixed-point input vector. vect2_size The length of the first input vector (must be greater or equal to 8). vect3 A pointer to a 32-bits fixed-point input vector. vect3_size The length of the second input vector (must be greater or equal to 8)
Return Value: NONE
Remarks: Due to its implementation, for the avr32-uc3 optimized version of the FIR, the output vector (vect1) has to have a length of 4*n elements to avoid overflows.

dsp32_vect_conv (See convolution_example.c and convolution_example.sce files)
Example: <u>Using Scilab</u> , Generate vect2 and vect3 input signals Compute a convolution of this two signals and obtains a vect1 vectors: <pre>vect1 = [-0 ... 0.01246589690 0.00668038504]</pre> <u>C source code</u> using the DSPLib with the pre-calculated vect2 and vect3 signals. <pre>vect1 = [0 ... 0.012420 0.006652];</pre>

4.5.2.19 Complex addition

Description: 16-bit fixed point version of the complex vector addition function.
Module: VECTORS
Prototype: <pre>void dsp16_vect_complex_add (dsp16_complex_t *vect1, dsp16_complex_t *vect2, dsp16_complex_t *vect3, int size);</pre>
Arguments: vect1 A pointer to the 16-bit complex result vector. vect2 A pointer to the first 16-bit complex input vector. vect3 A pointer to the second 16-bit complex input vector. size The size of the input vectors.
Return Value: NONE
Remarks: NONE



Description: 32-bit fixed point version of the complex vector additon function.
Module: VECTORS
Prototype: <pre>void dsp32_vect_complex_add (sp32_complex_t *vect1, dsp32_complex_t *vect2, dsp32_complex_t *vect3, int size);</pre>
Arguments: vect1 A pointer to the 32-bit complex result vector. vect2 A pointer to the first 32-bit complex input vector. vect3 A pointer to the second 32-bit complex input vector. size The size of the input vectors.
Return Value: NONE
Remarks: NONE

4.5.2.20 Complex subtraction

Description: 16-bit fixed point version of the complex vector subtraction function.
Module: VECTORS
Prototype: <pre>void dsp16_vect_complex_sub (dsp16_complex_t *vect1, dsp16_complex_t *vect2, dsp16_complex_t *vect3, int size);</pre>
Arguments: vect1 A pointer to the 16-bit complex result vector. vect2 A pointer to the minuend 16-bit complex vector. vect3 A pointer to the subtrahend 16-bit complex vector. size The size of the input vectors.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the complex vector subtraction function.
Module: VECTORS
Prototype: <pre> void dsp32_vect_complex_sub (dsp32_complex_t *vect1, dsp32_complex_t *vect2, dsp32_complex_t *vect3, int size); </pre>
Arguments: vect1 A pointer to the 32-bit complex result vector. vect2 A pointer to the minuend 32-bit complex vector. vect3 A pointer to the subtrahend 32-bit complex vector. size The size of the input vectors.
Return Value: NONE
Remarks: NONE

4.5.2.21 Complex absolute

Description: 16-bit fixed point version of the complex vector absolute function.
Module: VECTORS
Prototype: <pre>void dsp16_vect_complex_abs (dsp16_t *vect1, dsp16_complex_t *vect2, int size);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the result. vect2 A pointer to the 16-bit complex input vector. size The size of the input vector.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the complex vector absolute function.
Module: VECTORS
Prototype: <pre>void dsp32_vect_complex_abs (dsp32_t *vect1, dsp32_complex_t *vect2, int size);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the result. vect2 A pointer to the 32-bit complex input vector. size The size of the input vector.
Return Value: NONE
Remarks: NONE





4.5.2.22 Complex conjugate

Description: 16-bit fixed point version of the complex vector conjugate function.
Module: VECTORS
Prototype: <pre>void dsp16_vect_complex_conj (dsp16_complex_t *vect1, dsp16_complex_t *vect2, int size);</pre>
Arguments: vect1 A pointer to the 16-bit complex vector that will contain the result. vect2 A pointer to the 16-bit complex input vector. size The size of the input vector.
Return Value: NONE
Remarks: NONE

