

Deploying the ECG Pan-Tompkins filter cascade on an STM32 Discovery kit using the ASN Filter Designer



Author: Dr. Sanjeev Sarpal

Application note (ASN-AN029)

March 2024 (Rev 3)

Synopsis

As governments slash their budgets for medical care in an attempt to minimise their deficits, the demand for low-cost high-tech home solutions has never been greater. Although biomedical monitoring devices have been around for decades, they are very expensive and require trained medical personnel. With the advent of miniature contactless sensor technology, consumers now have a chance to monitor their vital life signs in a very affordable way. However, the challenge for the manufacturers is how to successfully clean the biometric sensor data without destroying the delicate biometric features within the dataset and implement the total solution in an affordable way.

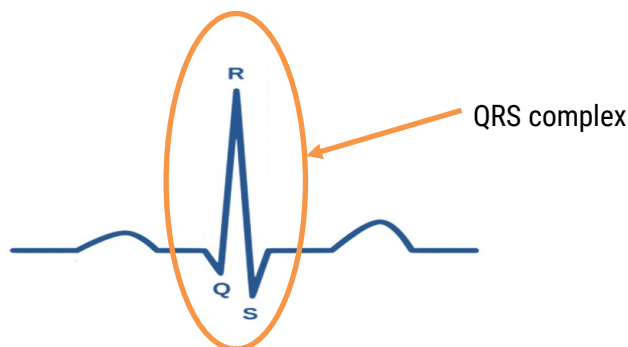
Common challenges

- Powerline and measurement noise that swamps the biometric data.
- Effects of BLW (baseline wander) and EMG caused by torso movement.
- Effects of digital filtering: broadens the QRS complex and may cause new anomalous artefacts to appear.
- Lack of knowledge in sensor signal processing and algorithms (FDA compliance).
- How to accurately track biometric features and classify heart arrhythmias.

Typical applications include: wearables, such as smartwatches; assisted home telemedicine applications and specialised medical devices.

1. Introduction

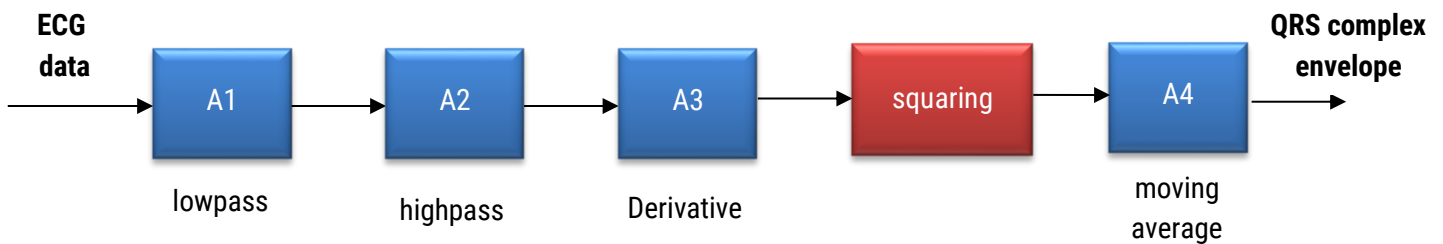
Although there are several methods for tracking the QRS complex in biomedical electrocardiographic signals (ECG) signals (e.g. Wavelets, the Hilbert transform, Savitzky-Golay filtering) the Pan-Tompkins algorithm is considered by many to be a standard textbook method. The QRS complex represents the ventricular depolarisation and the main spike visible in an ECG signal (see below). This feature makes it particularly suitable for measuring heart rate and tracking arrhythmias.



This application note demonstrates the design of just the Pan-Tompkins filter cascade using the ASN Filter Designer's FilterScript language, its interactive customisation and deployment to an embedded processor, such as an STM32. Readers looking for a complete description of the algorithm are referred to the [excellent references](#) on the subject.

1.1. The Pan-Tompkins algorithm

Although there are several sections to the complete Pan-Tompkins algorithm, a block diagram of the filter cascade is shown below:



The essence of the first stage of the filter cascade centres around building a bandpass filter with the A1 and A2 filters. A filter bandwidth of approximately 10Hz (e.g. 5-15Hz cut-off) is suggested to maximize the QRS contribution and reduce muscle noise, baseline wander, powerline interference and the P wave/T wave frequency content. The resulting signal is then passed through a 5-point derivative filter (A3) in order to provide rate information about the QRS complex. The final steps involve squaring the signal (`abs()` may also be used), and then integrating the resulting signal using the A4 filter. This integration process provides an envelope of the ECG complex and is fundamental for feature extraction.

Although several implementations exist in the biomedical community, the following filter definitions are used for an ECG signal sampled at 200 Hz.

