ASN Filter Designer
DSP ANSI C v1.x SDK user's guide

June 2025

ASN21-DOC012, Rev. 5

1. Overview

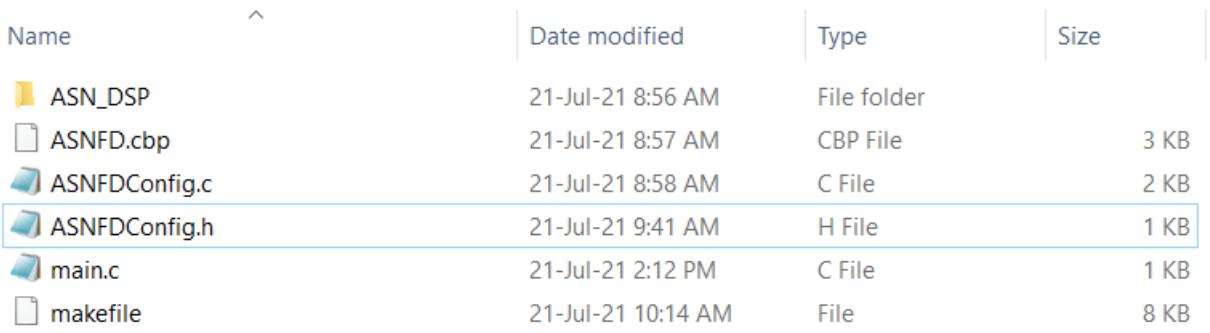
This document provides an overview of how to use the DSP ANSI C v1.x SDK with ASN Filter Designer up to version 5.3.5. Developers using ASN Filter Designer v5.4.0 or newer should refer to the newer DSP ANSI C v2.x SDK documentation respectively (ASN25-DOC003).

The content is as follows:

- Project folder structure generated from the ASN Filter Designer
- Modifying the makefile for your file system
- Filter cascade and non-linear functions and understanding main.c
- API description

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:



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.

makefile - Using this makefile, you can build and execute your filter code directly from the terminal. Steps involved in using the makefile are discussed in section 3.

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.

NB: - the makefile provided will only work under Windows.

3. Modifying the makefile for your file system

The following steps are required to modify the makefile for your computer:

a. Open the makefile in notepad and update the compiler's path in the `PATH_TO_COMPILER` variable in the makefile.

```
PATH_TO_COMPILER = "path to compiler"
```

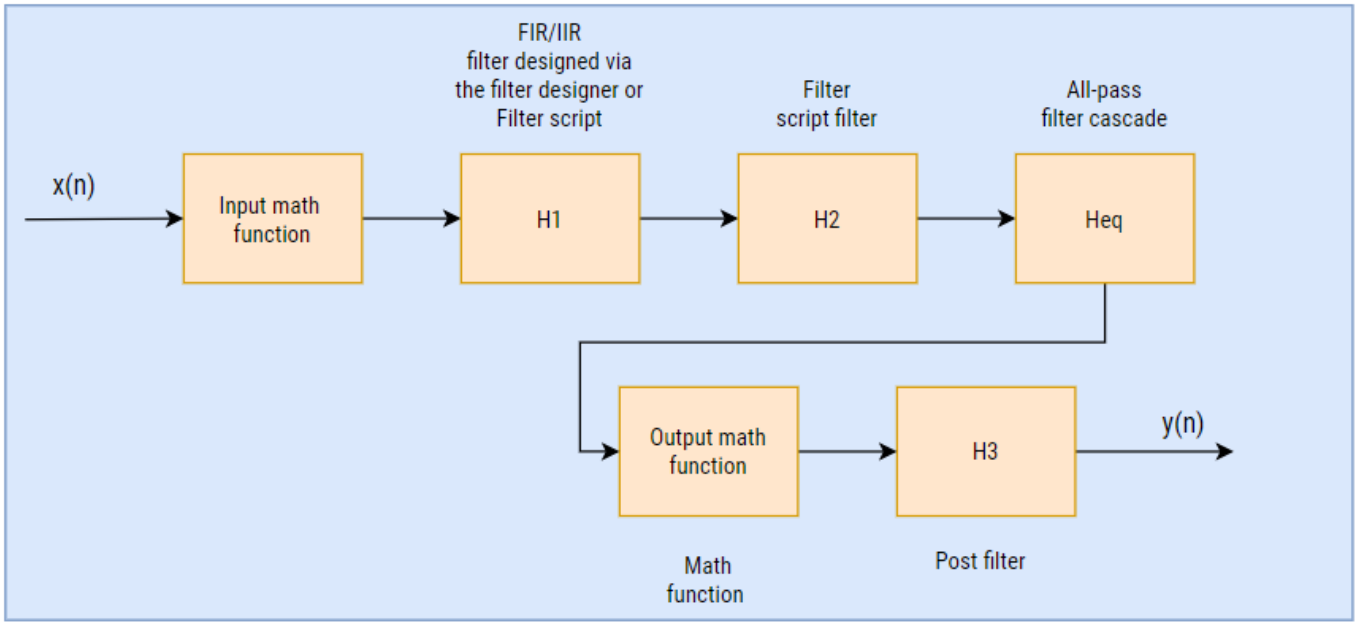
b. To build the project, open the terminal in the project folder type `make` then, you should see the project's build process.