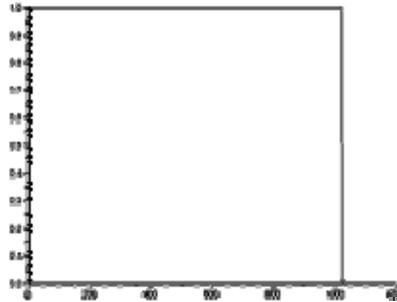
4.6 Windowing [Basic]

All the windowing functions currently supported by the DSP library.

- Rectangular
- Bartlett
- Blackman
- Hamming
- Gauss
- Hann
- Kaiser
- Welch

4.6.1 Standard description

4.6.1.1 Rectangular

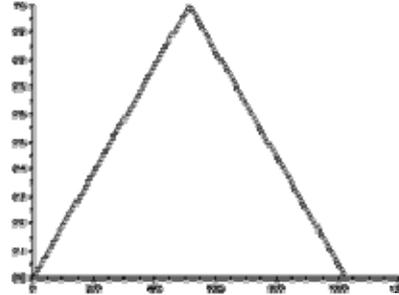


These functions generate a rectangular window that fits in the output vector. The rectangular window filled the output vector with 1.

Relative functions:

- dsp16_win_rect
- dsp32_win_rect

4.6.1.2 Bartlett



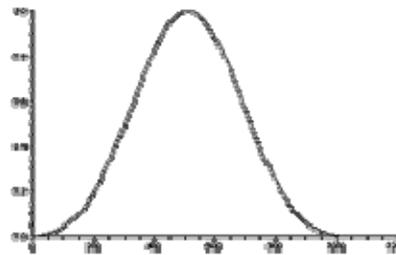
Also known simply as the triangular windows, these functions generate a bartlett window that fits in the output vector.

The amplitude of the signal is in the range [0; 1]

Relative functions:

- `dsp16_win_bart`
- `dsp32_win_bart`

4.6.1.3 Blackman



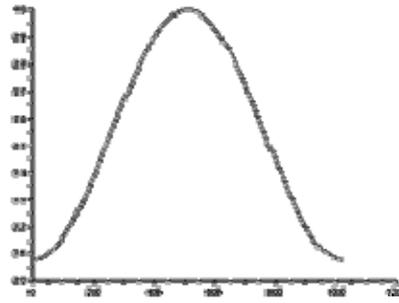
These functions generate a Blackman window that fits in the output vector.

The amplitude of the signal is in the range [0; 1]

Relative functions:

- `dsp16_win_black`
- `dsp32_win_black`

4.6.1.4 Hamming

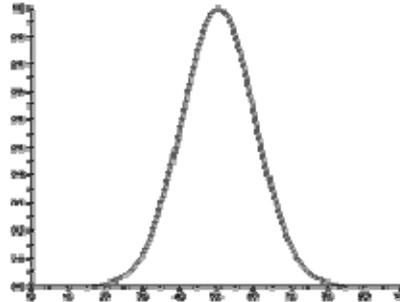


These functions generate a Hamming window that fits in the output vector. The amplitude of the signal is in the range [0; 1]

Relative functions:

- `dsp16_win_hamm`
- `dsp32_win_hamm`

4.6.1.5 Gauss



These functions generate a gaussian window that fits in the output vector. The amplitude of the signal is in the range [0; 1]

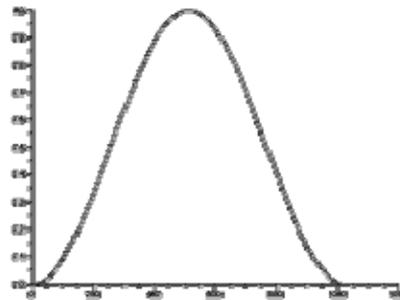
Note:

You can change the value of the teta coefficient by defining DSP_GAUSS_TETA at the compilation with a value that fits in the range]0; 0.5]

Relative functions:

- dsp16_win_gauss
- dsp32_win_gauss

4.6.1.6 Hann

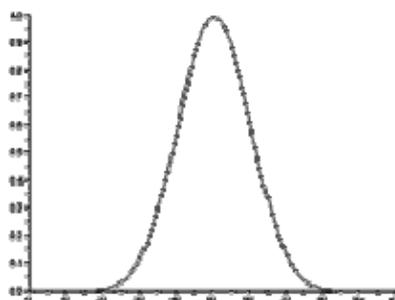


These functions generate a Hann window that fits in the output vector. The amplitude of the signal is in the range [0; 1]

Relative functions:

- dsp16_win_hann
- dsp32_win_hann

4.6.1.7 Kaiser

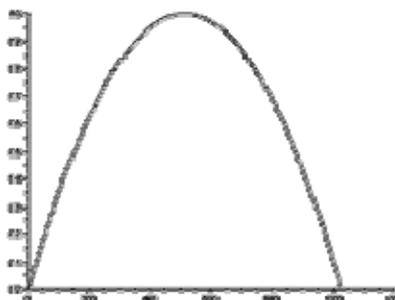


These functions generate a Kaiser window that fits in the output vector. The amplitude of the signal is in the range [0; 1]

Relative functions:

- `dsp16_win_kaiser`
- `dsp32_win_kaiser`

4.6.1.8 Welch



These functions generate a Welch window that fits in the output vector. The welch window is commonly used as a window for power spectral estimation. The amplitude of the signal is in the range [0; 1]

Relative functions:

- `dsp16_win_welch`
- `dsp32_win_welch`



4.6.2 C description

4.6.2.1 Rectangular

Description: 16-bit fixed point version of the rectangular windowing function.
Module: WINDOWING
Prototype: <pre>void dsp16_win_rect (dsp16_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the window. size The size of the output vector.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the rectangular windowing function.
Module: WINDOWING
Prototype: <pre>void dsp32_win_rect (dsp32_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the window. size The size of the output vector.
Return Value: NONE
Remarks: NONE

4.6.2.2 Bartlett

Description: 16-bit fixed point version of the bartlett windowing function.
Module: WINDOWING
Prototype: <pre>void dsp16_win_bart (dsp16_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the window. size The size of the output vector.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the bartlett windowing function.
Module: WINDOWING
Prototype: <pre>void dsp32_win_bart (dsp32_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the window. size The size of the output vector.
Return Value: NONE
Remarks: NONE



4.6.2.3 Blackman

Description: 16-bit fixed point version of the blackman windowing function.
Module: WINDOWING
Prototype: <pre>void dsp16_win_black (dsp16_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the window. size The size of the output vector.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the blackman windowing function.
Module: WINDOWING
Prototype: <pre>void dsp32_win_black (dsp32_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the window. size The size of the output vector.
Return Value: NONE
Remarks: NONE

4.6.2.4 Hamming

Description: 16-bit fixed point version of the hamming windowing function.
Module: WINDOWING
Prototype: <pre>void dsp16_win_hamm (dsp16_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the window. size The size of the output vector.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the hamming windowing function.
Module: WINDOWING
Prototype: <pre>void dsp32_win_hamm (dsp32_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the window. size The size of the output vector.
Return Value: NONE
Remarks: NONE



4.6.2.5 Gauss

Description: 16-bit fixed point version of the gaussian windowing function.
Module: WINDOWING
Prototype: <pre>void dsp16_win_gauss (dsp16_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the window. size The size of the output vector.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the gaussian windowing function.
Module: WINDOWING
Prototype: <pre>void dsp32_win_gauss (dsp32_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the window. size The size of the output vector.
Return Value: NONE
Remarks: NONE