Building block	Equation	Description
A1	$\frac{(1 - z^{-6})^2}{(1 - z^{-1})^2}$	A lowpass filter.
A2	$\frac{(-1/32 + z^{-16} - z^{-17} - z^{-32}/32)}{(1 - z^{-1})}$	A highpass filter.
A3	$(2 + z^{-1} - z^{-3} - 2z^{-4})/4$	A 5-point derivative filter (modified from the original definition for real-time implementation)
Squaring	<code>sqr()</code>	Sqr() or Abs() function.
A4	$1 + z^{-1} + z^{-2} + \dots + z^{-M}$	An Mth order moving average filter (integration)

Analysing the equations, notice that in many cases the filter coefficients are unity, leading to very computationally simple implementation. However, it is important to note that these equations are designed for ECG biomedical data sampled at 200Hz.

1.1.1. Practical implementation issues

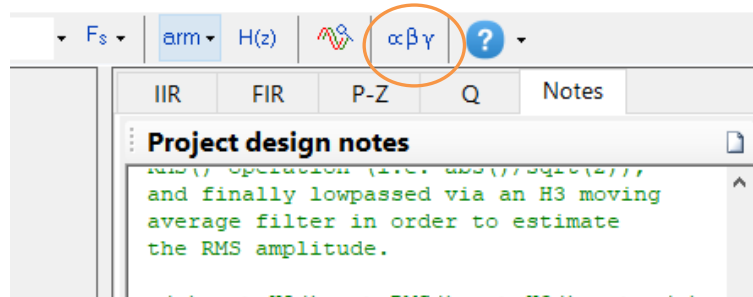
The reader may have noted that Eqns. A1 and A2 have poles on the unit circle. Although when cascaded these poles should be cancelled by zeros at the same location there will always be a degree of error depending on the quantisation used. As such, these errors will lead to longer settling times and a DC offset in the overall result, as any integration errors will accumulate. This problem may be easily overcome by nudging the poles slightly off the unit circle by a small *forgetting factor*, β such that $\alpha = 1 - \beta$

Building block	Equation
A1	$\frac{(1 - z^{-6})^2}{(1 - \alpha z^{-1})^2}$
A2	$\frac{(-1/32 + z^{-16} - z^{-17} - z^{-32}/32)}{(1 - \alpha z^{-1})}$

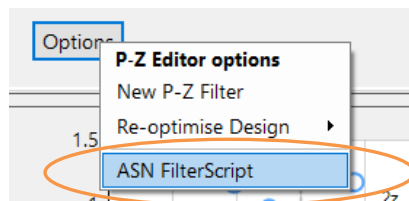
The exact value of α is best found by experimentation, but a good initial starting point is $\alpha = 0.995$. These corrected equations will now ensure BIBO filter stability and speed up the cascade's dynamic time domain performance and eliminate any DC offsets.

2. Symbolic math scripting language

As seen, there are three filters and one mathematical operation. Implementing the complete filter cascade can be simply achieved in the ASN Filter Designer with the symbolic filter scripting language.



Alternatively in the P-Z tab,



The scripting language supports over 82 scientific commands and provides designers with a familiar and powerful programming language, while at the same time allowing them to implement complex symbolic mathematical expressions for their filtering applications. The scripting language offers the unique and powerful ability to modify parameters on the fly with the so-called interface variables, allowing for real-time updates of the resulting frequency response. This has the advantage of allowing the designer to see how the coefficients of the symbolic transfer function expression affect the frequency response and the filter's time-domain dynamic performance.

Please refer to the ASN Filter Designer FilterScript [reference guide](#) for more information.

2.1. ASN FilterScript

The `digitaltf()` function builds a digital filter object based on the numerator and denominator specifications.

```
lpfnum={1,zeros(5),-2,zeros(5),1};  
lpfden={1,-2*alpha,sqr(alpha),zeros(10)};  
  
A1=digitaltf(lpfnum,lpfden,1,"void");
```

Repeating this operation for all three filters, we obtain three digital filter objects, i.e. A1, A2 and A3.

As ASN FilterScript only supports a single H2 filter object, these filters need to be split up between the H1 primary filter and the secondary H2 filter in FilterScript. Therefore, by assigning the A1 filter to the H1 filter and merging the A2 and A3 filters into a single filter for the H2 filter, we can implement the complete cascade.

The `augment()` function merges the A2 and A3 filters¹ in order to produce a new filter.

```
/// augment highpass and derivative filters  
Hd=augment(A2,A3,"void");
```

The A4 filter is assigned to [H3 post filter](#) respectively that performs the integration.

Finally, in order to get the cascade gain right, a post scaling operation is performed to set the gain to 0dB at 10Hz. You may of course adjust this to suit your specific requirements.

```
/// Correct cascade gain: set 10Hz to 0dB  
Fc=10; //10Hz  
G=computegain(Hd,Fc);  
G=G*computegain(A1,Fc);  
Gain=Gain/(G);
```

The complete FilterScript code is shown overleaf.

