ASN Filter Designer
DSP ANSI C v2.x SDK user's guide

July 2025

ASN25-DOC003, Rev. 2

1. Overview

This document provides an overview of how to use the DSP ANSI C v2.x SDK with ASN Filter Designer v5.4.0 and higher. A legacy document (ASN21-DOC012) is available for v1.x of the library, pertaining to ASN Filter Designer v5.3.5 and earlier versions.

The DSP ANSI C v2.x SDK provides significant improvements over v1.x, with strict data typing, improved memory management, and support for single-sample and multi-sample modes — specifically designed to help Arm developers implement reliable real-time embedded systems very easily.

Modes of operation:

1. **multi-sample (MS) mode**: highest speed – whereby 4 samples are computed in parallel. This method is approximately twice as fast as the single sample mode on most Arm processors with an HW FPU.
2. **single-sample (SS) mode**: good for real-time control applications, whereby single sample operation is required for minimising latency.

The SDK supports a variety of floating-point data types, including: `complex double`, `complex float`, `float` and `double` precision.

Fixed point arithmetic is currently not supported!

The content is as follows:

- Project folder structure generated from the ASN Filter Designer
- Filter cascade and non-linear functions
- Understanding main.c
- How to implement multiple filter cascades
- API description
- Setting up STM32 Cube IDE

Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

2. Project folder structure generated from the ASN Filter Designer

The ASN Filter Designer's automatic code generator produces the following files and folders in the project folder:

Name	Date modified	Type	Size
ASN_DSP	17-Jun-2025 15:40	File folder	
ASNFD.cbp	13-Jun-2025 9:59	CBP File	3 KB
ASNFDConfig.c	17-Jun-2025 15:40	C File	1 KB
ASNFDConfig.h	13-Jun-2025 9:59	VisualStudio.h.10.0	1 KB
main.c	17-Jun-2025 15:40	C File	3 KB

ASN_DSP - This folder contains all the framework dependencies required to compile the filter code. The ASN Filter Designer's code generation wizard will automatically select the correct datatype, and produce the necessary framework C files.

ASNFDConfig.c - This file contains API and filter structures of the designed filter(s).

ASNFDConfig.h – Associated header design file.

main.c - Contains example code to demonstrate the use of the SDK.

ASNFD.cbp –Code Blocks project file.

To use the filter code in your project, include the following folders and files in your project folder
`ASN_DSP, ASNFDConfig.c, ASNFDConfig.c`

Install Code Blocks or the MinGW compiler to quickly try the project.

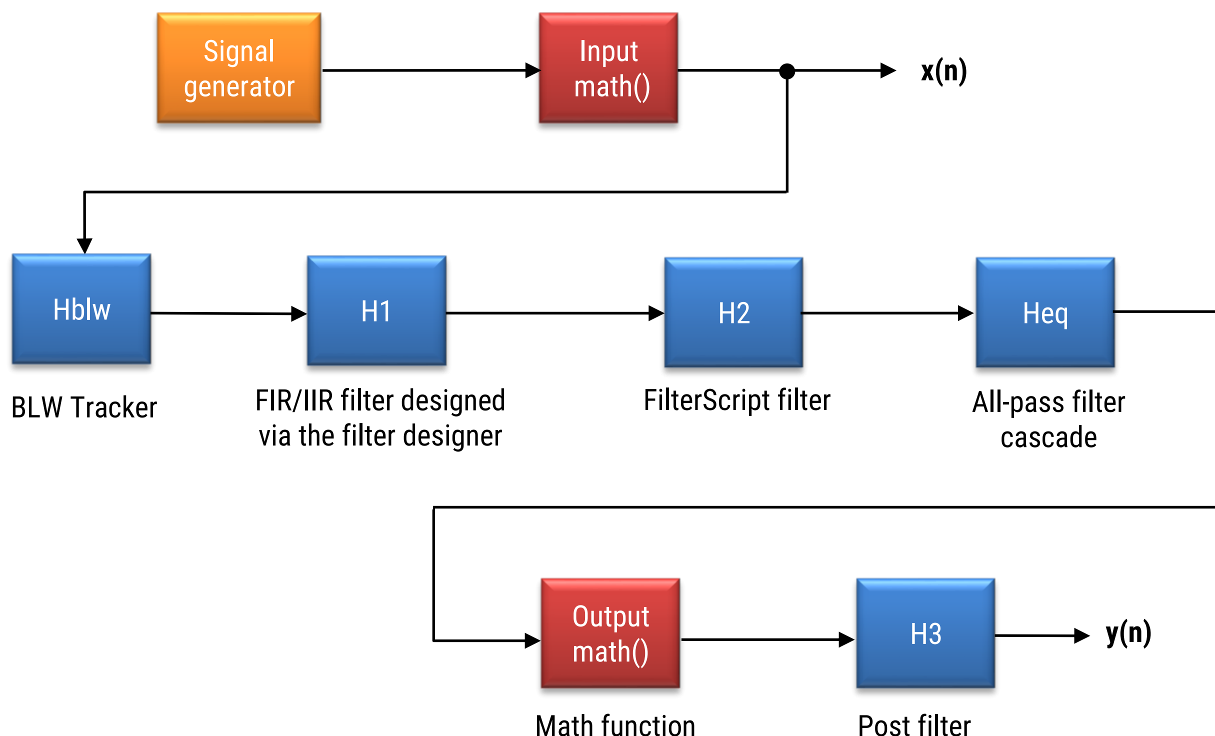
The framework SDK files are arranged in the ANSI C directory, where SS pertains to single-sample mode and MS pertains to multi-sample mode respectively.

