**Programming DSP processors (31561) Final project**

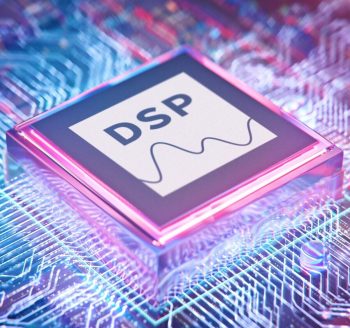
Objective: heartbeat detection and processing

From:

Omri Pony I.D. number: 316352210

barak Shinfeld I.D. number: 207478546

To: MR Itzhak Kroin, EE Faculty



1. **Introduction to the project: heartbeat – biological and engineering aspect**

The heart is a hollow organ made of muscle. The heart and the blood vessels that surround it are part of man's cardiovascular system.

The main objective of the heart is to pump blood through the [blood vessels](https://www.msdmanuals.com/home/quick-facts-heart-and-blood-vessel-disorders/biology-of-the-heart-and-blood-vessels/biology-of-the-blood-vessels).

The blood carries oxygen and nutrients to all parts of the body – therefore , the heart is one of the key organs in our body and its functionality is critical to our existence and health.

Each time the heart beats-blood is pumped out of the heart and into the body to supply oxygen to working muscles or to the lungs for re-oxygenation.

**Heart rate refers to the number of times the heart beats per minute** , and is directly related to the workload being placed on the heart. When the body is in a resting state (i.e. lying down in a quiet area for at least five minutes), resting heart rate is measured. **A normal resting heart rate ranges from 60-100 beats per minute (bpm)**. Resting rates higher than 100 bpm suggest that the heart is working too hard to circulate blood, and thus may indicate a serious problem that should be monitored by a physician. Resting rates lower than 60 bpm occur more often with endurance-trained athletes whose bodies are more efficient at utilizing oxygen from the blood.

The heart rate is controlled by the two branches of the autonomic nervous system: The **sympathetic nervous system** (SNS) and the **parasympathetic nervous system** (PNS). The sympathetic nervous system (SNS) releases the hormones (catecholamines - epinephrine and norepinephrine) to accelerate the heart rate parasympathetic nervous system (PNS) releases the hormone acetylcholine to slow the heart rate.

1. **System requirements:**

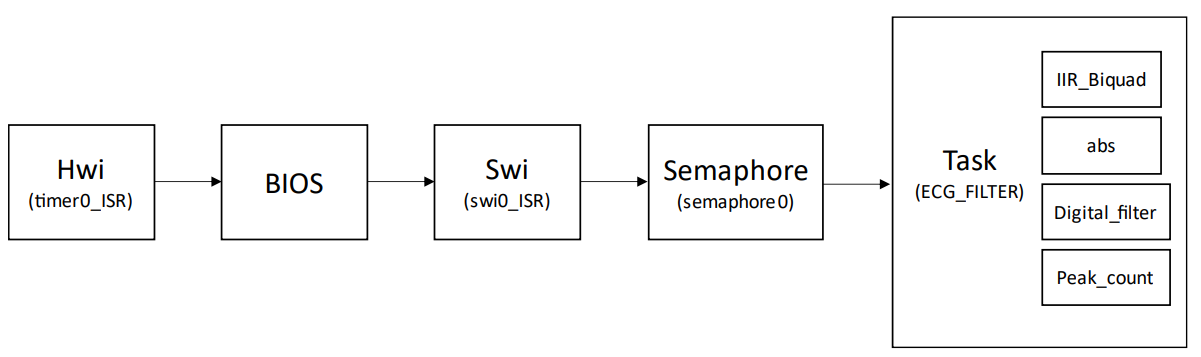
* Implementation of the given heartbeat pulse signal at a sampling frequency of .
* Use DSP's internal Timer that samples the heartbeat pulse signal at the set sampling rate continuously and cyclically.
* Use MATLAB software to design suitable filters to filter out unwanted frequencies and improve pulse detection.

Based on the instructor's recommendation, IIR filters were chosen.

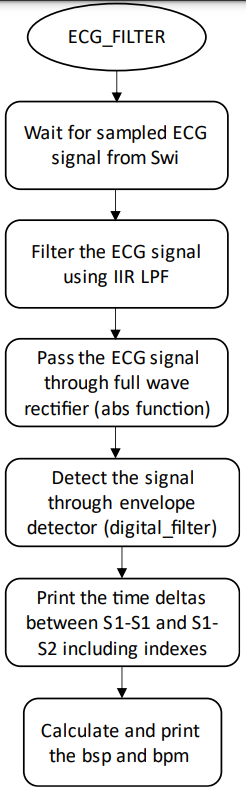
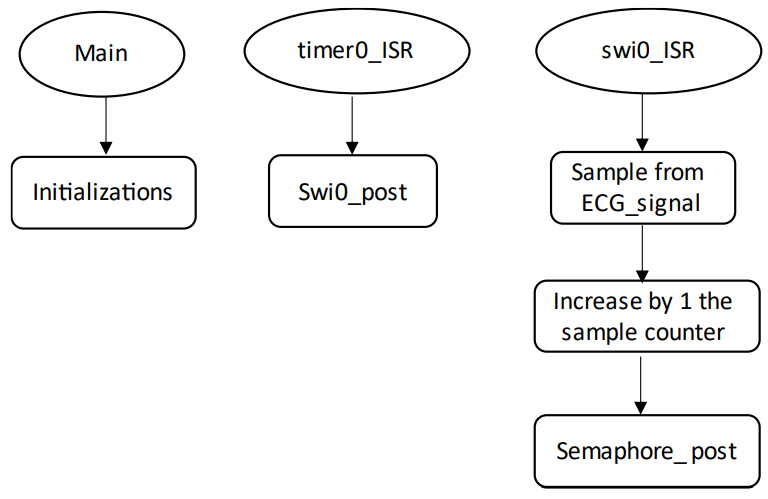
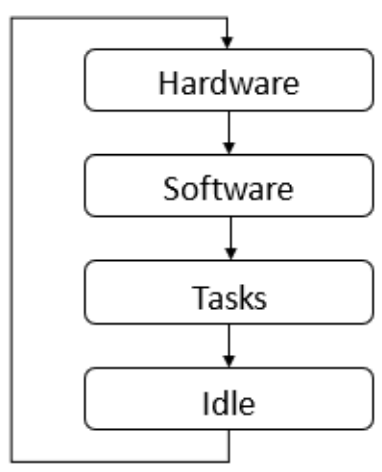
* Detect the time interval between the main beats (between S1 and S1) and between the main beat and the secondary beat (between S1 and S2), and calculate the heart rate based on the time interval between the main beats.
* Display the heart rate on the console screen in bps format.

In addition, we chose to display the screen rate in bpm format and the time intervals between beats and their indices in the sample array.

* Presentation of the heartbeat pulse signal graphs at the different processing stages - the original signal, after passing through the IIR filters (LPF , notch filter and then HPF) , after passing through a full wave rectifier, and after passing through an envelope detector in the time domain and frequency domain.
* Display the execution graph, CPU load, and task load.

1. **Block diagram of the program:**
2. **Flow chart of the system:**

The priority of the threads in the RTOS system is in the following order (includes the software interrupts , hardware interrupts , the task and the idle) is in the **left** hand side image. The **right** hand side image is the execution order of the threads in every run:



1. **Design Considerations-filters:**

The goal of that project is to calculate a person's heart rate and display it in beats per second (bps) format based on their heartbeat pulse signal given us by the instructor of the course.

The first step after displaying the "raw signal" in TD and in FD will be to understand the necessary filtering procedures needed to be done on the "raw signal".

First filter: An IIR elliptic filter was chosen with 7 order (4 sections) to filter the high frequencies – in which a sinusoidal disturbance is exist.

Second filter: An IIR elliptic filter was chosen with 6 order (3 sections) to filter the low frequencies – in which DC noise is exist.

We found that in real heartbeat pulse signal – the data should be stored in the frequencies between 20 Hz to 200 Hz only. Thus the data stored in the low frequencies can be eliminated by a HPF.

third filter: An IIR Chebyshev type 2 filter was chosen with 6 order (3 sections) to filter an unusual disturbance in 60 Hz. Thus we picked to filter using a notch filter (with making sure that we don’t harm the signal because of overshoot or from the other side – don’t create too "expensive" filter in coefficient terms.

To conclude the filter stage, we think that we succeeded to filter the signal with the minimum coefficients fine and maximum results and performances.

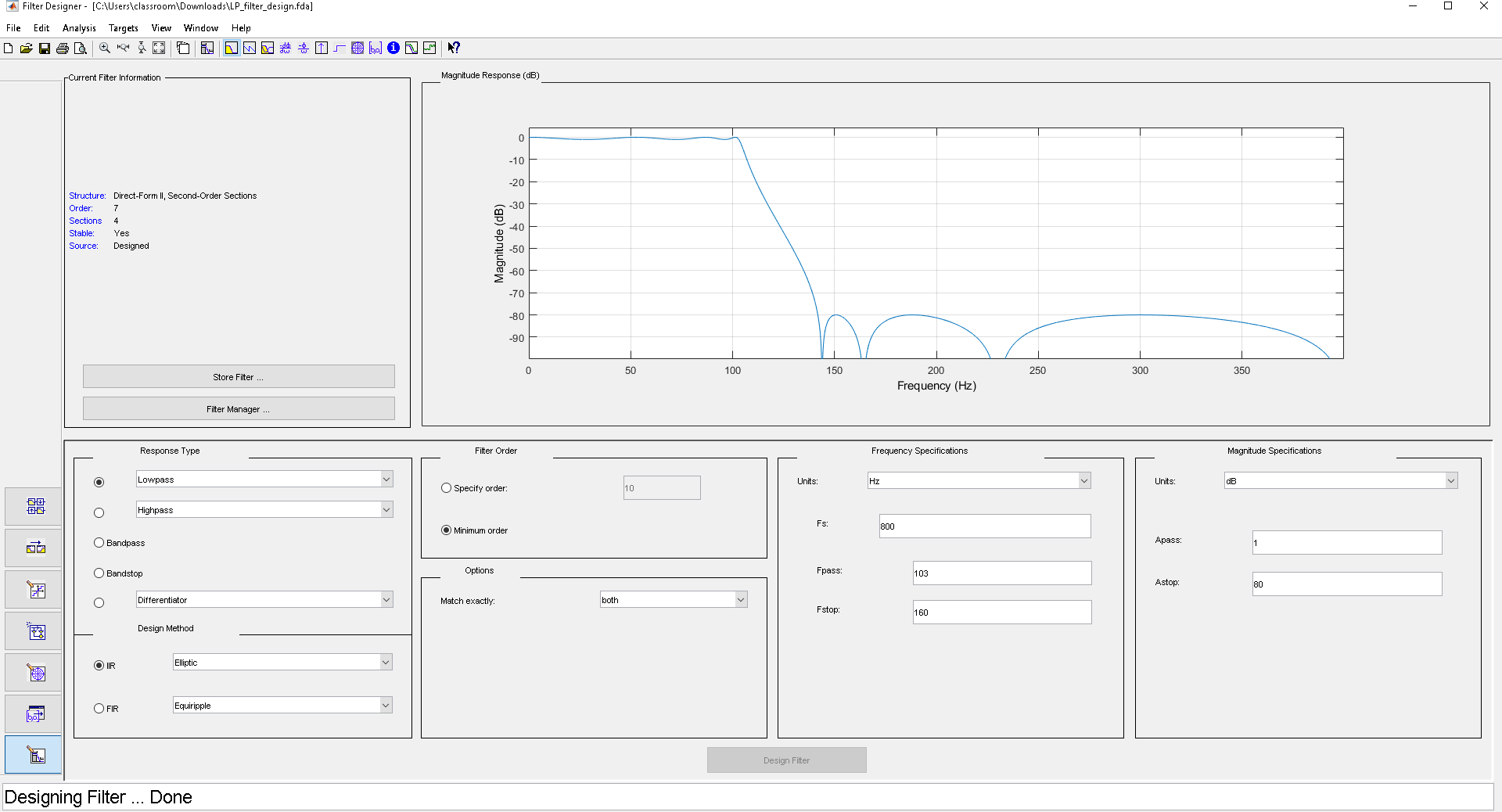
Need to be said that we simulated the entire process of filtering and simulating the signals in TD and in FD in Matlab. The entire collected data and graphs will be presented in that report.

In addition - The rationale for these design choices will be explained later.

Program implementation: The program provided in Lab 6 of the course (IIR Filter) was used as a starting point (the function of the IIR BIQUAD) , and changes were made to it, such as changing the cycle time of each sample in timer0.

In addition, functions relevant to the project goal were added, which will be explained later. The rest of the resources remained unchanged.