4.6.2.6 Hann

Description: 16-bit fixed point version of the hann windowing function.
Module: WINDOWING
Prototype: <pre>void dsp16_win_hann (dsp16_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the window. size The size of the output vector.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the hann windowing function.
Module: WINDOWING
Prototype: <pre>void dsp32_win_hann (dsp32_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the window. size The size of the output vector.
Return Value: NONE
Remarks: NONE



4.6.2.7 Kaiser

Description: 16-bit fixed point version of the kaiser windowing function.
Module: WINDOWING
Prototype: <pre>void dsp16_win_kaiser (dsp16_t *vect1, int size, int alpha);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the window. size The size of the output vector. alpha The alpha coefficient which must be greater than 0.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the kaiser windowing function.
Module: WINDOWING
Prototype: <pre>void dsp32_win_kaiser (dsp32_t *vect1, int size, int alpha);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the window. size The size of the output vector. alpha The alpha coefficient which must be greater than 0.
Return Value: NONE
Remarks: NONE

4.6.2.8 Welch

Description: 16-bit fixed point version of the welch windowing function.
Module: WINDOWING
Prototype: <pre>void dsp16_win_welch (dsp16_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 16-bit real vector that will contain the window. size The size of the output vector.
Return Value: NONE
Remarks: NONE

Description: 32-bit fixed point version of the welch windowing function.
Module: WINDOWING
Prototype: <pre>void dsp32_win_welch (dsp32_t *vect1, int size);</pre>
Arguments: vect1 A pointer to the 32-bit real vector that will contain the window. size The size of the output vector.
Return Value: NONE
Remarks: NONE

4.7 Advanced API

This group regroups all the functions available in the advanced library.

- ADPCM
- Re-Sampling

4.7.1 Standard description

4.7.1.1 Re-Sampling

All the signal re-sampling functions implemented in the DSP advanced library. Following is a brief description of the frequency re-sampling algorithm used in this module. It is aimed for anybody so no specifics digital signal processing knowledges are required to understand this presentation.

- Summary

The principle is simple, it consists in 2 main stages, up-sampling the signal frequency by an integer value (**L**), this action is also called **interpolation**, and then down-sampling it by another integer value (**M**), also known as **decimation**.

- L and M calculation

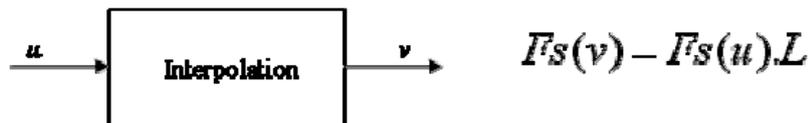
L and M are 2 integers that are calculated by getting the **GCD** (Greatest Common Divisor) of the input (**Fsin**) and the output (**Fsout**) frequencies.

The number resulting will divide Fsin and Fsout to respectively give M and L.

$$M = \frac{F_{s_{in}}}{\text{gcd}(F_{s_{in}}, F_{s_{out}})} ; L = \frac{F_{s_{out}}}{\text{gcd}(F_{s_{in}}, F_{s_{out}})}$$

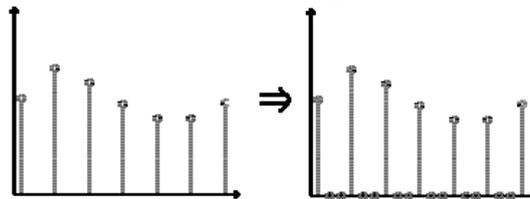
- Interpolation (frequency up-sampling)

This process up samples the frequency of the input signal by an integer factor. The factor used at this stage of the process by the re-sampling algorithm will be the pre-calculated "interpolation factor" **L**. Which means, if we consider this process as a black box with 1 input (**u**) and 1 output (**v**), the output signal sampling frequency (**F_s(v)**) will be equals to the input signal sampling frequency (**F_s(u)**) multiplied by L.



The following describes the algorithm used to implement the interpolation.

The method consists in **extending** the signal by filling "blank spaces" with **zeros**, in order to obtain a signal with the desired sampling rate.