Name	Date modified	Type
ASNFDFilterFW_C_MS_CMPLX_double	13-Jun-2025 10:03	File folder
ASNFDFilterFW_C_MS_CMPLX_float	13-Jun-2025 10:03	File folder
ASNFDFilterFW_C_MS_double	13-Jun-2025 10:03	File folder
ASNFDFilterFW_C_MS_float	13-Jun-2025 10:03	File folder
ASNFDFilterFW_C_SS_CMPLX_double	13-Jun-2025 10:03	File folder
ASNFDFilterFW_C_SS_CMPLX_float	13-Jun-2025 10:03	File folder
ASNFDFilterFW_C_SS_double	13-Jun-2025 10:03	File folder
ASNFDFilterFW_C_SS_float	13-Jun-2025 10:03	File folder
readme.txt	13-Jun-2025 9:59	Text Document

Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

3. Filter cascade and non-linear functions

In order to implement a complete application, the ASN Filter Designer uses a combination of filters and non-linear functions, as described below in the architecture diagram. All blocks can be enabled or disabled from within the ASN Filter Designer depending on the user's requirements.



Depending on the application, we can enable or disable Input and Output math functions. These functions are helpful for pre- or post-processing operations on the signal. The ASN Filter Designer supports the following math functions. You can select one of them according to your application's needs.

Function ()	Math operation	Description
`None`	-	Disable the function block.
`Abs`	$\|x\| = \sqrt{a^2 + b^2}$	Absolute.
`Ln`	$log_e\, x$	Natural logarithm.
`Angle`	$tan^{-1}\left(\frac{b}{a}\right)$	Compute the arctangent (phase in radians).
`RMS`	$\frac{\sqrt{a^2 + b^2}}{\sqrt{2}}$	Root mean square.
`Sqr`	x^2	Square.
`Sqrt`	$\sqrt{x}$	Square root.
`TKEO`	$y(n) = x^2(n-1) - x(n)(x-2)$	TKEO (Teager-Kaiser energy operator) algorithm.

Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

3.1. Hblw filter

The Hblw filter (BLW tracker) precedes the H1 primary filter and is primarily intended for BLW (baseline wander) suppression or tracking. The BLW tracker uses a linear phase Kolmogorov-Zurbenko filter cascade to implement either a tracking (LPF) or removal (HPF) filter cascade. This filter is ideal for designing a variety of low-frequency filters, such as biomedical BLW removal highpass filters for ECG and predictive maintenance applications, and lowpass DC tracking filters for DC loadcell applications.