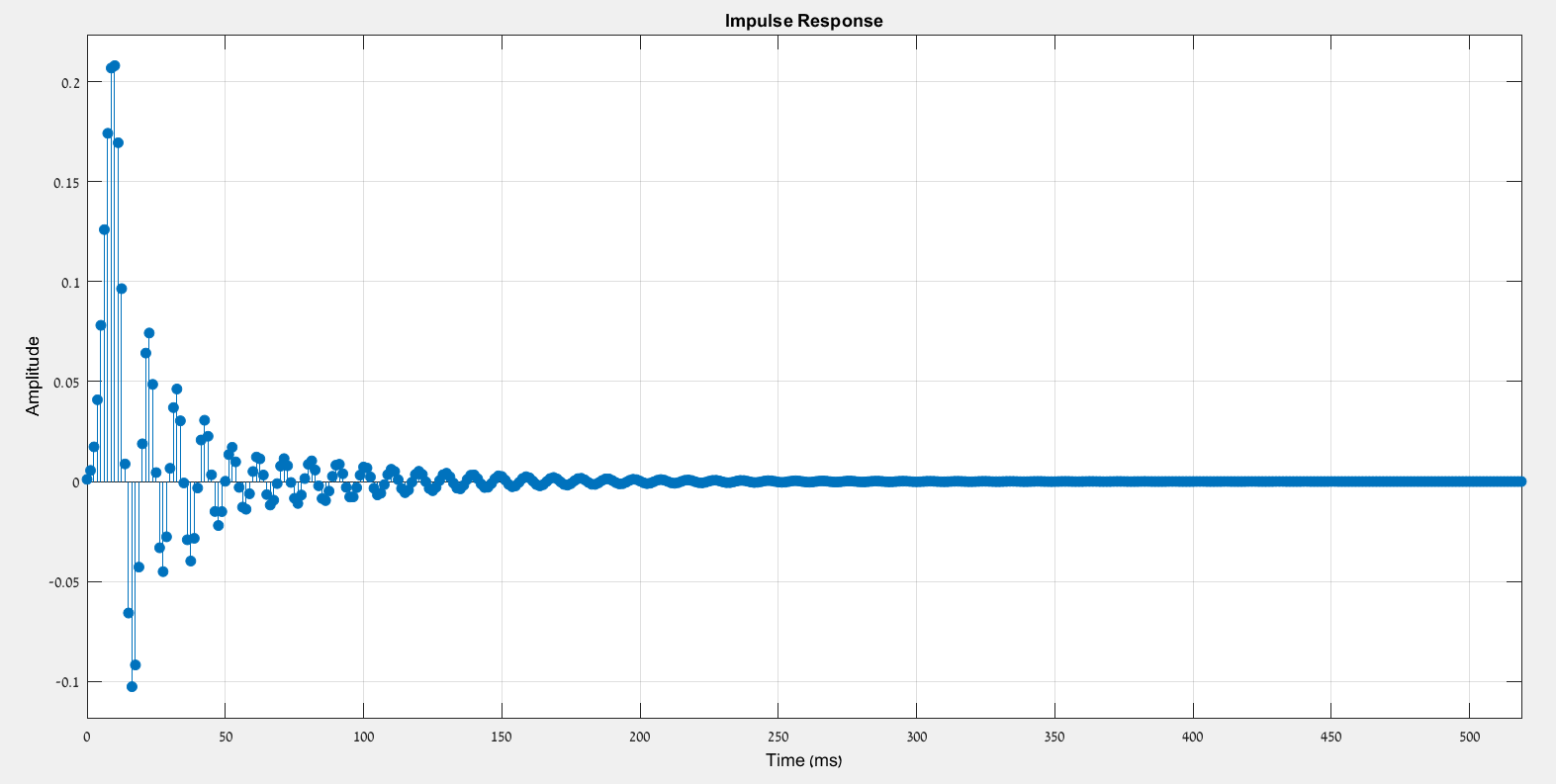
1. **LPF design considerations:**

The Matlab filter designer of the LPF is presented:

One can see that we picked in addition to choosing to work with elliptic IIR filter in a minimum order (to minimize the number of coefficients) in addition we decided to pick the maximum amount of so we chose to use (another level of may increase the number of coefficients – something we don’t want).

So - One can see when looking at the Frequency Response Plot A steep drop-off in magnitude shortly after 100 Hz, with deep attenuations reaching below -80 dB. The transition band, the range between the passband edge and the stopband edge, is very narrow.

We picked – aka large transition band because in an ideal lowpass filter, the transition from passband to stopband would be instantaneous (a vertical drop). In practical filters, some transition band exists-its width is dependent on the filter order and design. For an elliptic filter, which is designed for a maximally steep roll-off given a filter order, this plot seems to show expected behavior with the specified parameters.

in addition we will present here the impulse response of that filter:

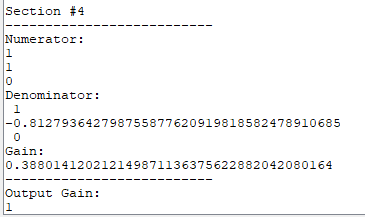
we can see that the response starts with a high peak and then exhibits oscillations that decrease in amplitude over time. These oscillations are due to the ripple in both the passband and the stopband. The response is approaching to zero over time, indicating a stable filter-it means that it will not cause unbounded outputs in response to bounded inputs (BIBO stable).

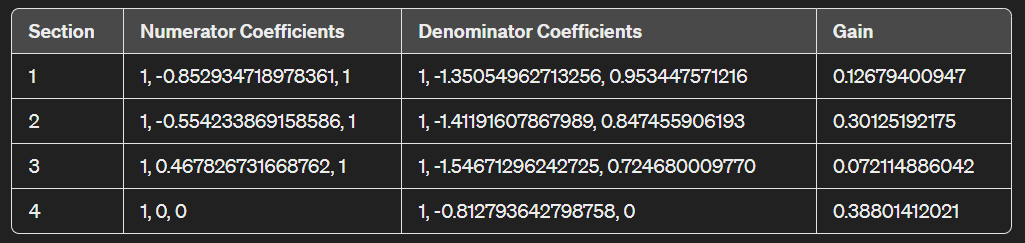
A graph of a circle

Description automatically generatedAnd the pole-zero map of the LPF is:

One can see that Poles (X's) are inside the unit circle, and zeros (O's) are on or just outside the unit circle. All poles should be within the unit circle for stability, and the zeros can be inside or outside the unit circle. For an elliptic filter, we expect zeros to be close to the unit circle, which corresponds to the sharp attenuation in the stopband. The placement of poles and zeros confirms that the filter is stable and is designed to have a fast roll-off. Zeros close to the unit circle create the deep notches in the frequency response, contributing to the filter's sharp transition from passband to stopband.

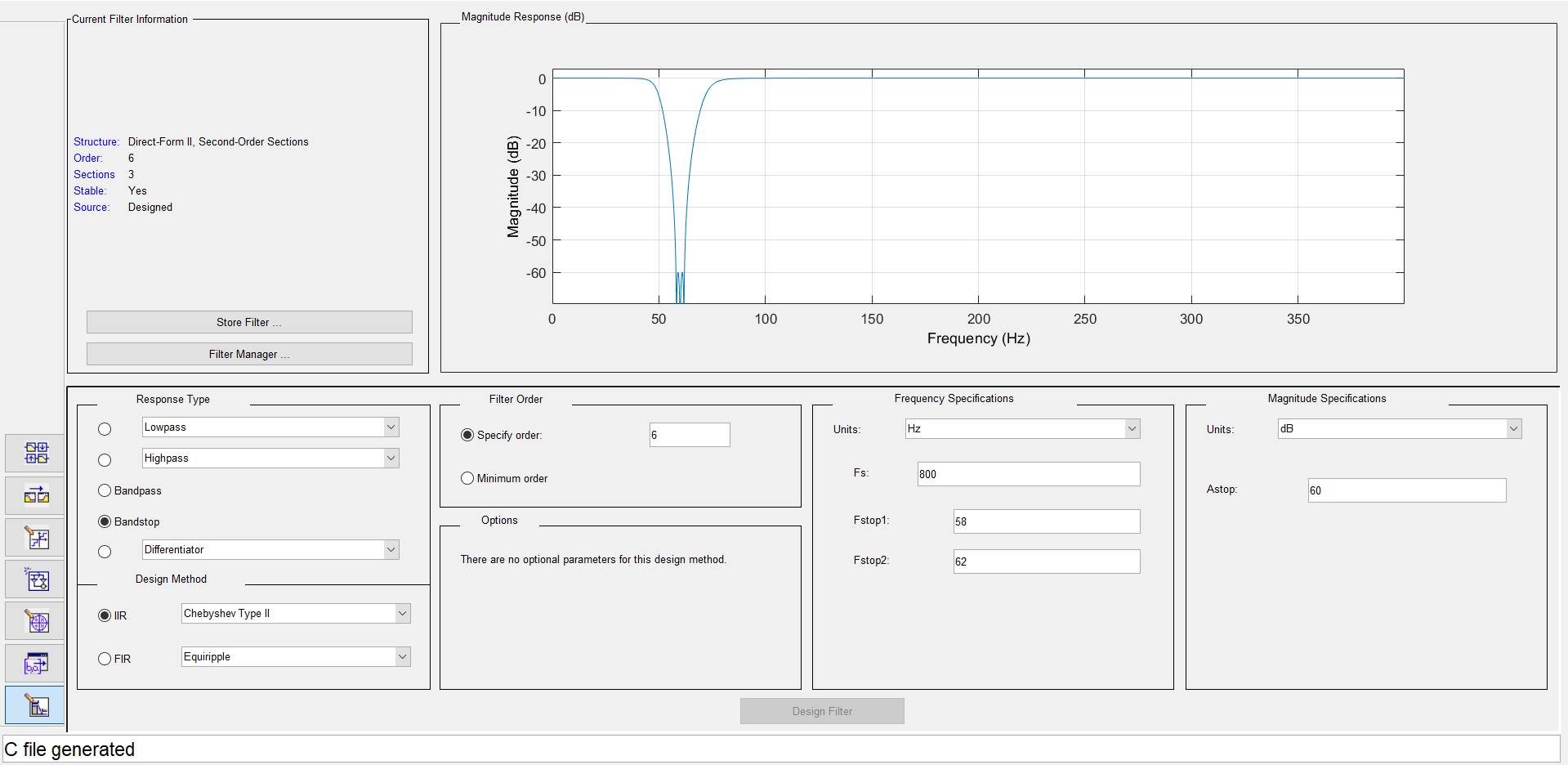
To sum up the first LPF design considerations we will present the sections of the filter as it was presented in Matlab filter designer – we need the sections for a header file that will contain all the sections of the filters we intend to use in this project:





1. **Notch filter considerations:**

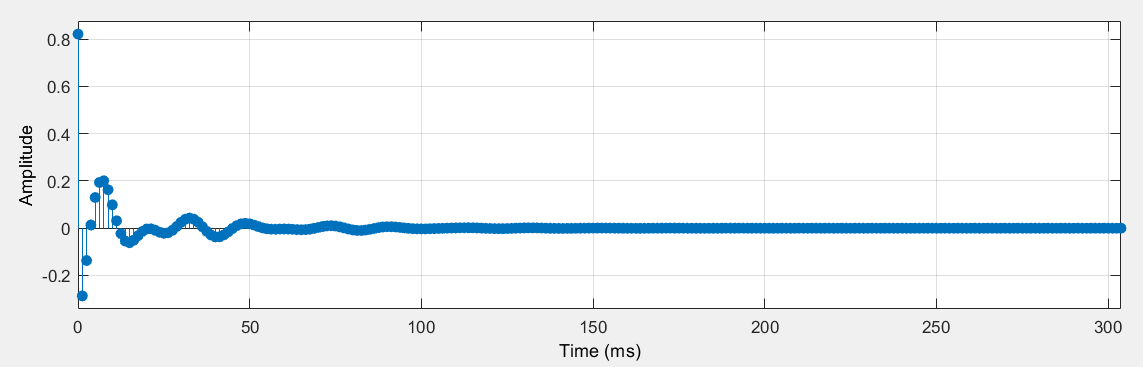
The Matlab filter designer of the notch filer (band stop filer with a specific )

 is presented: (we picked notch filer because there was a delta disturbance in 60 Hz)

One can see that we picked in addition to choosing to work with Chebyshev type II IIR filter when we specify order 6 such that we will get the same dimensions as the LPF before (for easier implementation in the fdacoeffs C header). in addition, we decided to pick lower amount of so we chose to use (again-for matching at all costs the LPF dimensions – order 6 with 3 or 4 sections)