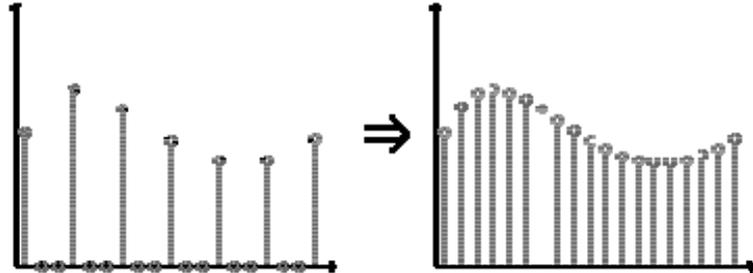


Then this signal goes through a filter in order to remove the highest frequencies, to give it back its original shape. The cut off frequency is calculated according to the input frequency of the signal.

The filter used in this algorithm, is most likely a **lowpass FIR** filter, which is used as a **poly-phase** filter. This optimizes greatly the performances of this process because

poly-phase filters are simply, classical filters cut into pieces. And in this case, the aim is to have one piece with the original samples and the other with the zeros used to up sample the signal.

Then, by **re-ordering** the **coefficients** in a certain way, this process is equivalent to apply a filter only on the original sample parts since the result of filtering a null signal is a null signal.



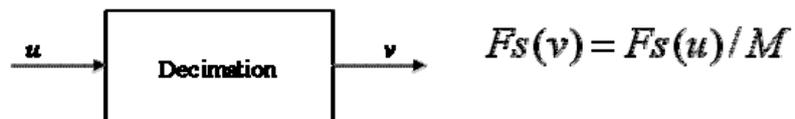
Now, the signal is interpolated, it needs to be down sampled.

- Decimation (frequency down-sampling)

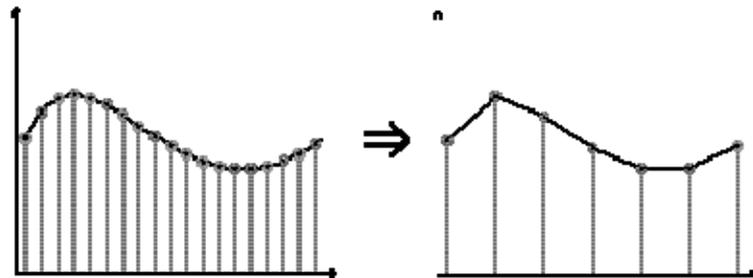
This process is much simpler than the interpolation.

It just consists in **removing samples** in order to keep the same signal wave form but with a lower sampling rate.

Therefore, to obtain the desired output sampling frequency, the signal has to be down sampled by **M** (decimation factor).



Every M samples are kept from the input signal and all the others are removed.



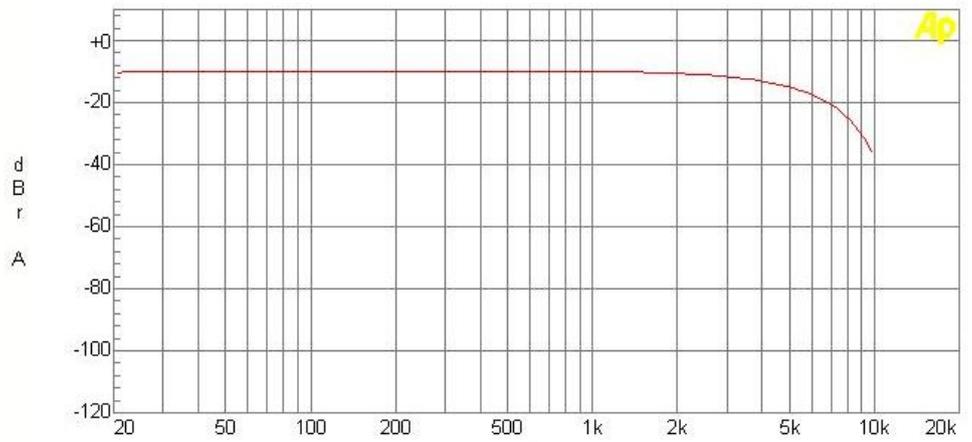
- Conclusion

By processing these 2 main stages, the signal is re-sampled by a factor equals to **L/M**. Therefore, the smaller the 2 frequencies have their GCD (Greatest Common Divisor), the more memory it will need (to store the FIR filter coefficients).