3.2. H1 filter

The H1 filter is the primary filter. It supports the design of standard prototype methods, such as Butterworth, Chebyshev for IIR filters, and Parks-McClellan, Kaiser for FIR filters using the UI within the tool. The H1 filter may also be fine-tuned via the pole-zero editor and even programmed with an exotic filter via ASN FilterScript.

3.3. H2 filter

The H2 filter block implements a single-section IIR/FIR floating point filter. This filter is available for performing experiments with the P-Z editor or the ASN FilterScript scripting language. The FilterScript language is primarily intended as a sandbox concept, allowing for the design and experimentation of transfer functions with symbolic mathematical expressions.

The Hblw, H1 and H2 filters may be fully programmed using ASN FilterScript.

3.4. H3 filter

Unlike the Hblw, H1 and H2 filters, the H3 filter is always lowpass and is preceded by an optional mathematical function operation (i.e**., Abs, Angle, Ln, RMS, Sqr** or **Sqrt,** and **TKEO**).

H3 supports the following four types of filters:

Type	Transfer function	Gain at DC	Order
IIR	$H_3(z) = \dfrac{1 + 2z^{-1} + z^{-2}}{1 + 2\alpha z^{-1} + \alpha^2 z^{-2}}$	$\dfrac{1 + 2\alpha + \alpha^2}{4}$	2
Moving Average	$H_3(z) = 1 + z^{-1} + z^{-2} \ldots + z^{-M}$	$\dfrac{1}{(M+1)}$	1-200
Feed through	$H_3(z) = 1$	1	-
Median	*data window*	-	3-195

Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

4. Understanding main.c

ASN Filter Designer's code generator will automatically generate a C project for CodeBlocks using the requested quantisation (i.e. float, double or complex). The C code is ANSI C compliant and is not processor-specific, so it can be used agnostically on a variety of hardware platforms.

Careful design has been undertaken to ensure that the complexities of the filtering operations are hidden from the developer. Main.c will differ depending on which mode (single or multi) has been selected. You may also define multiple filter cascade instances, as discussed in section 5.

4.1. multi-sample mode

```
#define TEST_LENGTH_SAMPLES  2048 // must be a multiple of 4
#define TWO_PI 6.283185071795

double OutputValues[TEST_LENGTH_SAMPLES];
double InputValues[TEST_LENGTH_SAMPLES];

extern ASNFDDefinition_t ASNFD;  // add extra objects if required
```

```
initFilterCascade();

uint32_t n;
// setup test sinusoid input
for (n=0; n<TEST_LENGTH_SAMPLES; n++)
     InputValues[n]= sin(TWO_PI*10.0*n/ASNFD.Fs);

FilterCascadeData(&ASNFD,OutputValues,InputValues,TEST_LENGTH_SAMPLES);

for (n=0; n<TEST_LENGTH_SAMPLES; n++)
    printf("% 0.6f, % 0.6f \r\n", InputValues[n],OutputValues[n]);


ResetCascade(&ASNFD);
```

As seen, the automatically generated code initialises the filter cascade via the `ASNFD` object and the `initFilterCascade()` function. A test sinusoid is then defined (10Hz in this case) and assigned to the `InputValues` array. `FilterCascadeData()` is then called on this test data in order to perform the filtering operation. In order to achieve high implementation efficiency on Arm Cortex-M processors, 4 samples are computed in parallel. Therefore, `TEST_LENGTH_SAMPLES` **must be a multiple of 4 samples**, where a good default value is 128 or 256.

An optional reset command `ResetCascade(&ASNFD)` may be called to reset the filter cascade by setting the contents of the delaylines of all filters to zero.

Calling this function will not affect the filter coefficients.

Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

4.2. single-sample mode

```
initFilterCascade();    // intialise filter cascade

uint32_t n;

float Input,Output;
for (n=0; n<TEST_LENGTH_SAMPLES; n++)
{
    Input = sinf((float)(TWO_PI*10.0*n/ASNFD.Fs));
    Output = FilterCascadeData(&ASNFD,Input);
    printf("%3d, %0.6f, %0.6f \r\n",n,Input,Output);
}

ResetCascade(&ASNFD);
```

As seen, the automatically generated code initialises the filter cascade via the `ASNFD` object and the `initFilterCascade()` function. A test sinusoid is then defined (10Hz in this case) and assigned to the `Input` variable. `FilterCascadeData()` is then called on `Input` on a sample-by-sample basis in order to perform the filtering operation.