c. To see the output of the filter with the example code, type the following command in the terminal `".\bin\Debug\ASNFD.exe"`

```
[redacted]\ASNFDFilterFW_C_float> .\bin\Debug\ASNFD.exe
lib name: ASN-DSP, version number : 1.0.3
Input: 0.000000, Output: 0.000000
Input: 0.248566, Output: 0.000001
Input: 0.481530, Output: 0.000008
```

4. 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.

The H1 (primary) filter(s) are designed via standard prototype methods, such as Butterworth, Chebyshev for IIR filters, and Parks-McClellan for FIR filters using the UI within the tool.

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. Both H1 and H2 filters may be fully programmed using FilterScript.

Unlike the 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	$H_3(z) = \frac{1 + 2z^{-1} + z^{-2}}{1 + 2\alpha z^{-1} + \alpha^2 z^{-2}}$	$\frac{1 + 2\alpha + \alpha^2}{4}$	2
Moving Average	$H_3(z) = 1 + z^{-1} + z^{-2} \ldots + z^{-M}$	$\frac{1}{(M+1)}$	1-200
Feed through	$H_3(z) = 1$	1	-
Median	*data window*	-	3-195

4.1. Understanding main.c

ASN Filter Designer's code generator will automatically generate a C project for CodeBlocks. 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. The `main.c` script is shown below:

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "ASNFDConfig.h"

#define TEST_LENGTH_SAMPLES  128
#define TWO_PI 6.283185071795

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

extern ASNFDDefinition_t ASNFD;

int main()
{
    libinfo info;
    getLibInfo(&info);
    printf("lib name: %s, version number : %d.%d.%d %s\r\n", info.libraryname, info.versionMajor, info.ve

    initFilter();

    uint32_t n;
    // setup test sinusoid input
    for (n=0; n<TEST_LENGTH_SAMPLES; n++)
        InputValues[n]= sin(TWO_PI*50.0*n/ASNFD.Fs);


    Filterdata(OutputValues,InputValues, TEST_LENGTH_SAMPLES);

    for (n=0; n<TEST_LENGTH_SAMPLES; n++)
        printf("InputValue[%.3d]: % 0.6f, OutputValue: % 0.6f\r\n", n,InputValues[n],OutputValues[n] );

    return 0;
}
```

As seen, the automatically generated code initialises the filter cascade via the `ASNFD` object and the `initFilter()` function. A test sinusoid is then defined (50Hz in this case) and assigned to the `InputValues` array. `Filterdata()` 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. NB. Calling this command will not affect the filter coefficients.

5. API description

A detailed overview of the ASN-DSP library's API is now given. A complete list can be found `ASNFDFilter.h/.c` files in the `ASN-DSP` directory.

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. filter using the filter cascade
5. Reset the filter cascade
6. Set filter coefficients of the H1 filter
7. Set filter coefficients of the H2 filter
8. Set filter coefficients of the Heq filter
9. Set coefficients of the H3 filter

5.1.1. 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

This API Initialises the filter structure to its default values.

Argument

`filterptr`: pointer to filter structure.

Example

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

5.1.2. 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 deallocate the memory blocks utilised by the filter structure.

Argument:

`filterptr`: Pointer to filter structure.

Example

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

5.1.3. 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);
```

5.1.4. Filter using the filter cascade

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.

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.

Argument:

`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);
```

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

Argument

`filterptr`: Pointer to filter structure.

Example

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

5.1.6. 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

These APIs are used to set the filter coefficients of the H1 filter.

Argument:

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

Example

```
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 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);
```

5.1.7. 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

These APIs are used to set the filter coefficients of the H2 filter.

Argument:

`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);
```

5.1.8. 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

These APIs are used to set the filter coefficients of the Heq (all-pass equalisation) filter if enabled.

Argument:

`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);
```

5.1.9. Set coefficients 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

These APIs are used to set the filter coefficients of the H3 filter (if enabled).

Argument:

`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);
```

Document Revision Status

Rev.	Description	Date
1	Document released	29/07/2021
2	API definition update	04/08/2021
3	API definition update	09/08/2021
4	Added section 4.1. and more explanation to FilterData()	13/01/2023
5	Added details about v2.x	05/06/2025