¹ `augment()` calls the `conv()` function that is the same as performing an algebraic multiplication of the numerator and denominator filter polynomials.

```

/// Pan-Tompkins algorithm for ECG BPM detection
//
// This script uses both the H1 and H2 filters
// to implement a filter cascade as described by
// Pan-Tompkins: https://en.wikipedia.org/wiki/Pan-Tompkins\_algorithm
//
/// IMPORTANT: Fs = 200Hz
//
// Date: 11 March 2024
// Copyright 2024 Advanced Solutions Nederland BV.

/// forgetting factor: alpha=1-beta
interface alpha = {0.992,0.998,0.001,0.995};

Main()

/// Design lowpass
//
//      (1-z^-6)^2
// A1(z) = -----
//      (1-alpha*z^-1)^2
//
lpfnum={1,zeros(5),-2,zeros(5),1};
lpfden={1,-2*alpha,sqr(alpha),zeros(10)};
A1=digitaltf(lpfnum,lpfden,1,"void");

/// Design highpass
//
//      (-1/32 + z^-16 -z^-17 + z^-32/32)
// A2(z) = -----
//      (1-alpha*z^-1)
//
hpfnum={-1/32,zeros(15),1,-1,zeros(14),1/32};
hpfden={1,-alpha,zeros(31)};
A2=digitaltf(hpfnum,hpfden,1/32,"void");

/// Design derivative filter
// A3(z) = (2 + z^-1 - z^-3 - 2z^-4)/4
//
deriv={2,1,0,-1,-2};
A3=digitaltf(deriv,1,1/4,"symbolic");

/// augment highpass and derivative filters
Hd=augment(A2,A3,"void");

/// place augmented filter into H2
Num = getnum(Hd);
Den = getden(Hd);
Gain = getgain(Hd);

/// place lowpass filter into H1 filter
H1Num = getnum(A1);
H1Den = getden(A1);
H1Gain = getgain(A1);

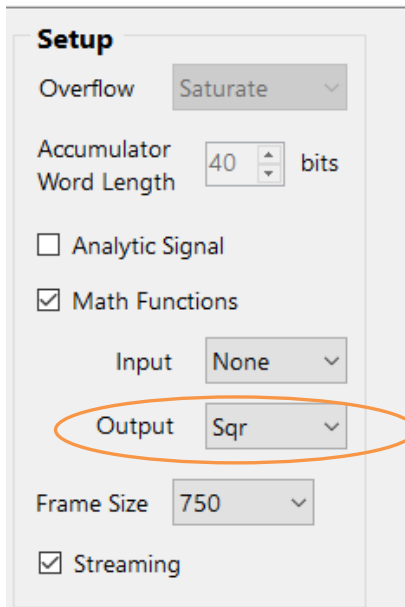
/// Correct cascade gain: set 10Hz to 0dB
Fc=10; //10Hz
G=computegain(Hd,Fc);
G=G*computegain(A1,Fc);
Gain=Gain/(G);

```

3. Other IP blocks

As discussed in section 1.1, the non-linear function and integration are implemented in the Math Functions block and H3 filter. Details of these IP blocks are discussed in the following subsections.

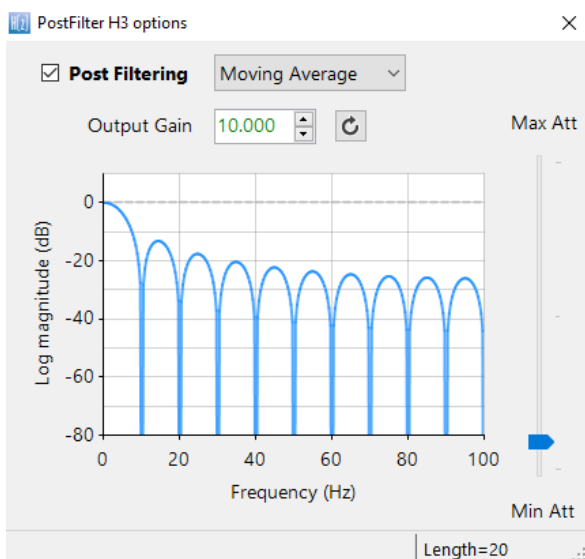
3.1. Mathematic functions




The squaring operation can be implemented by selecting the `Sqr()` function via the **Math Functions** → **Output** in the signal analyser.

Depending on the dataset, you may also use the `abs()` function.