This mode is approximately twice as slow as the multi-sample mode version on most Arm Cortex-M processors with a HW FPU!

Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

5. Support for multiple filter cascades

If more than one filter cascade is required, **you must define an extra object** in both `main.c` and `ASNFilterConfig.c`. For example, to implement two filter cascades, we can define the objects as: `ASNFD` and `ASNFD2`

```
extern ASNFDDefinition_t ASNFD, ASNFD2;     (main.c)
       ASNFDDefinition_t ASNFD, ASNFD2;     (ASNFilterConfig.c)
```

You may then define each cascade's configuration in `ASNFilterConfig.c` – see below for an example.

```
#include "ASN_DSP/ASNFDFilter.h"
#include "ASNFDConfig.h"

ASNFDDefinition_t ASNFD,ASNFD2;

void initFilterCascade()
{
    InitASNFDDefinition(&ASNFD);

    ASNFD.Arithmetic = SinglePrecision;
    ASNFD.Architecture = FIR;
    ASNFD.Fs = 600.0000f;
    ASNFD.Structure = FIRDirectForm;

    // H1 filter
    float H1Numerator [] = { 0.014193f,  0.025876f,  0.039599f,  0.054409f,   // b[0:3]
                             0.069101f,  0.082355f,  0.092904f,  0.099699f,   // b[4:7]
                             0.102045f,  0.099699f,  0.092904f,  0.082355f,   // b[8:11]
                             0.069101f,  0.054409f,  0.039599f,  0.025876f,   // b[12:15]
                             0.014193f
                           };  // b[16]

    setH1Numerator(&ASNFD,(float*) H1Numerator,17); // set coefficients


    InitialiseCascade(&ASNFD); // instantiate cascade

    // 2nd filter IIR with same cut-off as FIR

    InitASNFDDefinition(&ASNFD2);

    ASNFD2.Arithmetic = SinglePrecision;
    ASNFD2.Architecture = IIR;
    ASNFD2.Fs = 600.0000f;
    ASNFD2.Structure = IIRTransposedDirectFormII;

    ASNFD2.Biquad = true;

    ASNFD2.H1NumBiquads = 2;
    // H1 filter SOS coefficients {b0, b1, b2, a1, a2}
    float H1SOS [10]= { 0.113146f, 0.113146f, 0.000000f, 0.773710f, 0.000000f,
                        0.014229f, 0.028459f, 0.014229f, 1.720012f,-0.776930f};
    setH1SOS(&ASNFD2, H1SOS);

    InitialiseCascade(&ASNFD2); // instantiate cascade
}
```

First cascade

Second cascade

Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

For single-sample mode, the code in main.c would be

```
initFilterCascade();   // intialise filter cascades

uint32_t n;
float Input,Output,Output2;

for (n=0; n<TEST_LENGTH_SAMPLES; n++)
{
    Input = sinf((float)(TWO_PI*2.0*n/ASNFD.Fs));
    Output = FilterCascadeData(&ASNFD,Input);
    Output2 = FilterCascadeData(&ASNFD2,Input);
    printf("% 0.6f, % 0.6f % 0.6f \r\n", Input,Output,Output2);
}

ResetCascade(&ASNFD);
ResetCascade(&ASNFD2);
```


Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

6. API description

A detailed overview of the ASN-DSP library's API for both the **single-sample** and **multi-sample** mode of operation is now given.

The content is as follows:

1. Initialise filter structure to default values
2. Release or deallocate the memory blocks
3. Initialise the filters and non-linear functions
4. Filtering using the filter cascade
5. Resetting the filter cascade
6. Setting the configuration of the Hblw filter
7. Setting the filter coefficients of the H1 filter
8. Setting the filter coefficients of the H2 filter
9. Setting the filter coefficients of the Heq filter
10. Setting the configuration of the H3 filter

6.1. Initialise filter structure to default values

Data type	API prototype
float	`void InitASNFDDefinition(ASNFDDefinition_t* filterptr);`
double	`void InitASNFDDefinition(ASNFDDefinition_t* filterptr);`
complex float	`void CMPLX_InitASNFDDefinition(CMPLX_ASNFDDefinition_t* filterptr);`
complex double	`void CMPLX_InitASNFDDefinition(CMPLX_ASNFDDefinition_t* filterptr);`

Description
Initialises the filter structure to its default values.