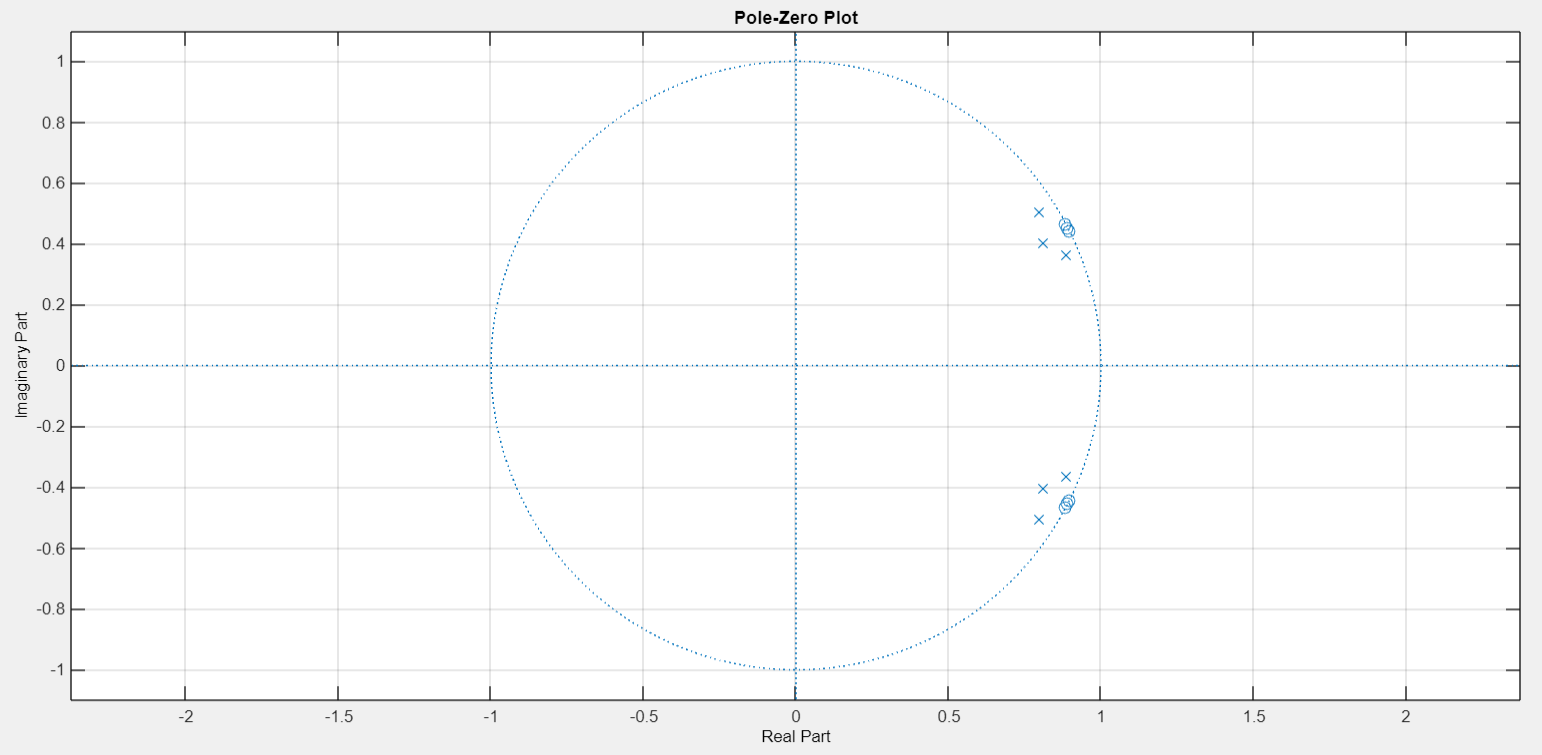
We expect to see a sharp dip in the frequency response at the notch frequency which is around 60 Hz.

The magnitude response graph shows a very sharp attenuation around the 60 Hz mark, which is consistent with the purpose of a notch filter. The magnitude specifications indicate a 60 dB attenuation in the stopband, which is quite significant.

in addition we will present here the impulse response of that filter:

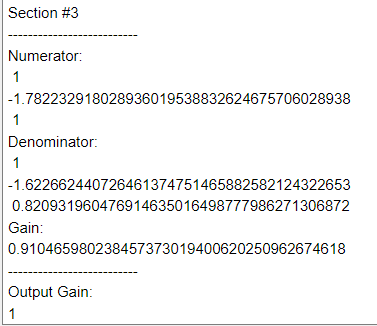
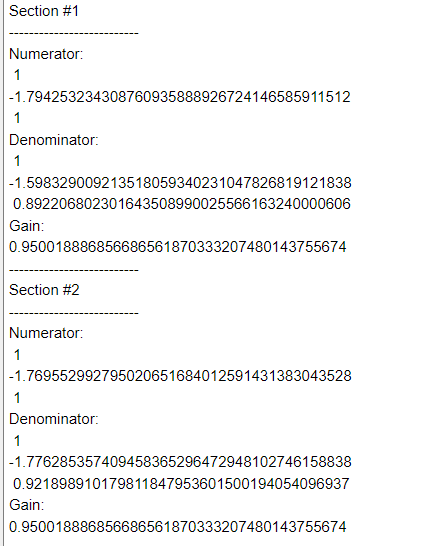
The impulse response shows how the filter reacts to a delta function input. The initial peak is followed by a decaying oscillation. Typically, the impulse response of a notch filter will have an oscillatory decay, which reflects the filter's resonant frequency around the notch frequency.

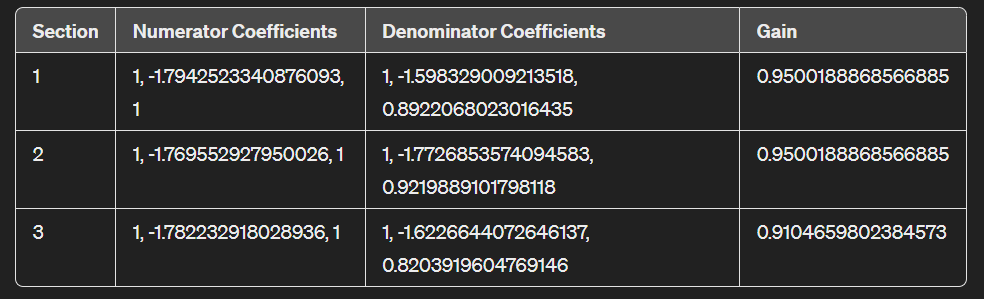
The behavior seen here means that the filter has a resonant frequency close to the notch frequency, and it quickly stabilizes, which is good for transient response.

And the pole-zero map of the notch filer is

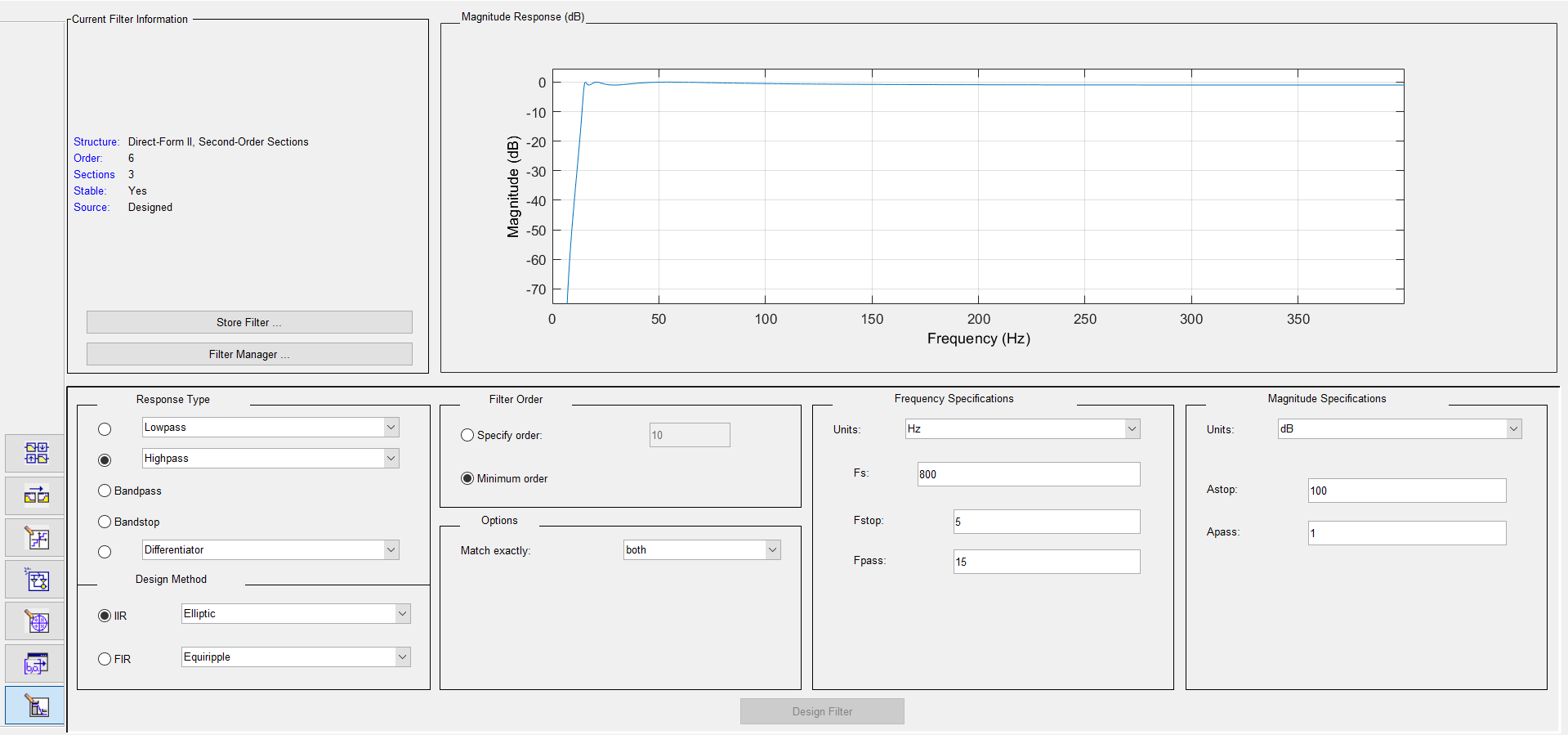
The pole-zero map displays the poles (X) and zeros (O) of the filter's transfer function in the complex plane. All poles are within the unit circle, confirming stability. For a notch filter, you would expect to see pairs of poles and zeros closely aligned on either side of the unit circle, which create the sharp attenuation at the notch frequency.

To sum up the notch filer design considerations we will present the sections of the filter as it was presented in Matlab filter designer – we need the sections for a header file that will contain all the sections of the filters we intend to use in this project:





1. **HPF design considerations:**

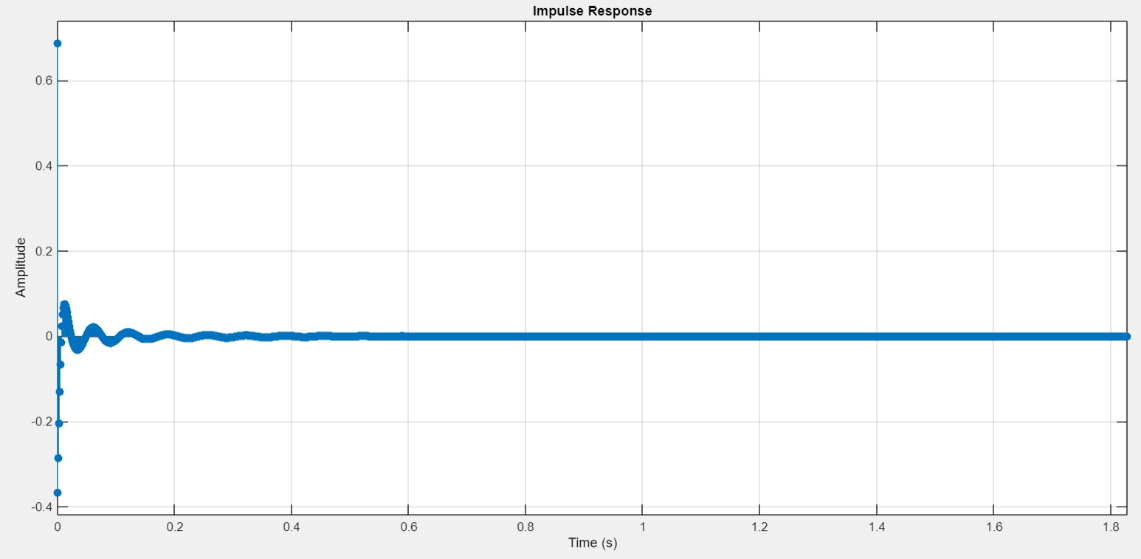
The Matlab filter designer of the notch filer:

One can see that we picked in addition to choosing to work with elliptic IIR filter when we demand minimum order such that we will get the same dimensions as the notch before – order 6 with 3 sections (for easier implementation in the fdacoeffs C header).