3.2. Integration via the H3 Filter



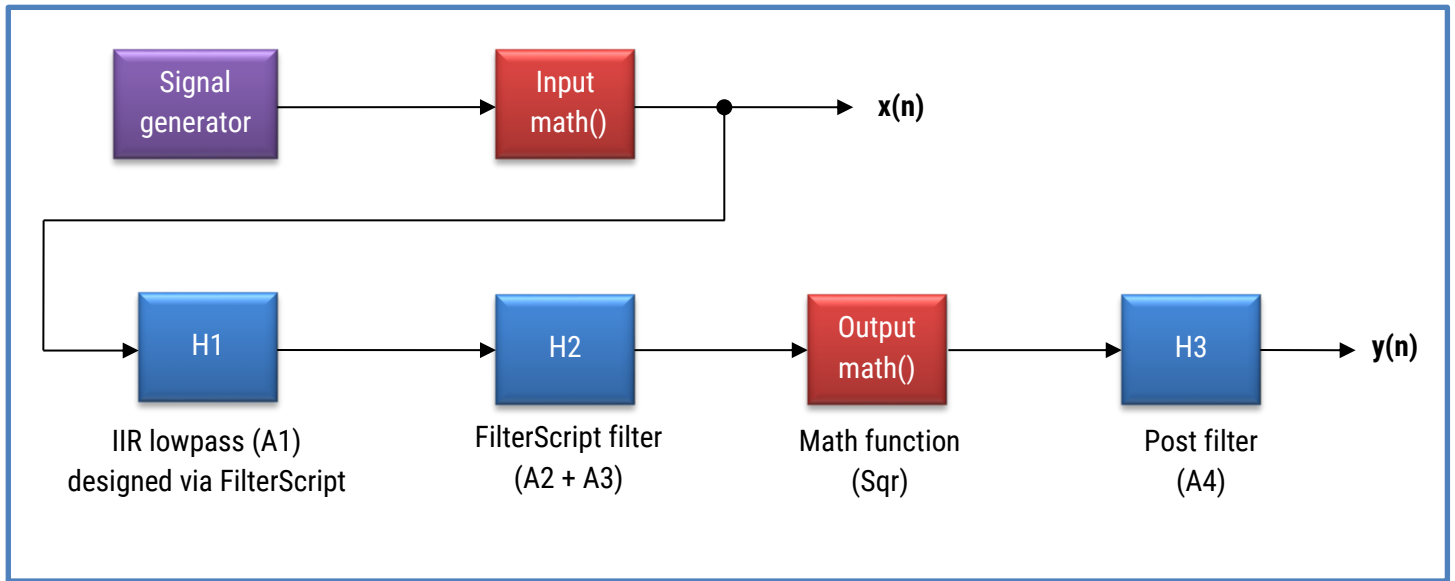
The ASN Filter designer's signal analyser implements an extra post filter, H3. Unlike the H1 and H2 filters, the H3 filter is *always lowpass* and is preceded by an optional mathematical function operation (see above).

For the application considered herein, the H3 the integration block is implemented as a moving average filter in the signal analyser's H3 post filter, .

The ASN filter designer's signal analyser complete filtering chain is shown overleaf together with the signal generator and the input/output math function blocks.

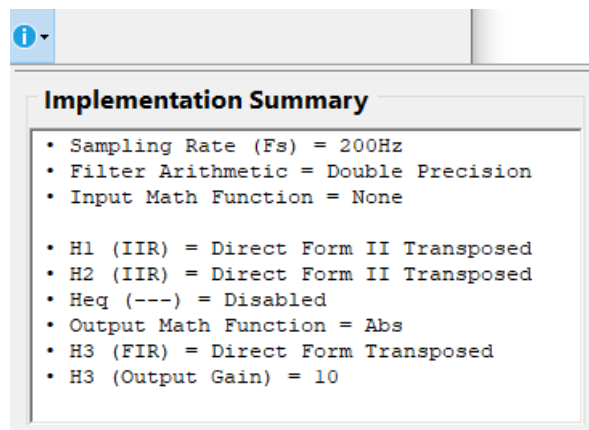
3.3. Implementation summary and frequency response

An illustration of the signal processing chain within the tool is shown below. As seen, the A1, A2, A3 and A4 filters are spread out over the H1, H2 and H3 filters respectively.