Argument
`filterptr`: pointer to filter structure.

Example
```
InitASNFDDefinition(&ASNFD);
```


Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

6.2. Release or deallocate the memory blocks

Data type	**API prototype**
`float`	`void DeinitASNFDDefinition(ASNFDDefinition_t* filterptr);`
`double`	`void DeinitASNFDDefinition(ASNFDDefinition_t* filterptr);`
`complex float`	`void CMPLX_DeinitASNFDDefinition(CMPLX_ASNFDDefinition_t* filterptr);`
`complex double`	`void CMPLX_DeinitASNFDDefinition(CMPLX_ASNFDDefinition_t* filterptr);`

Description

Release or deallocates the memory blocks utilised by the filter structure.

Argument

`filterptr`: Pointer to filter structure.

Example

```
DeinitASNFDDefinition(&ASNFD);
```

6.3. Initialise the filters and non-linear functions

Data type	**API prototype**
`float`	`void InitialiseCascade(ASNFDDefinition_t* filterptr);`
`double`	`void InitialiseCascade(ASNFDDefinition_t* filterptr);`
`complex float`	`void CMPLX_InitialiseCascade(CMPLX_ASNFDDefinition_t* filterptr);`
`complex double`	`void CMPLX_InitialiseCascade(CMPLX_ASNFDDefinition_t* filterptr);`

Description

Initialises the filters and non-linear functions according to the design specifications.

Argument

`filterptr`: Pointer to filter structure.

Example

```
InitialiseCascade(&ASNFD);
```


Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

6.4. Filtering using the filter cascade

The library supports two modes of operation: **single-sample** and **multi-sample** mode. As such, the `FilterCascadeData()` method handles input data either as an array of values (multi-sample mode) or as scalar value (single sample mode), as described in the following two sub-sections.

6.4.1. single-sample mode

Data type	API prototype
`float`	`float FilterCascadeData(ASNFDDefinition_t* filterptr, const float Input);`
`double`	`double FilterCascadeData(ASNFDDefinition_t* filterptr, const double Input);`
`complex float`	`complex float FilterCascadeData(ASNFDDefinition_t* filterptr, const complex float Input);`
`complex double`	`complex double FilterCascadeData(ASNFDDefinition_t* filterptr, const complex double Input);`

Description

Performs the filtering operation using the filter cascade on a sample-by-sample basis, and returns the result.

This method is approximately twice as slow as the multi-sample mode version on most Arm Cortex-M with HW FPU. As such, it is recommended to use the multi-sample mode version for performance, and this version for real-time applications where low latency is paramount.

Arguments

`filterptr`: Pointer to filter structure.
`Input`: scalar input sample value.

Example

```
Output=FilterCascadeData(&ASNFD,Input);
```


Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

6.4.2. multi-sample mode

Data type	API prototype
`float`	`void FilterCascadeData(ASNFDDefinition_t* filterptr, float* Output, const float* Input, const uint32_t FrameSize);`
`double`	`void FilterCascadeData(ASNFDDefinition_t* filterptr, double* Output, const double* Input, const uint32_t FrameSize);`
`complex float`	`void CMPLX_FilterCascadeData(ASNFDDefinition_t* filterptr, float* Output, const float* Input, const uint32_t FrameSize);`
`complex double`	`void CMPLX_FilterCascadeData(ASNFDDefinition_t* filterptr, double* Output, const double* Input, const uint32_t FrameSize,);`

Description
Performs the filtering operation using the filter cascade.

This method is advocated by virtue of its compatibility with data buffers in embedded applications, whereby a block of sampled data is written to one buffer by the DMA while the CPU simultaneously processes the previous block from the other buffer. This technique, known as ping-pong buffering, ensures seamless switching between buffers, enabling continuous data flow with minimal latency and improved processor efficiency—particularly in real-time signal processing and sensor data acquisition tasks.