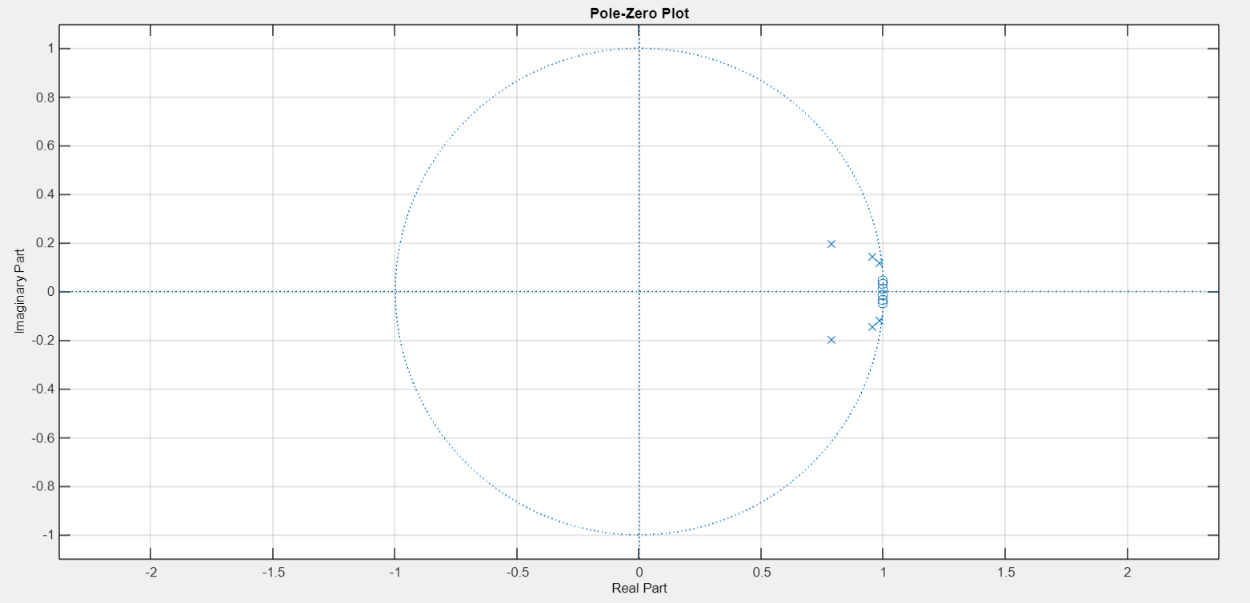
in addition, we decided to pick high amount of so we chose to use (again-for matching at all costs the LPF dimensions – order 6 with 3 or 4 sections)

we made "trial and error" such that we will get the number of sections (and order number) that we wanted – all for easier implementation in the header coefficients file – that is the reason that we don’t keep on consistency on "filer order" choosing option. We did preferred in the filter designing process to use Elliptic filter because it is known for its steep transition bands.

From the magnitude response one can see a steep drop starting before 5 Hz and stabilizing around 15 Hz. For a high-pass filter designed to eliminate DC disturbances, the expectation is a quick transition from stopband to passband just above 0 Hz, effectively blocking out DC. In addition - The filter shows a stopband attenuation () of 100 dB, which is significant, meaning it heavily attenuates frequencies below the passband. The passband ripple () is 1 dB, which is typical for filters where some ripple is acceptable. The transition from a very deep stopband to the passband is sharp- **It means that the filter will effectively block DC and very low-frequency components.**

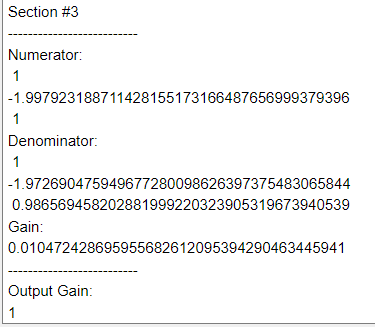
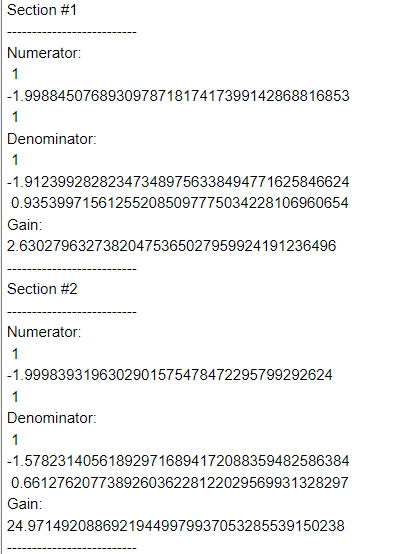
in addition we will present here the impulse response of that filter:

The filter shows a slightly oscillatory settling towards zero, which might introduce some ringing in the time-domain response. However, the amplitude of these oscillations decreases quickly, which is a good indication of filter stability and robustness.

And the pole-zero map of the notch filer is

The pole-zero plot indicates the location of poles (X) and zeros (O) in the s-plane. Zeros are on the origin (consistent with high-pass characteristics), and poles are inside the unit circle, which confirms stability. In a high-pass filter, zeros at the origin are expected because they enforce the attenuation of DC (0 Hz). Poles should be within the unit circle for stability.

To sum up the high pass filer design considerations we will present the sections of the filter as it was presented in Matlab filter designer – we need the sections for a header file that will contain all the sections of the filters we intend to use in this project:

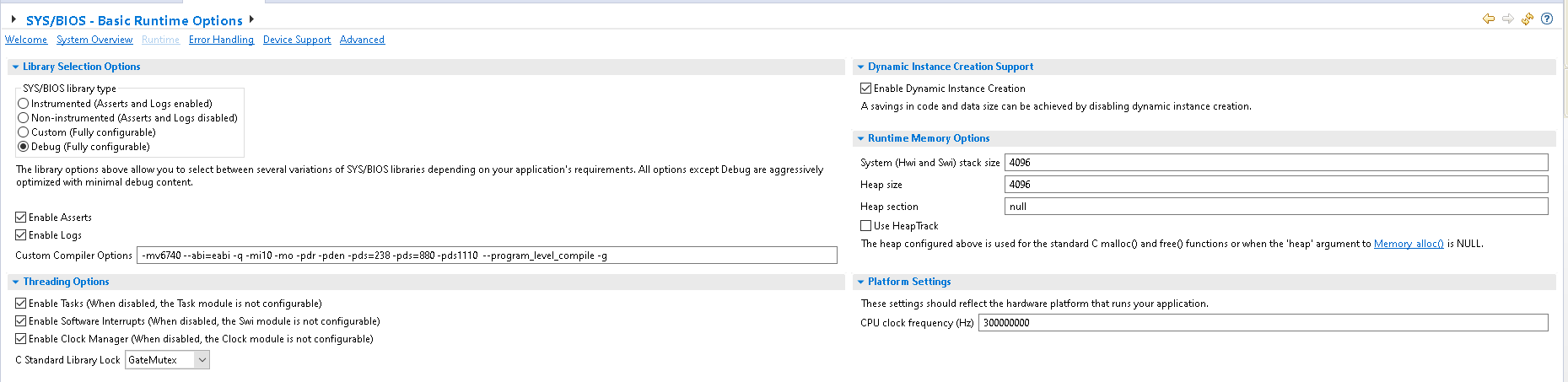


A screenshot of a computer

Description automatically generated

1. **System basic definitions – in cfg file :**
2. Bios:

This unit is used for real-time thread management (communication and synchronization), peripheral support, interrupt handling, and memory management. In this unit, we define the CPU clock frequency, which we chose to leave at 300 MHz, as it was in the template provided by the instructor in Chapter 3 exercise.



1. Timer:

timer0 is a counter used to create fixed time intervals. It operates based on the system clock and can be reset after reaching a predefined maximum value.

In this project, the timer0 is used to generate time intervals between samples, according to a sampling rate of 800 Hz. In addition we chose to work with timer ID of ANY to prevent problems in the system ( as the instructor of the course advised us to do).

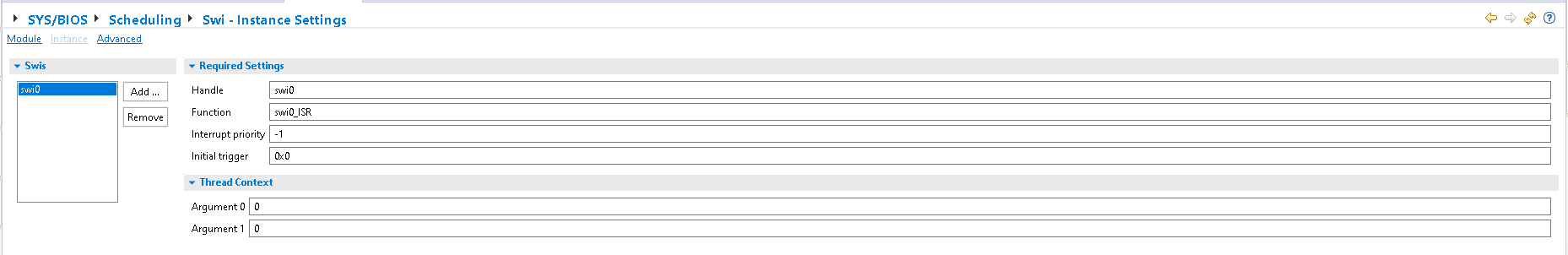
The counter is configured to count in "period in microsec" format, therefore its maximum value is calculated using the formula:

1. Swi :

An swi0 is a software interrupt for the processor.

It's used to switch between tasks, ensuring a higher priority task (the one within the swi0 interrupt routine) takes precedence over a lower priority task. In our case, the call to swi0 occurs within the timer0 interrupt routine.

Within the 0swi interrupt routine, we sample a value from the ECG\_signal array for processing.



1. Semaphore:

Semaphore0 is used for synchronization in multithreaded operating systems. It allows one thread to control when other threads can access a shared resource, while they continue their work during the wait. This ensures that only one specific thread uses the shared resource at any given time.

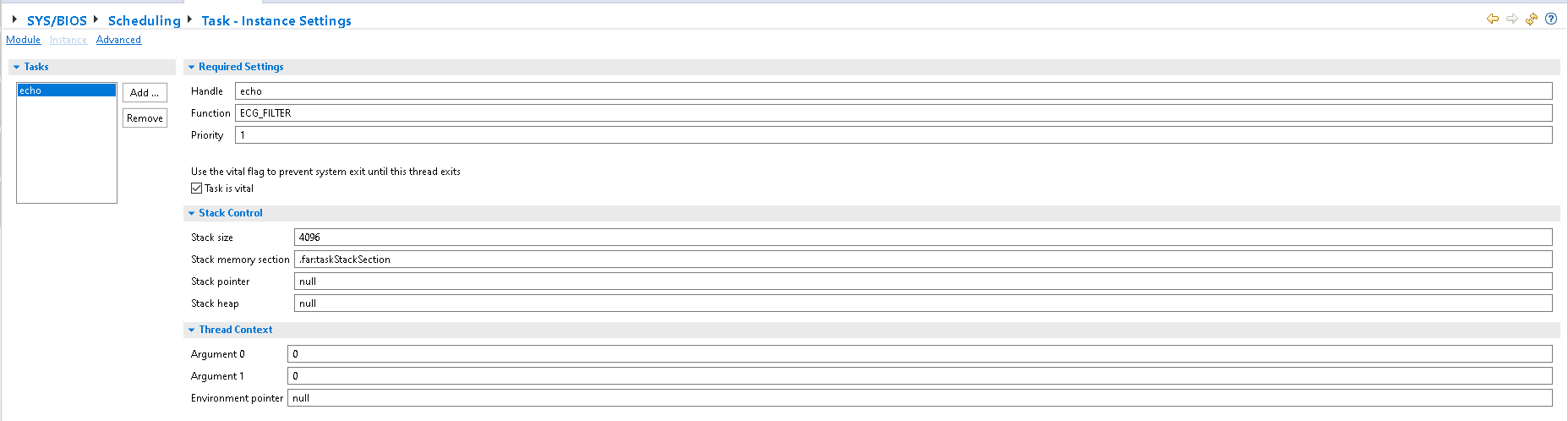
A white rectangular object with a black stripe

Description automatically generated with medium confidenceIn this case, the semaphore synchronizes calls to TASK at the end of the swi0 interrupt routine The semaphore signals a resource is available using the semaphore\_ post command -while In the TASK function The semaphore signals to wait for a resource using the semaphore\_pend

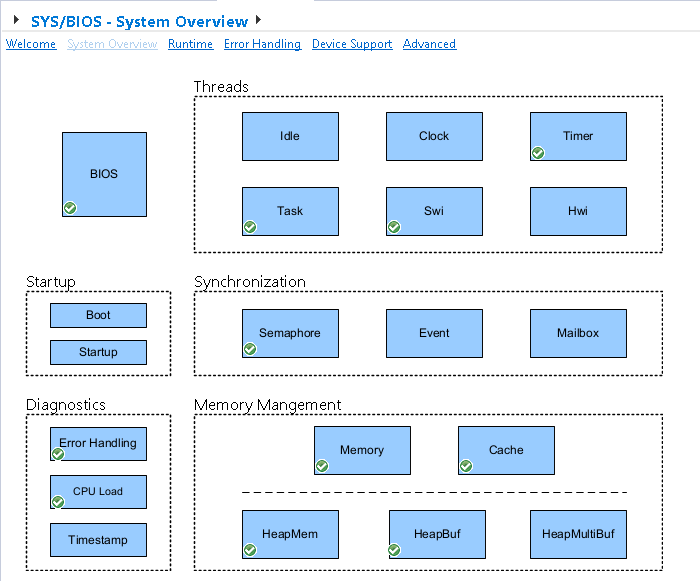
1. Task:

This is the work unit that is executed by the program. It can be used to organize and perform actions in a specific order or at specific intervals. In our case, it is a kind of main function within an infinite loop that calls all the other functions in the program.

The functions are IIR\_Biquad, digital\_filter, and peak\_counts (excluding the breaks).



1. System overview:



1. **System's functions :** we used C headers for easier code reading – the different C headers are presented in that section:
2. **ECG-signal.h :**

This is a header file that contains an array representing the ECG signal for processing. The array is called ECG\_ signal accordingly, and it contains 1788 values representing the electrical signals of the heart over time.