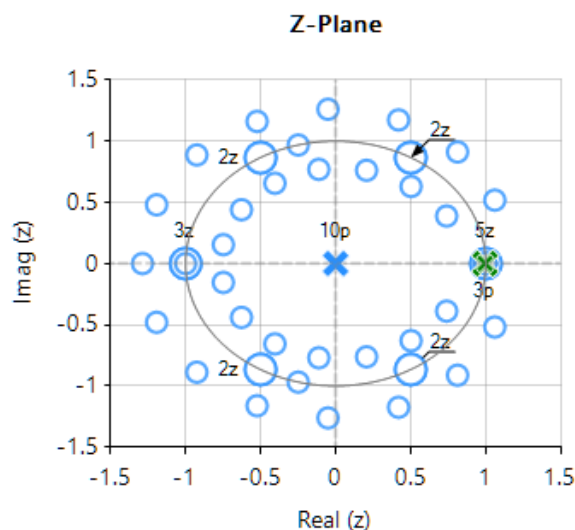
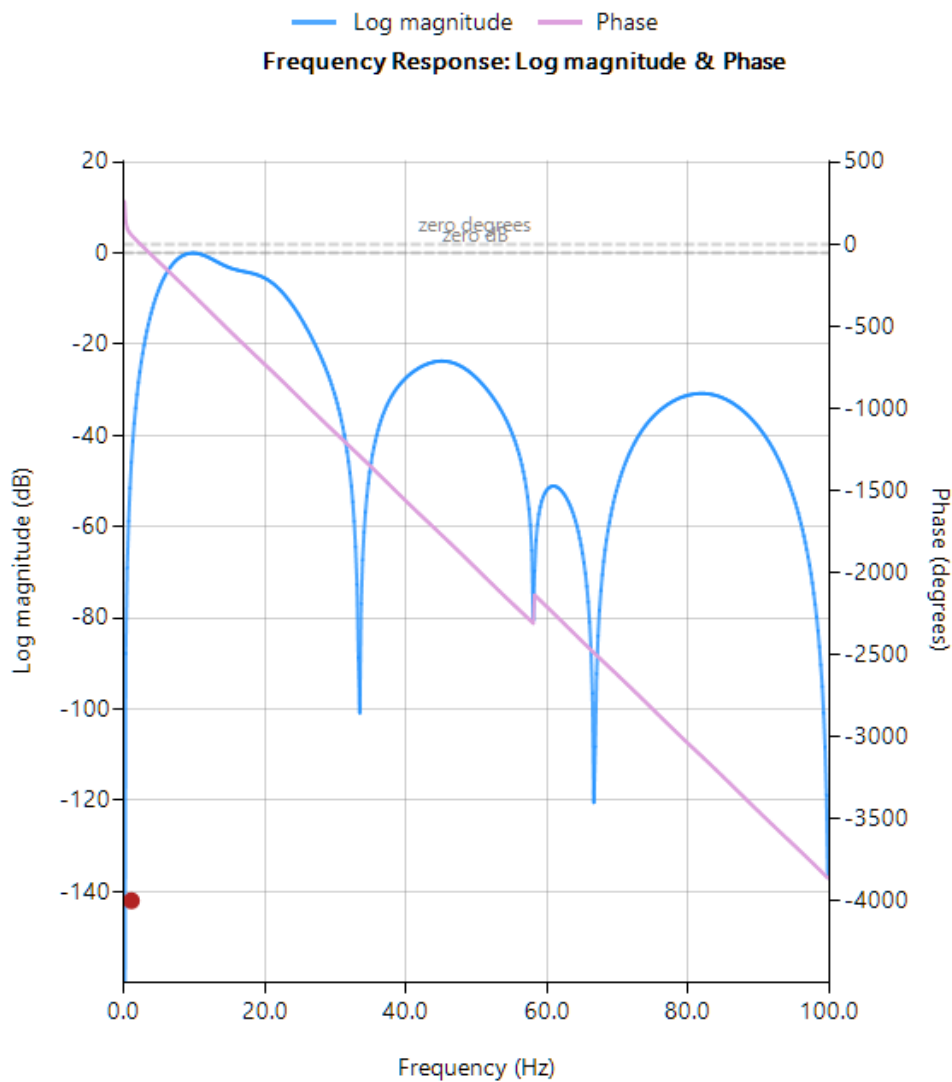
The Input math () block is disabled for the application considered herein.

An implementation summary can be found in the signal analyser UI, via the information menu:



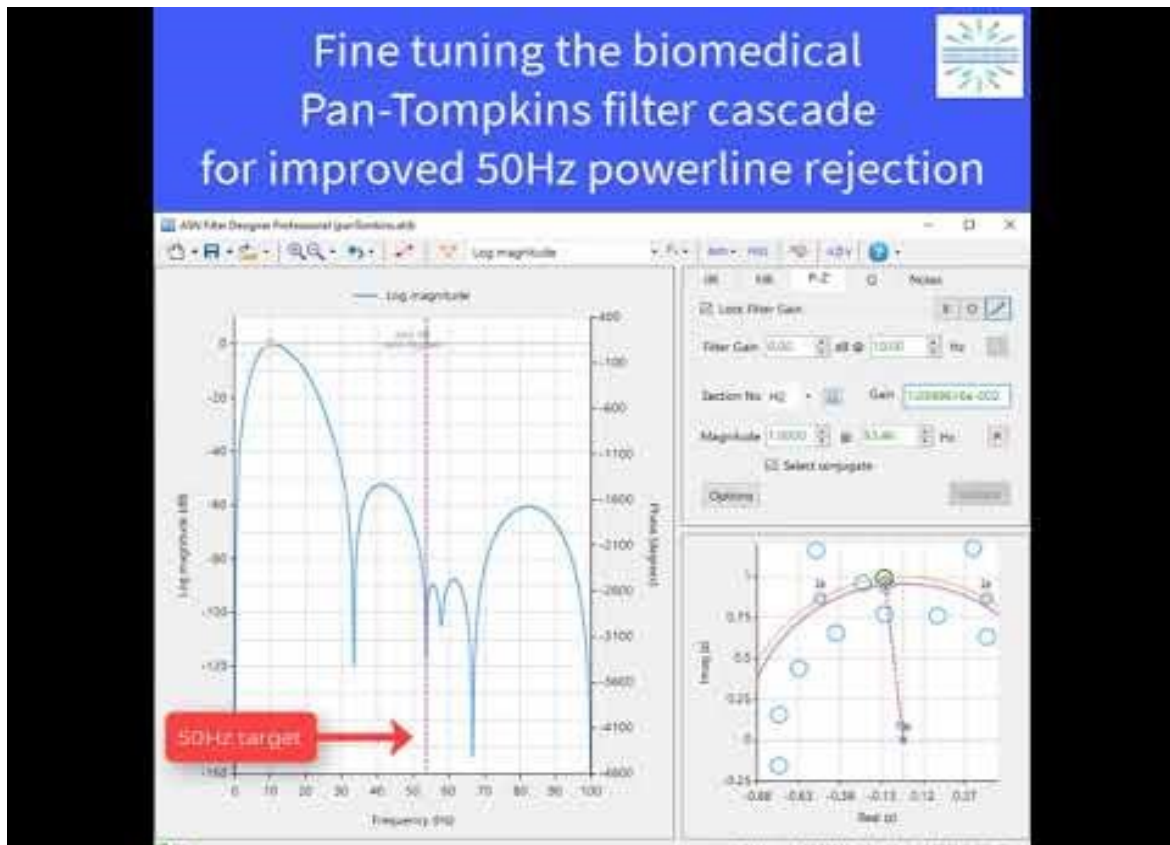
4. P-Z chart, cascade frequency response and output

Upon running the FilterScript code, we obtain the following P-Z chart and frequency response. As seen, all three filters have been cascaded/combined correctly.