In order to achieve high implementation efficiency on Arm Cortex-M processors, 4 samples are computed in parallel. Therefore, `FrameSize` **must be a multiple of 4 samples**, where a good default value is 128 or 256.

Arguments
`filterptr`: Pointer to filter structure.
`FrameSize`: Size of input buffer data.
`Output`: Pointer to an output buffer.
`Input`: Pointer to the input buffer.

Example
```
FilterCascadeData(&ASNFD,1024,Output,Input);
```


Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

6.5. Reset the filter cascade

Data type	API prototype
`float`	`void ResetCascade(ASNFDDefinition_t* filterptr);`
`double`	`void ResetCascade(ASNFDDefinition_t* filterptr);`
`complex float`	`void CMPLX_ResetCascade(CMPLX_ASNFDDefinition_t* filterptr);`
`complex double`	`void CMPLX_ResetCascade(CMPLX_ASNFDDefinition_t* filterptr);`

Description

Resets the filter cascade by setting the contents of the delaylines of all filters to zero.

The coefficients will remain unaffected. This does not need to be called in normal operation, but is included for the cases where you want to quickly reset the complete filter cascade.

Argument

`filterptr`: Pointer to filter structure.

Example

```
ResetCascade(&ASNFD);
```

6.6. Set configuration of Hblw filter

Description

Sets the configuration of the BLW Tracker.

Since the BLW Tracker is implemented as a cascade of moving average (MA) filters with unity coefficients, it is not necessary to explicitly define the filter coefficients. Therefore, only three parameters are required, as described below.

Arguments

`BLWTrackerMode`: Mode of operation: Removal (highpass), Tracking (lowpass).
`BLWTrackerL`: Length of MA filter.
`BLWTrackerR`: number of sections.

Examples

```
ASNFD.BLWTrackerMode = Removal; // Highpass filtering
ASNFD.BLWTrackerL = 357;
ASNFD.BLWTrackerR = 3;


ASNFD.BLWTrackerMode = Tracking; // Lowpass filtering
ASNFD.BLWTrackerL = 53;
ASNFD.BLWTrackerR = 2;
```


Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

6.7. Set filter coefficients of H1 filter

Data type	API prototype

float	void setH1Numerator(ASNFDDefinition_t* filterptr,float* ptr,int len);
	void setH1Denominator(ASNFDDefinition_t* filterptr,float* ptr,int len);
	void setH1SOS(ASNFDDefinition_t* filterptr, float* h1sos);

double	void setH1Numerator(ASNFDDefinition_t* filterptr, float* ptr, int len);
	void setH1Denominator(ASNFDDefinition_t* filterptr, float* ptr, int len);
	void setH1SOS(ASNFDDefinition_t* filterptr, float* h1sos);

complex float	void CMPLX_setH1Numerator(CMPLX_ASNFDDefinition_t* filterptr, complex float* ptr,int len);
	void CMPLX_setH1Denominator(CMPLX_ASNFDDefinition_t* filterptr, complex float* ptr,int len);
	void CMPLX_setH1SOS(CMPLX_ASNFDDefinition_t* filterptr,complex float* h1sos);

complex double	void CMPLX_setH1Numerator(CMPLX_ASNFDDefinition_t* filterptr, complex double* ptr, int len);
	void CMPLX_setH1Denominator(CMPLX_ASNFDDefinition_t* filterptr, complex double* ptr, int len);
	void CMPLX_setH1SOS(CMPLX_ASNFDDefinition_t* filterptr,complex double* h1sos);

Description

Sets the filter coefficients of the H1 filter.

Arguments

`filterptr`: pointer to filter structure.
`ptr, h1sos`: Pointer to the array.
`len`: length of the array.