A black screen with white text

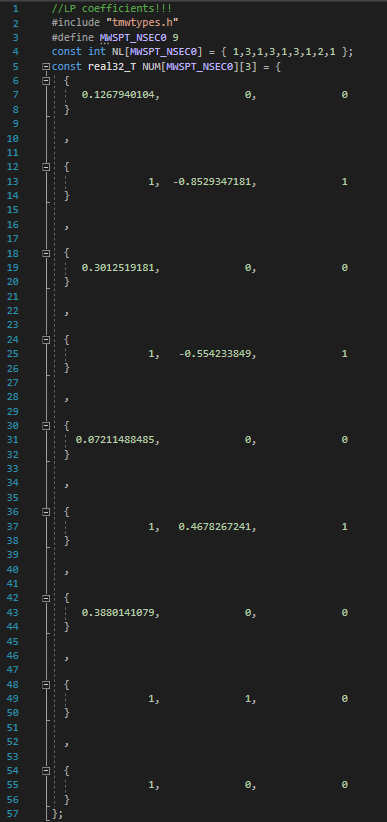
Description automatically generatedThis file was given to us by the lecturer and we did not implement it, except for placing it in the program. In the main program, we read a value from it at each sample.

There are 1788 values in the ECG\_signal array – the image is for verifying that the signal is the one that we were given. (the header can be found in the project's ZIP)

1. **fdacoefs.h :**

A screenshot of a computer program

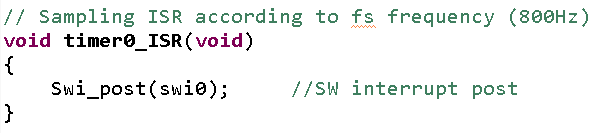
Description automatically generatedThis is a header file that defines two-dimensional arrays containing the coefficients of the IIR filter we designed using MATLAB software. These are the numerical values that define the characteristics of the filter according to the parameters we entered in the MATLAB Filter Designer interface.

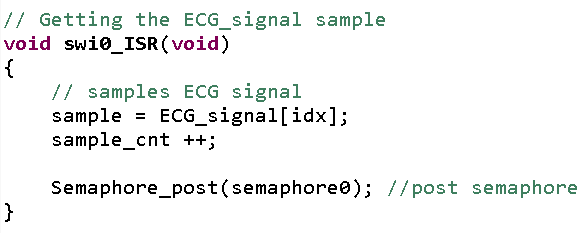


A computer screen with numbers and lines

Description automatically generated

All the coefficients that can be seen in the C header file – are numerators , denominators and gain of the filters exactly as explained earlier:

1. **timer0\_ISR ()**
2. **swi0\_ISR ()**

In this routine, a value is sampled from the **ECG\_signal** array and stored in a variable called sample. We defined another variable called **sample\_cnt** which counts up every time a sample is taken. Later in the program, we can see that **sample\_cnt** is reset after 800 samples, which represents one second, and then starts counting again. At the end of the routine, a semaphore is called to signal that resources are available.

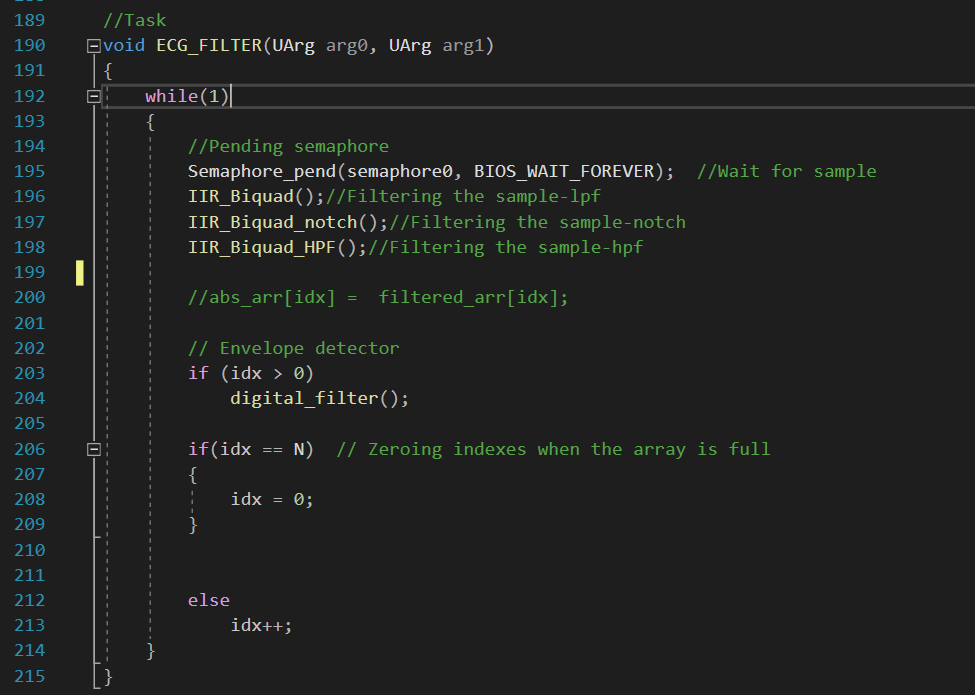
1. **ECG\_filter()**

This function is the Task function, and it is the main function of the program. It calls the IIR Biquad functions (we have 3 IIR Biquad functions for our 3 filters) to perform a preliminary filtering of the sampled signal.

After finishing the filtering stage – the **IIR\_Biquad\_HPF (last filter function)** performs an absolute value operation on the signal that came out of the filter (absolute value is a description of a full wave rectifier) and stores the value in the **abs\_arr** array.

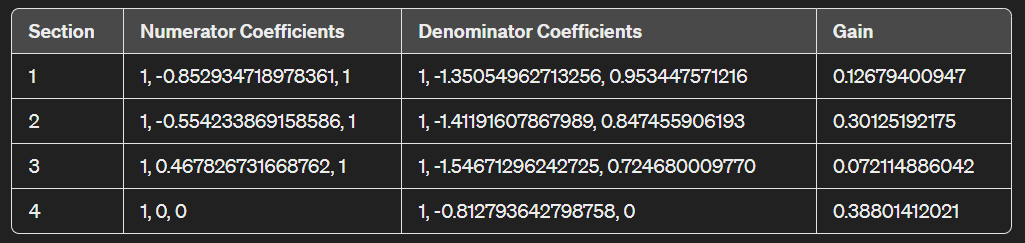
Next, the **IIR\_Biquad\_HPF** performs an envelop detector operation (digital filter). After the signal passes through the above stages, it calls the **peak\_count** function which counts the number of peaks (inside the count\_peak function there is a call to a function which calculates the time intervals between the peaks and the heart rate). At the end of the program, the index of the sample array idx increases by 1 until it reaches the end of the array (N samples) and resets. Along with the reset of idx, the variables that store the peak indices and the peak counter are also reset.

**Overall , the task function is responsible only for the calling to the filtering IIR\_Biquads functions.**



1. **IIR\_Biquad () – first filtering function:**

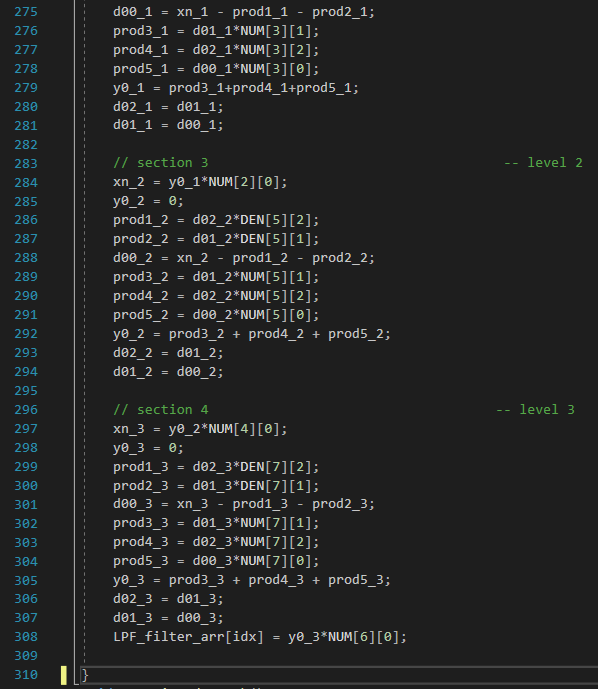
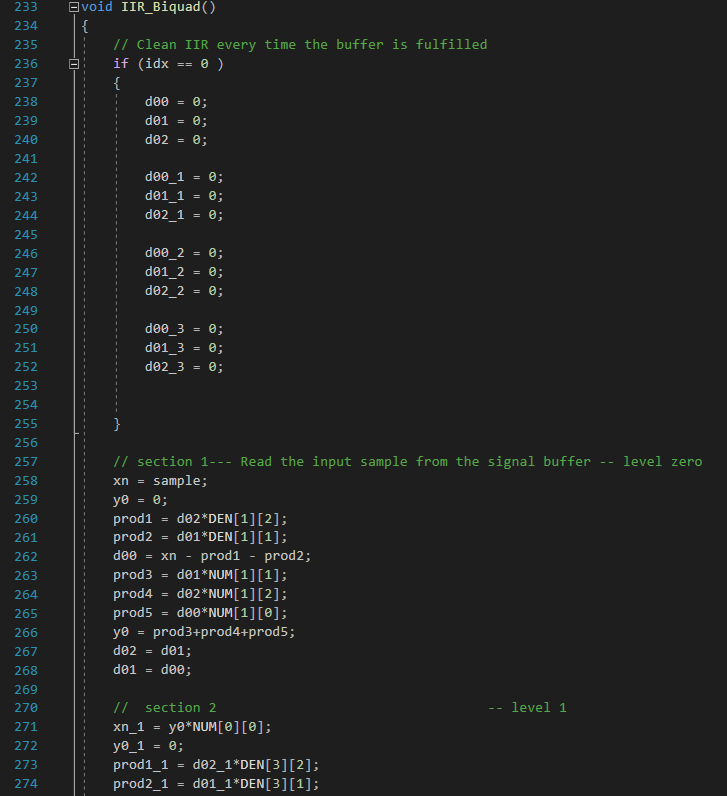
This function is a digital LPF IIR filter for filtering unwanted frequencies in the given signal. We previously demonstrated the design of the filter and it’s coefficients in a numerator, denominator and gain prespective.

It takes as input the sampled value from the ECG\_signal array and outputs after 4 stages the filtered value.

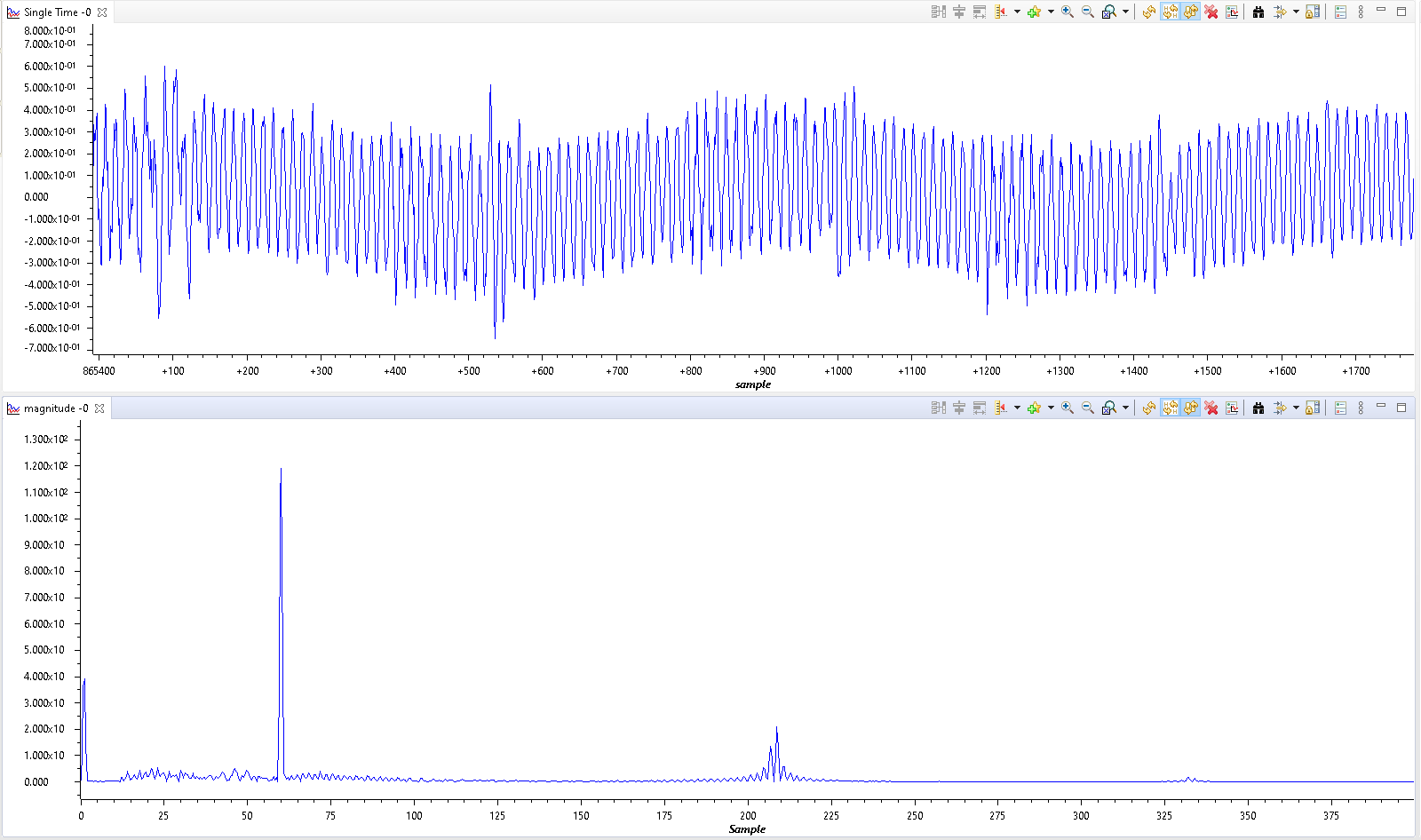
The code base was taken from the lecture presentation on IIR Filter.