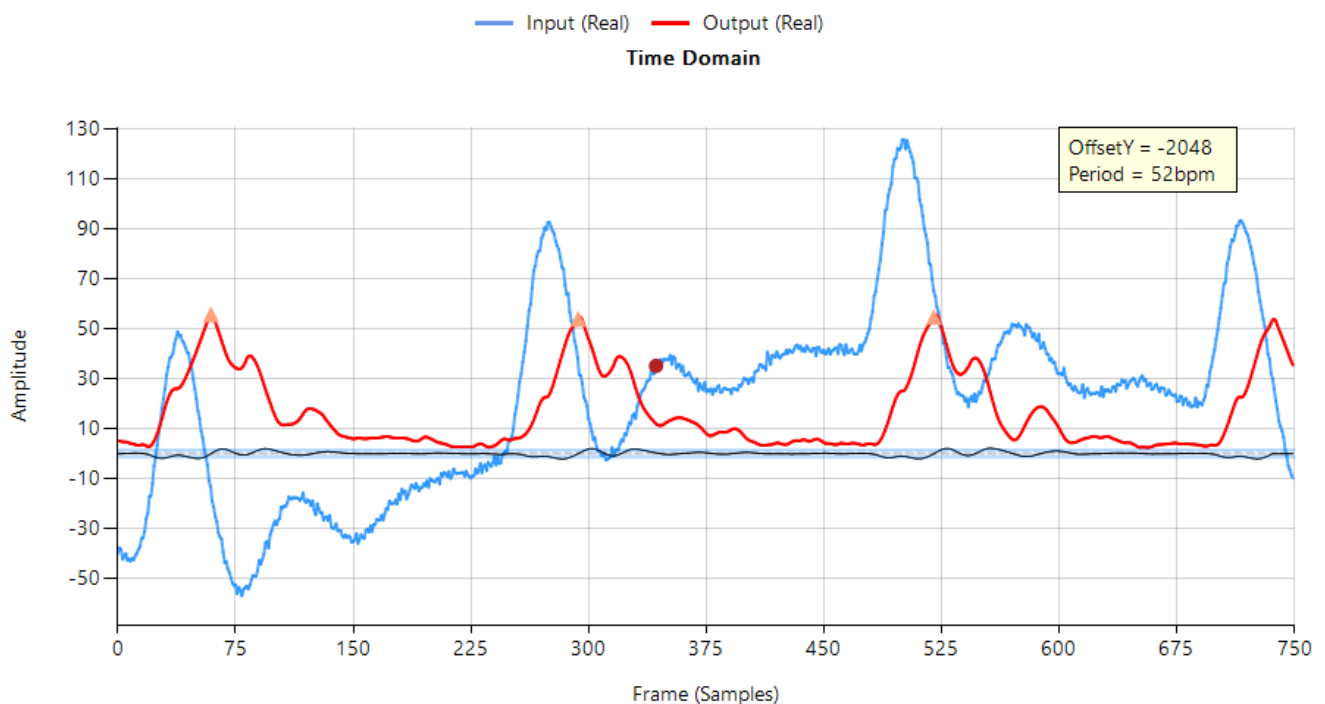
4.1. Better 50/60Hz powerline rejection

You may use the P-Z chart editor in order to improve the filter cascade's 50/60Hz powerline rejection performance. The video below demonstrates the procedure.