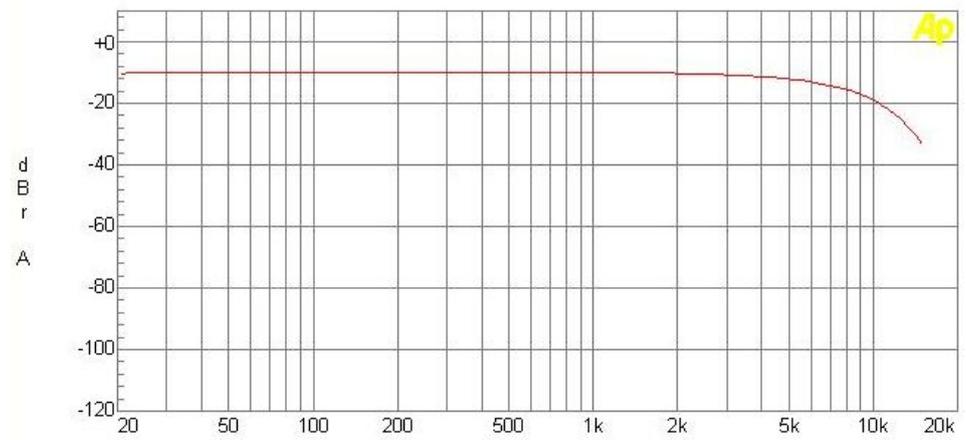
This method is one of the most used in digital signal processing systems. It will generate a clean signal and evaluate at best the waveform of the output signal.

- Frequency response

The signal is attenuated on high frequencies. Following are traces showing the frequency response of the re-sampling algorithm over different sampling rate conversions.



Frequency response from 32KHz to 44.1KHz



Frequency response from 48KHz to 48.51KHz

4.7.2 C description

4.7.2.1 ADPCM

Description: IMA/DVI ADPCM sample encoder.
Module: ADVANCED
Prototype: <pre>S8 dsp_adpcm_ima_encode_nibble (S16 nibble, S16 *step_index, S16 *predicted_value);</pre>
Arguments: <p>nibble The sample to encode.</p> <p>step_index A pointer to a 16 bits data which contains the current step index of the ADPCM algorithm.</p> <p>predicted_value A pointer to a 16 bits data which contains the current predicted value of the ADPCM algorithm.</p>
Return Value: A 4-bit data that corresponds to the sample encoded.
Remarks: NONE

Description: IMA/DVI ADPCM sample decoder.
Module: ADVANCED
Prototype: <pre>S16 dsp_adpcm_ima_decode_nibble (S8 nibble, S16 *step_index, S16 *predicted_value);</pre>
Arguments: <p>nibble The sample to decode.</p> <p>step_index A pointer to a 16 bits data which contains the current step index of the ADPCM algorithm.</p> <p>predicted_value A pointer to a 16 bits data which contains the current predicted value of the ADPCM algorithm.</p>
Return Value: A 4-bit data that corresponds to the sample encoded.
Remarks: NONE





Description: IMA/DVI ADPCM encoder.
Module: ADVANCED
Prototype: <pre>void dsp_adpcm_ima_encode (void *out, S16 *in, int size, S16 *step_index, S16 *predicted_value);</pre>
Arguments: out A 4-bit data vector that will contain the encoded data. in A 16-bit data vector that contains the data to encode. size The number of data to encode. step_index A pointer to a 16 bits data which contains the current step index of the ADPCM algorithm. predicted_value A pointer to a 16 bits data which contains the current predicted value of the ADPCM algorithm.
Return Value: NONE
Remarks: Can be performed "in-place".