We added the **LPF\_filter\_arr** array in order to see the output signal from the filter:

First we will present the IIR\_Biquad () itself:



And in the next page we will present the results.

first we plotted the raw signal – as was given to us – has 1788 samples and we sample in a sampling frequency of 800 Hz. We plotted the raw signal in TD and in FD:

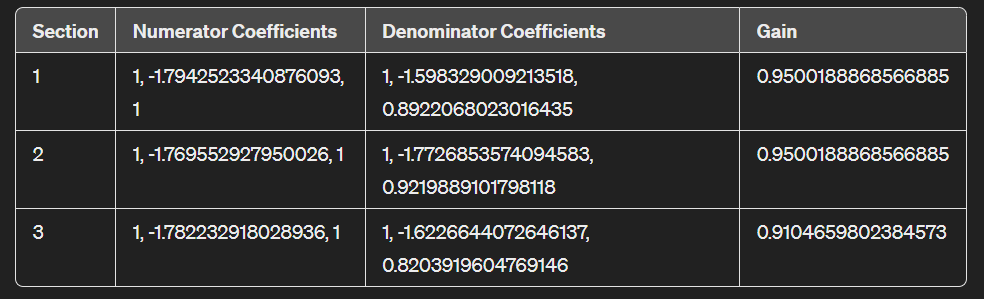
A screenshot of a computer

Description automatically generatedwe activated the LPF such that the sinusoidal disturbance in the high frequencies disappeared! in the image below – one can see the LPF\_filter\_arr (array that stores the results of the first IIR\_Biquad – in TD and in FD:

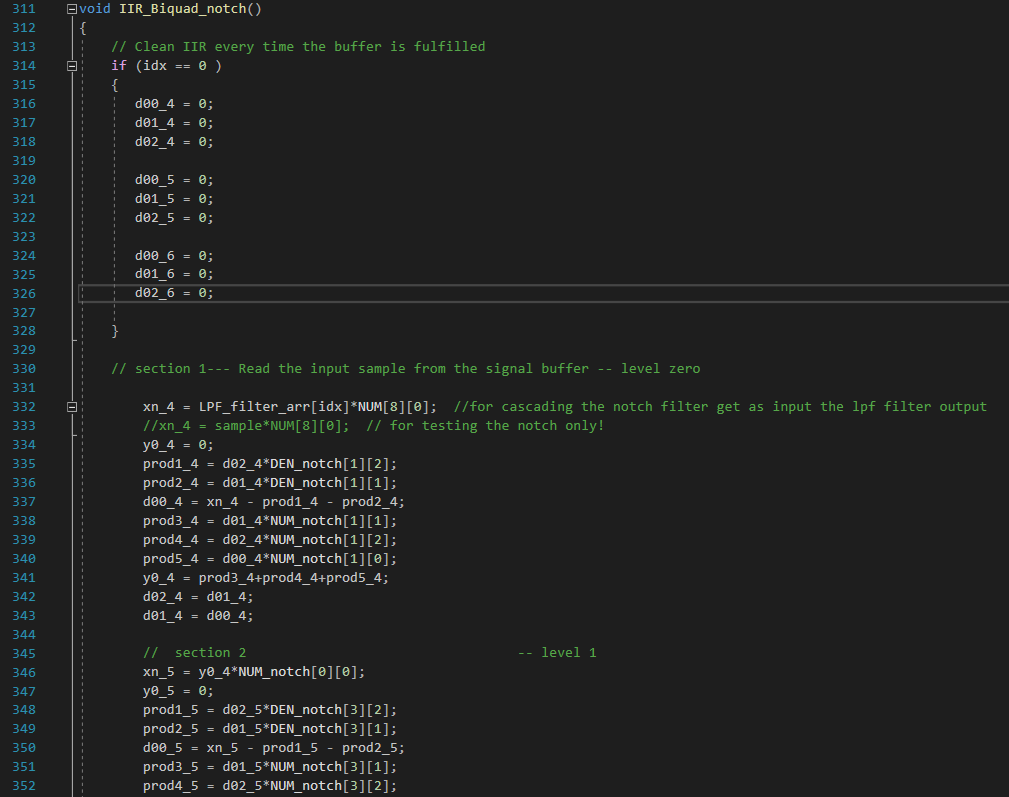
1. **IIR\_Biquad\_notch () – second filtering function:**

This function is a digital BSF IIR filter -notch filter for filtering unwanted frequencies in the given signal-in this case we want to eliminate a delta disturbance in 60 Hz.

We previously demonstrated the design of the filter and it’s coefficients in a numerator, denominator and gain perspective.

It takes as input the sampled value from the ECG\_signal array and outputs after 4 stages the filtered value.

We added the **notch\_filter\_arr** array in order to see the output signal from the filter:

First we will present the IIR\_Biquad\_notch itself:

A screenshot of a computer program

Description automatically generated

And now we will present two results – the first will be the output of the notch filter when the LPF is not activated (checking only the notch ability to filter according to the results we got in the Matlab):

And the second will be the output of the notch filter when the LPF is activated (checking the cascading procedure):

1. **IIR\_Biquad\_HPF – third filtering function:**

This function is a digital HPF IIR filter - for filtering unwanted frequencies in the given signal-in this case we want to eliminate disturbances in the low frequencies.

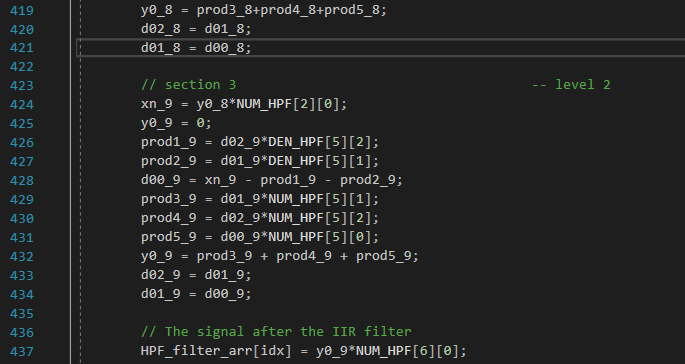
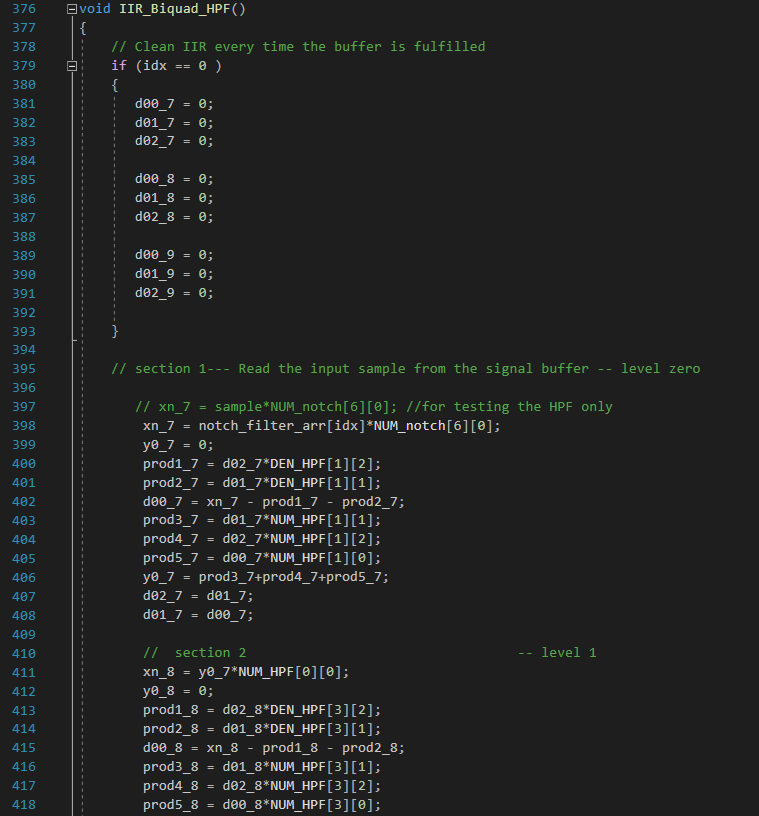
We previously demonstrated the design of the filter and it’s coefficients in a numerator, denominator and gain perspective.

A screenshot of a computer

Description automatically generatedIt takes as input the sampled value from the ECG\_signal array and outputs after 4 stages the filtered value.

We added the **HPF\_filter\_arr** array in order to see the output signal from the filter:

First we will present the IIR\_Biquad\_notch itself:



A screenshot of a computer

Description automatically generatedAnd now we will present two results – the first will be the output of the HPF when the notch and the LPF are not activated (checking only the HPF ability to filter according to the results we got in the Matlab):

A screenshot of a computer

Description automatically generatedAnd the second will be the output of the HPF filter when the LPF and the notch filters are activated (checking the cascading procedure):

As one can see – the filters work well alone (when each of them is the only one that works – aka when the input signal to them is the sample of the ECG\_signal – and in addition the cascading works correctly and the filtering procedure is good.

Pay attention to the cascading: ( LPF all the way to the notch filter)