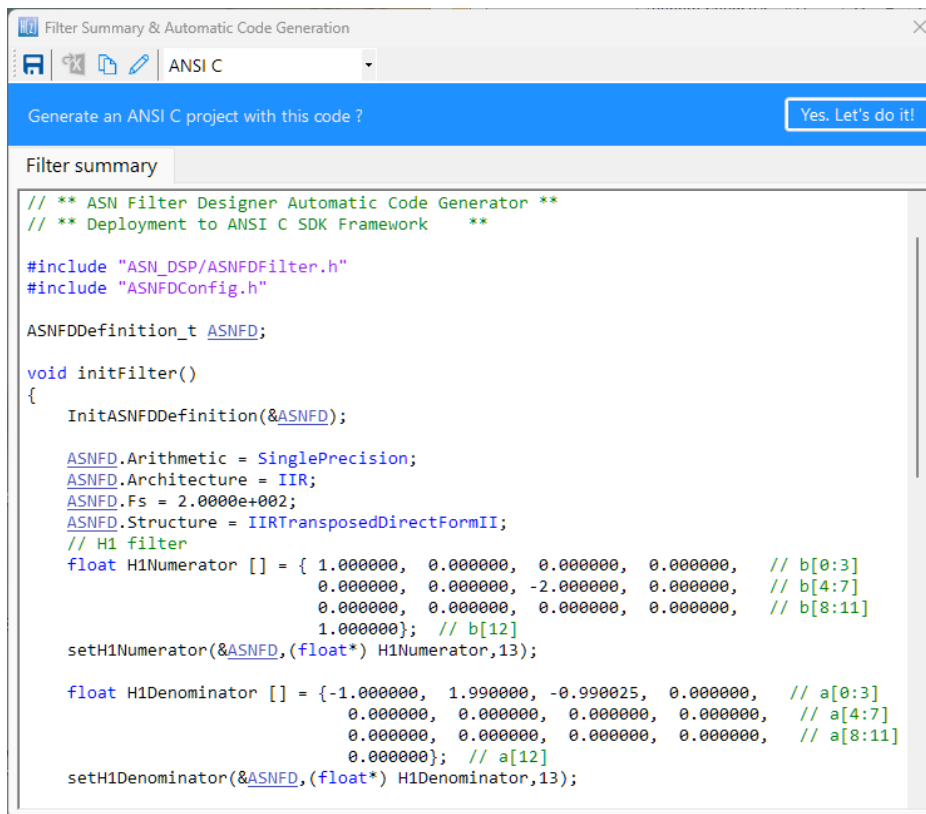
As seen, within a few minutes you can significantly improve the frequency response of the standard filter cascade by just nudging the position of the zeros using the mouse.

4.2. Output



5. Deploying to an embedded platform

ASN Filter Designer's ANSI C SDK framework provides developers with a comprehensive ANSI C code base that can be used to deploy developed filtering applications to any C embedded platform. This agnostic feature, allows product developers to quickly integrate digital filtering functionality into their existing designs with the minimum amount of effort. This allows developers to directly deploy their filtering application from within the tool to any **Arm, Arduino, ESP32, Beagle Bone platform** for direct use in their application.



```
// ** ASN Filter Designer Automatic Code Generator **
// ** Deployment to ANSI C SDK Framework **

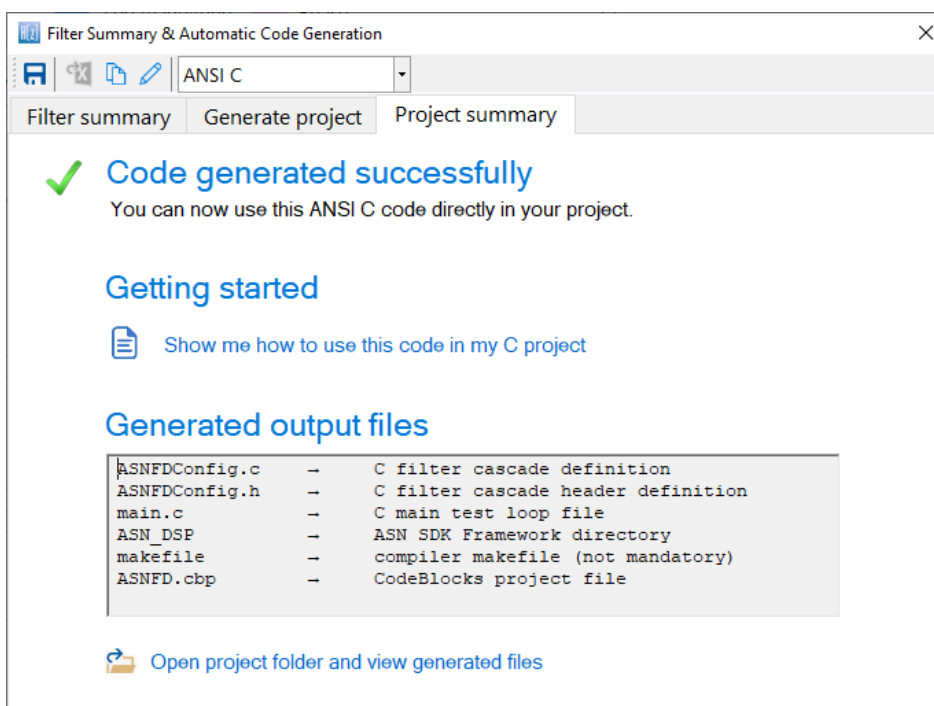
#include "ASN_DSP/ASNFDFilter.h"
#include "ASNFDFilterConfig.h"

ASNFDDefinition_t ASNFD;

void initFilter()
{
    InitASNFDDefinition(&ASNFD);

    ASNFD.Arithmetic = SinglePrecision;
    ASNFD.Architecture = IIR;
    ASNFD.Fs = 2.00000e+002;
    ASNFD.Structure = IIRTransposedDirectFormII;
    // H1 filter
    float H1Numerator [] = { 1.000000, 0.000000, 0.000000, 0.000000, // b[0:3]
                            0.000000, 0.000000, -2.000000, 0.000000, // b[4:7]
                            0.000000, 0.000000, 0.000000, 0.000000, // b[8:11]
                            1.000000}; // b[12]
    setH1Numerator(&ASNFD,(float*) H1Numerator,13);

    float H1Denominator [] = {-1.000000, 1.990000, -0.990025, 0.000000, // a[0:3]
                              0.000000, 0.000000, 0.000000, 0.000000, // a[4:7]
                              0.000000, 0.000000, 0.000000, 0.000000, // a[8:11]
                              0.000000}; // a[12]
    setH1Denominator(&ASNFD,(float*) H1Denominator,13);
}
```



Code generated successfully
You can now use this ANSI C code directly in your project.