Examples

```
float H1SOS[] = { 0.15919464484408, 0.15919464484408, 0.00000000000000,
0.68165454497010, 0.00000000000000},{ 0.04362563518631, 0.01333388307825,
0.04362563518631, 1.38973983422963,-0.49030168693325};

setH1SOS(&ASNFD, H1SOS);

float H1Numerator[] = { 0.98453370859690, -0.98453370859690,  0.98453370859690, -
0.98453370859690};

setH1Numerator(&ASNFD,(float*) H1Numerator,4);

float H1Denominator[] = {-1.00000000000000,  0.96906741719379, -0.92300230934793,
0.89445146398370};

setH1Denominator(&ASNFD,(float*) H1Denominator,4);
```


Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

6.8. Set filter coefficients of H2 filter

Data type	**API prototype**
`float`	`void setH2Numerator(ASNFDDefinition_t* filterptr, float* ptr,int len);` `void setH2Denominator(ASNFDDefinition_t* filterptr, float* ptr,int len);`
`double`	`void setH2Numerator(ASNFDDefinition_t* filterptr, double* ptr,int len);` `void setH2Denominator(ASNFDDefinition_t* filterptr, double* ptr,int len);`
`complex float`	`void CMPLX_setH2Numerator(CMPLX_ASNFDDefinition_t* filterptr, complex float* ptr,int len);` `void CMPLX_setH2Denominator(CMPLX_ASNFDDefinition_t* filterptr, complex float* ptr,int len);`
`complex double`	`void CMPLX_setH2Numerator(CMPLX_ASNFDDefinition_t* filterptr, complex double* ptr,int len);` `void CMPLX_setH2Denominator(CMPLX_ASNFDDefinition_t* filterptr, complex double* ptr,int len);`

Description

Sets the filter coefficients of the H2 filter.

Arguments

`filterptr`: Pointer to filter structure.
`ptr`: Pointer to the array.
`len`: Length of the array.

Example

```
float H2Numerator [] = {0.98453370859690, -0.98453370859690, 0.98453370859690, -
0.98453370859690};

setH2Numerator(&ASNFD,(float*) H2Numerator,4);

float H2Denominator [] = {-1.00000000000000, 0.96906741719379, -0.92300230934793,
0.89445146398370};

setH2Denominator(&ASNFD,(float*) H2Denominator,4);
```


Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

6.9. Set filter coefficients of Heq filter

Data type	API prototype
`float`	`void setHeqSOS(ASNFDDefinition_t* filterptr, float* heq);`
`double`	`void setHeqSOS(ASNFDDefinition_t* filterptr, double* heq);`
`complex float`	`void CMPLX_setHeqSOS(CMPLX_ASNFDDefinition_t* filterptr, complex float* heq);`
`complex double`	`void CMPLX_setHeqSOS(CMPLX_ASNFDDefinition_t* filterptr, complex double* heq);`

Description

Sets the filter coefficients of the all-pass equalisation filter (if enabled).

Arguments

`filterptr`: Pointer to filter structure.
`heq`: Pointer to the array.
`len`: Length of the array.

Example

```
float HeqSOS[]= {0.15919464484408, 0.15919464484408, 0.00000000000000,
0.68165454497010, 0.00000000000000},{ 0.04362563518631, 0.01333388307825,
0.04362563518631, 1.38973983422963,-0.49030168693325};

setHeqSOS(&ASNFD, HeqSOS);
```


Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

6.10. Set configuration of H3 filter

Data type	API prototype
float	`void setH3Numerator(ASNFDDefinition_t* filterptr, float* ptr, int len);` `void setH3Denominator(ASNFDDefinition_t* filterptr,  float* ptr, int len);`
double	`void setH3Numerator(ASNFDDefinition_t* filterptr, double* ptr,int len);` `void setH3Denominator(ASNFDDefinition_t* filterptr, double* ptr, int len);`
complex float	`void CMPLX_setH3Numerator(ASNFDDefinition_t* filterptr, complex float* ptr, int len);` `void CMPLX_setH3Denominator(ASNFDDefinition_t* filterptr, complex float* ptr, int len);`
complex double	`void CMPLX_setH3Numerator(CMPLX_ASNFDDefinition_t* filterptr, complex double* ptr, int len);` `void CMPLX_setH3Denominator(CMPLX_ASNFDDefinition_t* filterptr, complex double* ptr, int len);`

Description

Sets the filter coefficients of the H3 filter (if enabled).