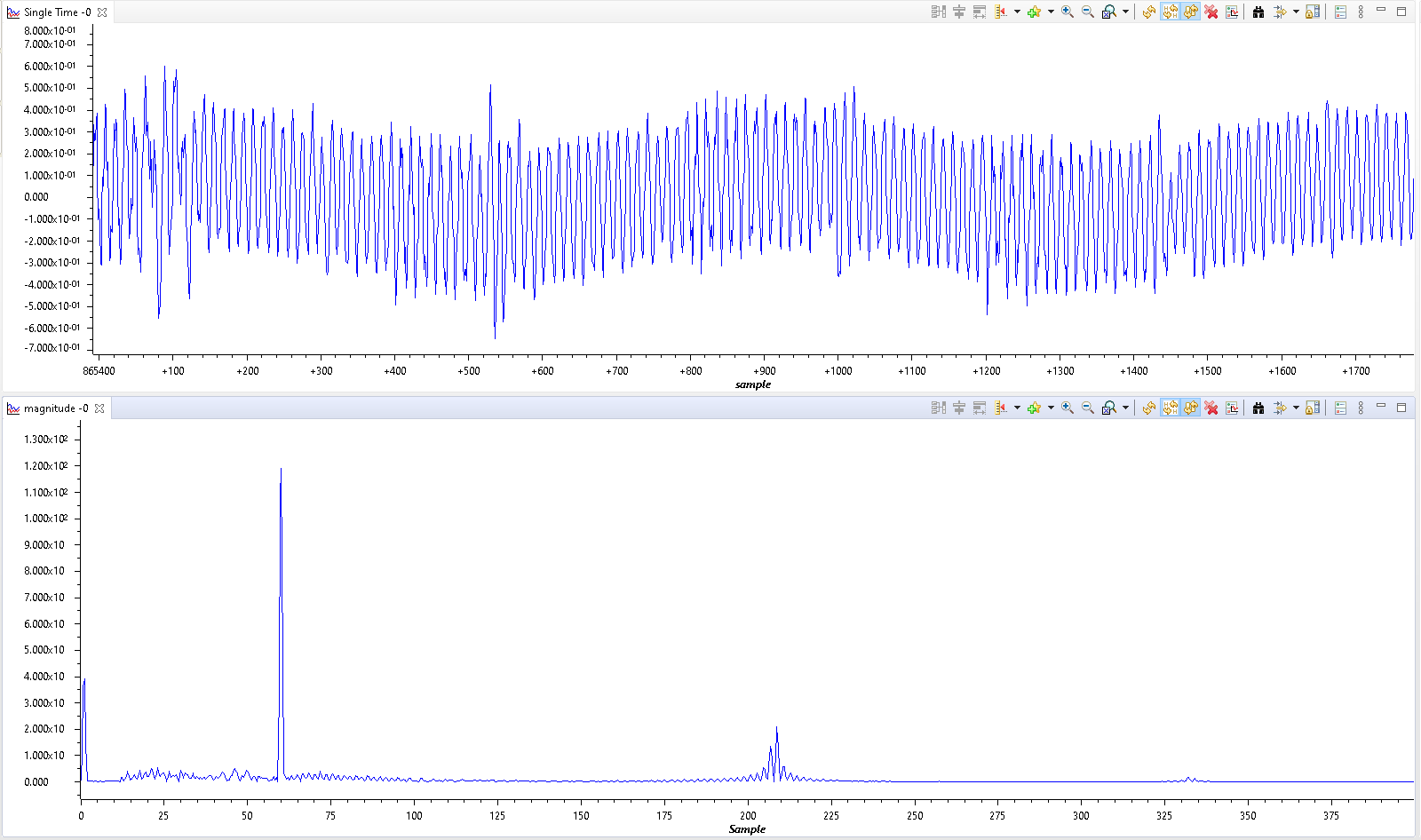
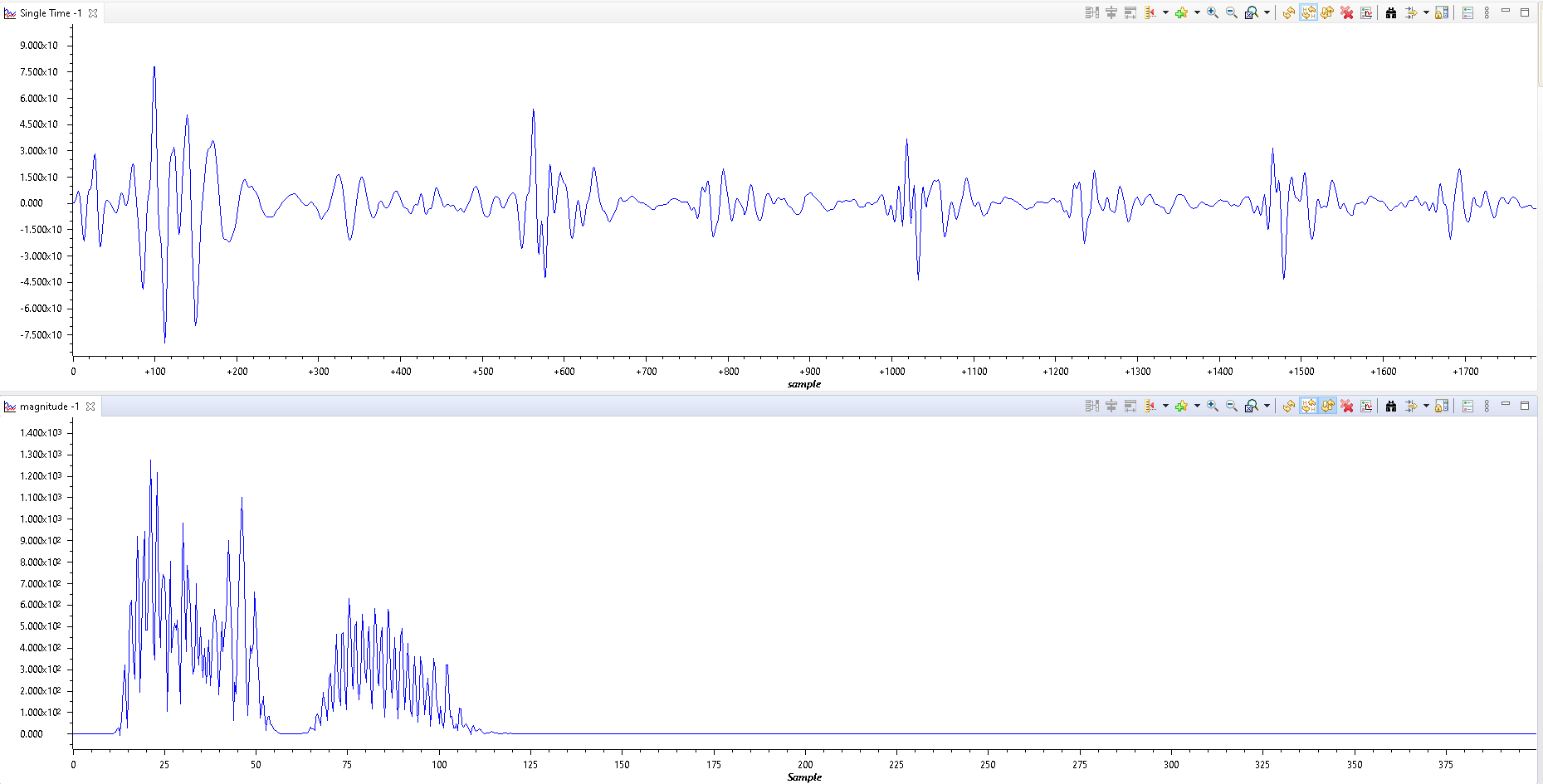








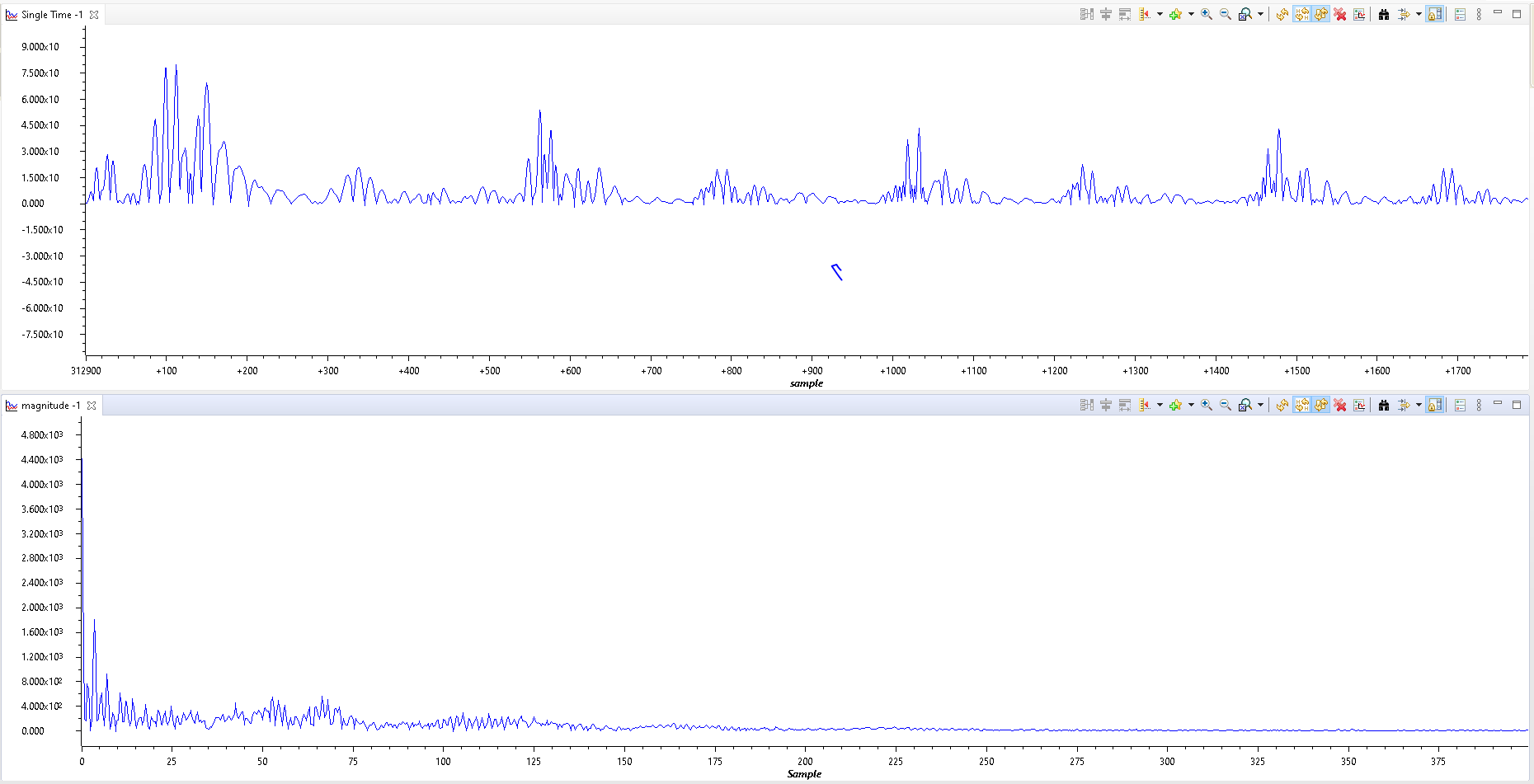
So – looking at the plots in TD and in FD of the raw signal and the filtered signal:



1. **Absolute value:**

After passing through the IIR filters (the output of the 3 filters is **HPF\_filter\_arr**) we want to perform an absolute value operation on our signal. The ABS operation is a preparation to the envelop detector operation (digital filter):

In the following image the output of the absolute value operation (saved in an array named abs\_arr in TD and in FD:



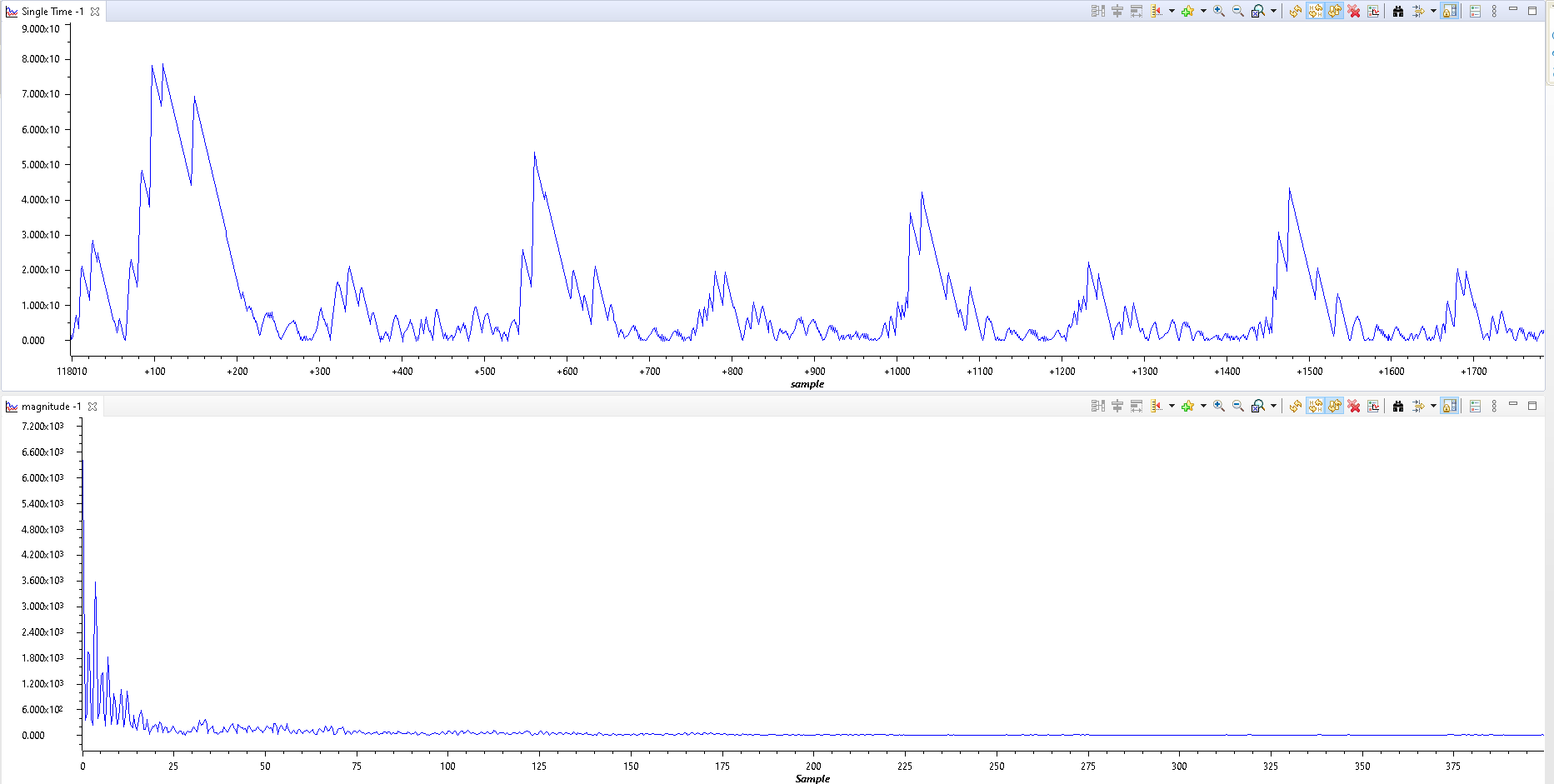
1. **Envelop detector:**

This is a digital envelope detector. After passing through the IIR filters (the output of the 3 filters is **HPF\_filter\_arr)** and performing an absolute value operation, the sampled signal goes through a digital filter. The envelope detector is an algorithm used to extract the envelope of a modulated signal, which represents the original information contained in the signal. In the analog world, the envelope detector consists of a diode, resistor, and capacitor, where the diode allows for the rapid charging of the capacitor, and the resistor allows for the slow discharge of the capacitor. In software implementation, the envelope detector works by comparing the current numerical value with the previous numerical value: if the current value is greater than the previous one, then the current value is saved; if the current value is less than the previous one, the current value is not saved and it decreases by a certain delta, i.e., it drops at a constant slope. The value of delta is found in a variable called num-its value was chosen to balance the slope of the increase with the slope of the decrease of the information signal.