Getting started
[Show me how to use this code in my C project](#)

Generated output files

ASNFDFilterConfig.c	-	C filter cascade definition
ASNFDFilterConfig.h	-	C filter cascade header definition
main.c	-	C main test loop file
ASN_DSP	-	ASN SDK Framework directory
makefile	-	compiler makefile (not mandatory)
ASNFDFilter.cbp	-	CodeBlocks project file

[Open project folder and view generated files](#)

Please refer to the [ANSI C SDK user guide](#) for step-by-step instructions on how to use the generated code in your project.

5.1. 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.

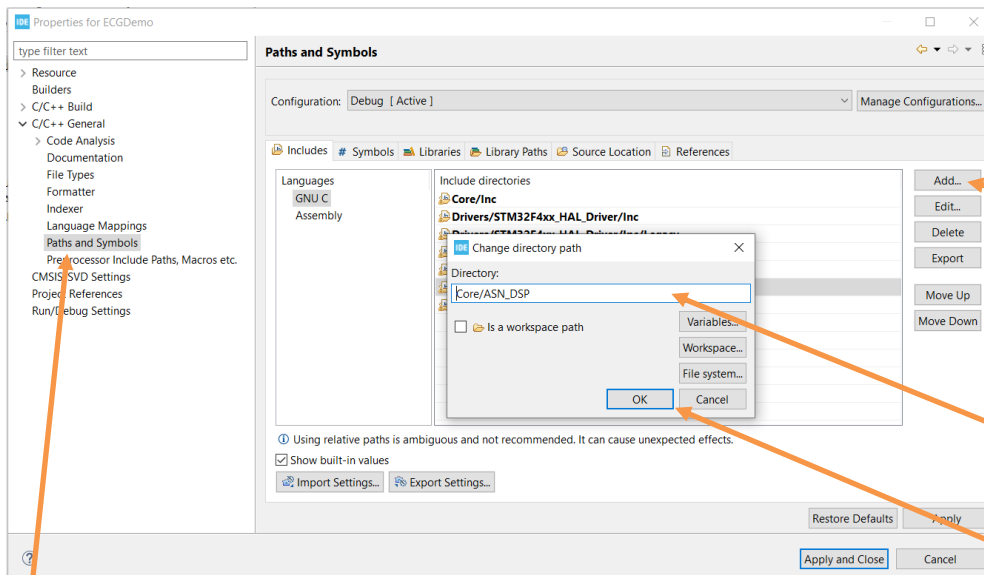


1. Generate the ANSI C filter project using ASN Filter Designer – see section 5.

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

Name	Date modified
ASN_DSP	01-Sep-21 6:31 PM
ASNFD.cbp	01-Sep-21 2:16 PM
ASNFDConfig.c	01-Sep-21 6:31 PM
ASNFDConfig.h	01-Sep-21 2:17 PM
main.c	01-Sep-21 6:33 PM
makefile	01-Sep-21 2:16 PM

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”

5.2. Benchmarks

Running the algorithm on the STM32 Discovery kit, we obtained the following benchmarks with the onboard Arm Cortex-M4F² processor.

Test	Clock count	Time (ms) @ 180MHz
1024 samples	1899735	10.55
32 samples	59553	0.33

Analysing the benchmark results, it can be seen that even for 5 seconds worth of data (1024 samples), the filtering operation is completed in about 11ms when using a 180MHz clock. Also, for 32 samples or 160ms worth of measurement data ($F_s = 200\text{Hz}$), the filtering operation is completed in $330\mu\text{s}$, making the implementation suitable for real-time QRS detection.

6. Product support and further reading

ASN Filter Designer

1. ASN Filter Designer [product home page](#)
2. ASN Technical support: support@advsolned.com
3. [ANSI C SDK user guide](#) for step-by-step instructions on how to use the generated code in your project.

Biomedical QRS detection and BPM tracking

1. A detailed description of the Pan-Tompkins algorithm on [Wiki](#)
2. Pan-Tompkins's [original paper](#) from 1985
3. Good overview of [QRS detection methods](#) and challenges
4. Robust BPM measurement using [Savitzky-Golay filtering](#) (ASN Filter Designer reference design)

Document Revision Status

Rev.	Description	Date
1	Document reviewed and released.	26/10/2021
2	Added more detail in the introduction and STM32 details	19/06/2023
3	Added section 1.1.1 and updated code	11/03/2024

² F suffix signifies that the device has an FPU (floating point unit).