Arguments

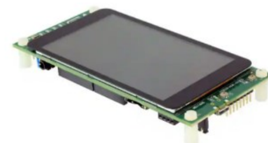
`filterptr`: Pointer to filter structure.
`ptr`: Pointer to the array.
`len`: Length of the array.

Example

```
float H3Numerator[] = { 0.98453370859690, -0.98453370859690,  0.98453370859690, -
0.98453370859690};

setH3Numerator(&ASNFD,(float*) H3Numerator,4);

float H3Denominator[] = {-1.00000000000000,  0.96906741719379, -0.92300230934793,
0.89445146398370};

setH3Denominator(&ASNFD,(float*) H3Denominator,4);
```


Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

7. STM32 Cube IDE and deploying to an STM32 Discovery kit



The STM32F469 Discovery kit is a very popular development platform for biomedical signal processing applications. The onboard Arm Cortex-M4 based microcontroller is a very capable processor, providing enough computational performance while maintaining low power and cost with floating-point operations.

The following steps should be undertaken for integrating the deployed C code library into an STM32CUBE-IDE project.

1. Generate the ANSI C filter project using ASN Filter Designer – see section 2.

   After generating the code, the working directory of the project should look like below.

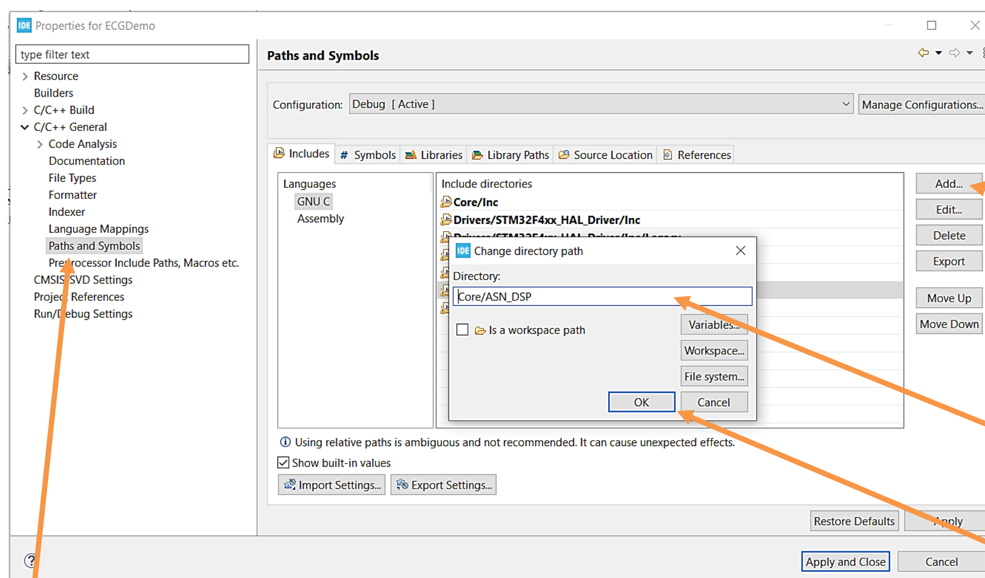
Name	Date modified
ASN_DSP	17-Jun-2025 15:40
ASNFD.cbp	13-Jun-2025 9:59
ASNFDConfig.c	17-Jun-2025 15:40
ASNFDConfig.h	13-Jun-2025 9:59
main.c	17-Jun-2025 15:40

2. Copy the `ASN_DSP` folder to your project folder.

3. Copy `ASNFDConfig.c` to the `Src` folder and `ASNFDConfig.h` file to the `Inc` folder.

4. Now open your project with STM32Cube-IDE, and go to **Properties → C/C++ General → Paths & Symbols**

5. Click on the "Add" button and enter the path of the ASN_DSP library folder.



Step 2: Click on "Add"

Step 3: Enter the path to the folder

Step 4: Click on "OK"

Step 1: Click on "paths and Symbols"

Copyright © 2025 Advanced Solutions Nederland. All rights reserved.

Document Revision Status

Rev.	Description	Date
1	Document released	20/06/2025
2	Added STM32 Cube integration steps	16/07/2025

Copyright © 2025 Advanced Solutions Nederland. All rights reserved.