Description: IMA/DVI ADPCM decoder.
Module: ADVANCED
Prototype: <pre>void dsp_adpcm_ima_decode (S16 *out, void *in, int size, S16 *step_index, S16 *predicted_value);</pre>
Arguments: out A 4-bit data vector that will contain the decoded data. in A 16-bit data vector that contains the data to decode. size The number of data to encode. step_index A pointer to a 16 bits data which contains the current step index of the ADPCM algorithm. predicted_value A pointer to a 16 bits data which contains the current predicted value of the ADPCM algorithm.
Return Value: NONE
Remarks: Can be performed "in-place".





4.7.2.2 Re-Sampling

Description: This function is the 16-bit signal resampling setup function.
Module: ADVANCED
Prototype: <pre>dsp_resampling_t *dsp16_resampling_setup (int input_sample_rate, int output_sample_rate, int buffer_size, int filter_order, int nb_channels, malloc_fct_t malloc_fct, dsp_resampling_options_t options);</pre>
Arguments: input_sample_rate The sample rate of the input signal. output_sample_rate The sample rate of the output signal. buffer_size The size of the input vectors. filter_order The order of the filter to be used. malloc_fct A pointer to a memory allocation function. nb_channels The number of channels to compute. options Add specific options to the algorithm.
Return Value: A pointer to a structure containing the context that will be used during the re-sampling process.
Remarks: It has to be called only once at the initialization of the resampling process. The output must be freed with the <code>dsp16_resampling_free</code> function once the re-sampling process is completed.

Description: Function used to free the previously allocated structure issued by the <code>dsp16_resampling_setup</code> function.
Module: ADVANCED
Prototype: <pre>void dsp16_resampling_free (dsp_resampling_t *dsp_resampling, free_fct_t free_fct);</pre>
Arguments: dsp_resampling The re-sampling context structure to be freed. free_fct A pointer to the free function to be used.
Return Value: NONE
Remarks: NONE

Description: Returns the maximal length in sample of the output signal.
Module: ADVANCED
Prototype: <pre>int dsp16_resampling_get_output_max_buffer_size (dsp_resampling_t *dsp_resampling);</pre>
Arguments: dsp_resampling The re-sampling context structure associated.
Return Value: NONE
Remarks: NONE

Description:	Returns the maximal length in sample of the output signal.
Module:	ADVANCED
Prototype:	<pre>int dsp16_resampling_get_output_current_buffer_size (dsp_resampling_t *dsp_resampling);</pre>
Arguments:	dsp_resampling The re-sampling context structure associated.
Return Value:	NONE
Remarks:	NONE

Description:	The re-sampling computation function.
Module:	ADVANCED
Prototype:	<pre>void dsp16_resampling_compute (dsp_resampling_t *dsp_resampling, dsp16_t *output, dsp16_t *input, int channel);</pre>
Arguments:	<p>dsp_resampling The re-sampling context structure associated.</p> <p>output A pointer to a 16-bit vector used to store output data. The length of this vector is defined by the output of the <i>dsp16_resampling_get_output_current_buffer_size</i> function.</p> <p>input A pointer to a 16-bit vector used as an input to the re-sampling process. It has to be of a length defined to the <i>dsp16_resampling_setup</i> function as for its sampling rate.</p> <p>channel The channel input argument (starting from 0 to <i>nb_channels - 1</i> referred in <i>dsp16_resampling_setup</i>)</p>
Return Value:	NONE
Remarks:	NONE



Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
Avr32@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN ATMEL'S TERMS AND CONDITIONS OF SALE LOCATED ON ATMEL'S WEB SITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© 2009 Atmel Corporation. All rights reserved. Atmel®, Atmel logo and combinations thereof, AVR®, AVR Studio® and others, are the registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.