A computer screen with white text

Description automatically generatedThe code of that part is : (num will be picked up using trial and error) (dividing factor is another variable that we found to be the optimal at **divider\_factor=2**)

the result of that part will be stored in an array named "**abs\_arr\_envelop**"

In the following image we show the signal after the envelop detector with **num=1** :

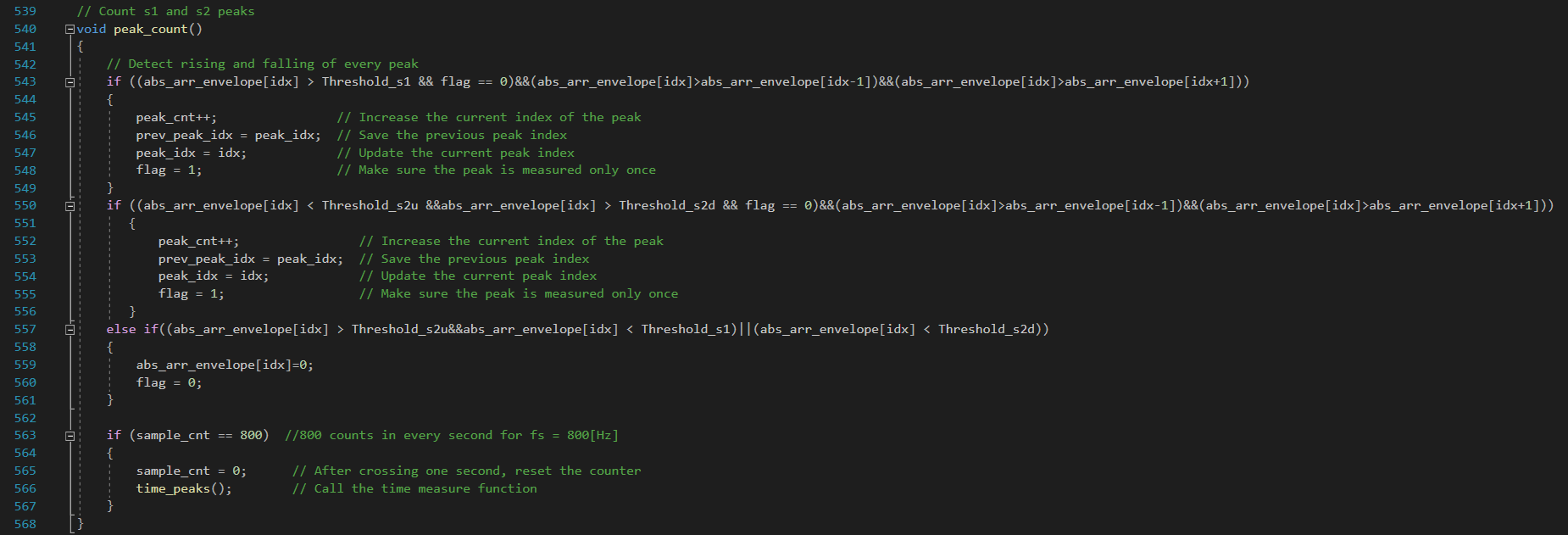
A screenshot of a computer

Description automatically generatedAnd in the following image we show the signal after the envelop detector with **num=0.136**:

The goal in changing the 'num' variable was to sharpen the envelop detector signal result. We see that the result with num=0.136 is much shaper than the result with num=1 that this is extremely essential for the next part that will be finding the S1 and S2 peaks and through them finding the heartbeat.

1. **Peak\_count :**

This function counts the number of peaks detected and saves the indices of the current peak and the previous peak. It detects a peak when it receives a value greater than a 3 values of thresholds – the upper threshold , the lower threshold and the middle threshold. This value was determined based on a comparison between the values we want to receive (S1 and S2) and the values we want to filter out. Additionally, the function ensures that each peak is counted only once by checking if the current value is greater than the threshold value.

The code of the peak\_count function is : the call of the function is from the last IIR biquad (HPF) – after performing ABS and envelop detection:

1. **time\_peaks()**

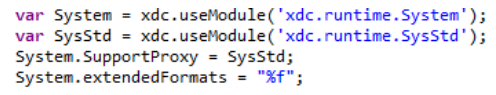
This function calculates the difference between the main beats (S1-S1) and between a main beat and a secondary beat (S1-S2). The difference between two main beats is measured when the current peak is odd and there are at least 3 peaks (i.e., when there are at least two main beats), and the time interval between a main beat and a secondary beat is measured when the current peak is even and there are at least 2 (i.e., there is one main and one secondary beat).

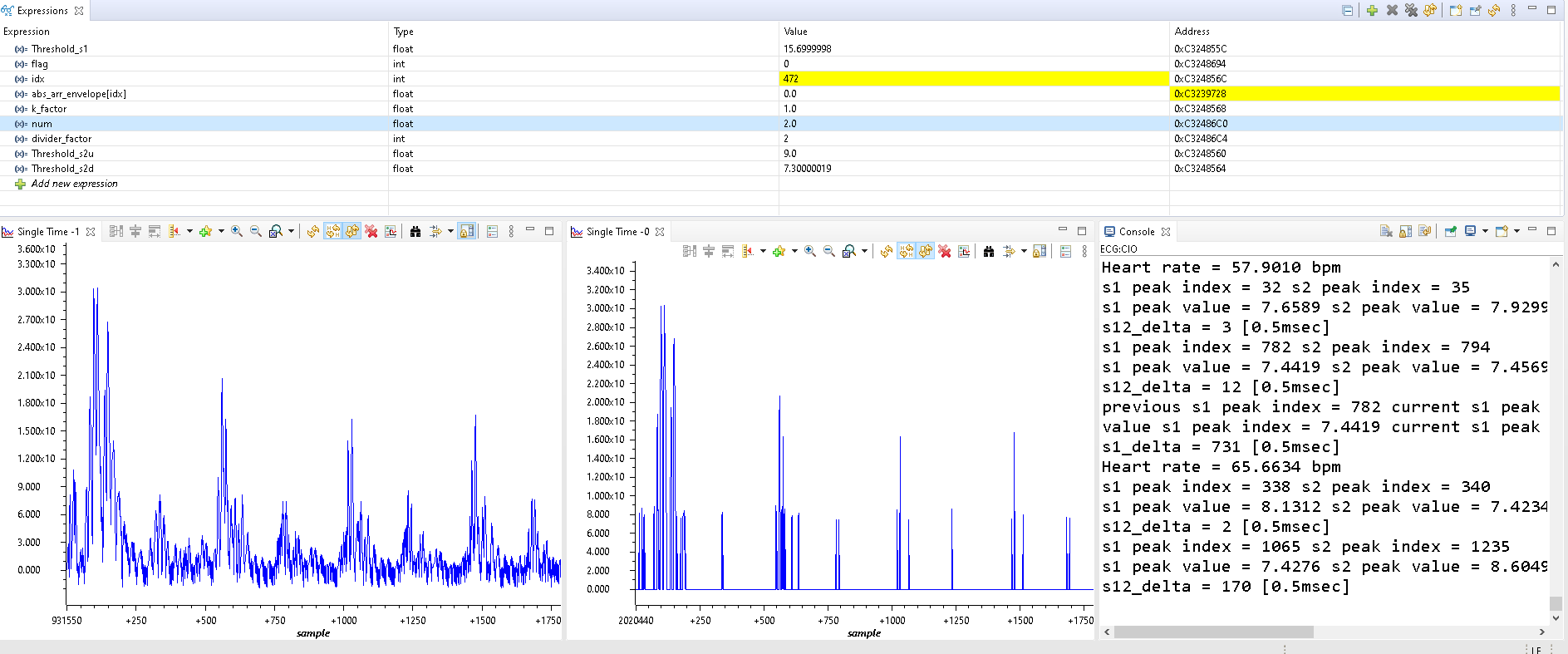
When calculating the time between S1-S2, the value of S1 (peak\_prev\_second) is saved so that the interval between the current S1 and the previous S1 can be calculated. After calculating the time between S1-S1 (delta\_1s), the heart rate in beats per second (bps) is calculated by dividing the value 800 (the number of cells in the array that defines a second) by delta\_1s (the number of cells in the array that defines the time interval between peaks). This ratio describes how many S1-S1 cycle times fit into one second, i.e., the heart rate per second. Additionally, we calculated the heart rate per minute by multiplying the bps by 60. The heart rate, time intervals, and time indices are displayed on the screen.



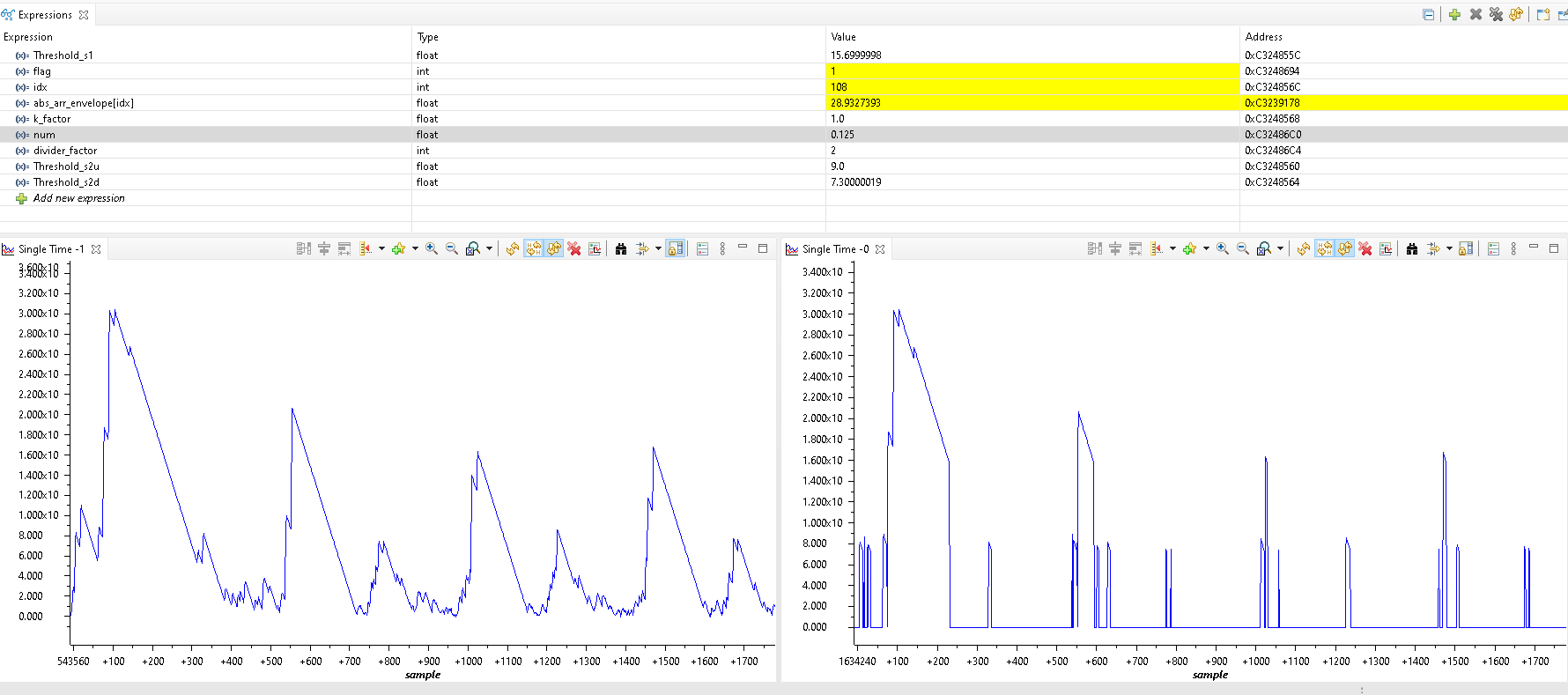
1. **printing to the monitor of the CCS:**

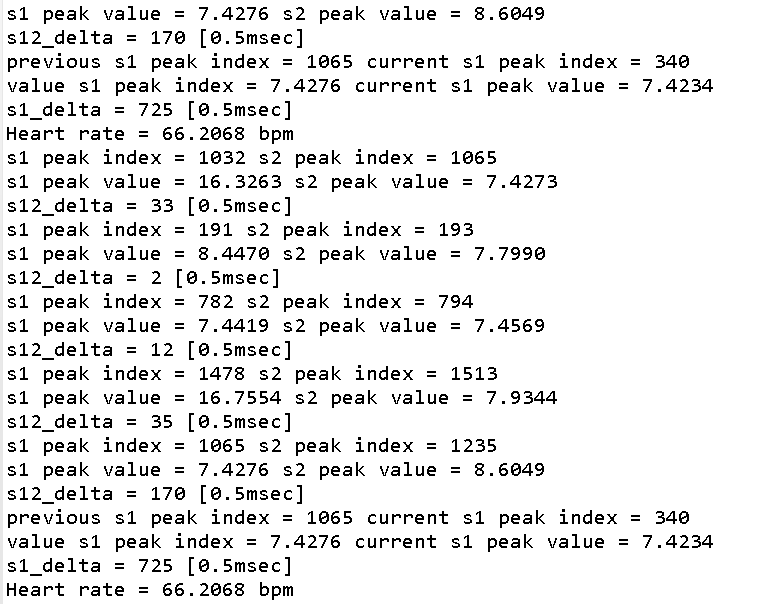
In order to use the **System\_printf** function, including the option to print numbers that are floats, the following lines of code should be added to the ECG.cfg file:



1. **Final running – the variables values and console display**

Running with **num=2** , **threshold\_s1=15.7** , **threshold\_s2u=2** and **threshold\_s2d=9**:

Running with **num=0.125** , **threshold\_s1=15.7** , **threshold\_s2u=2** and **threshold\_s2d=9**:

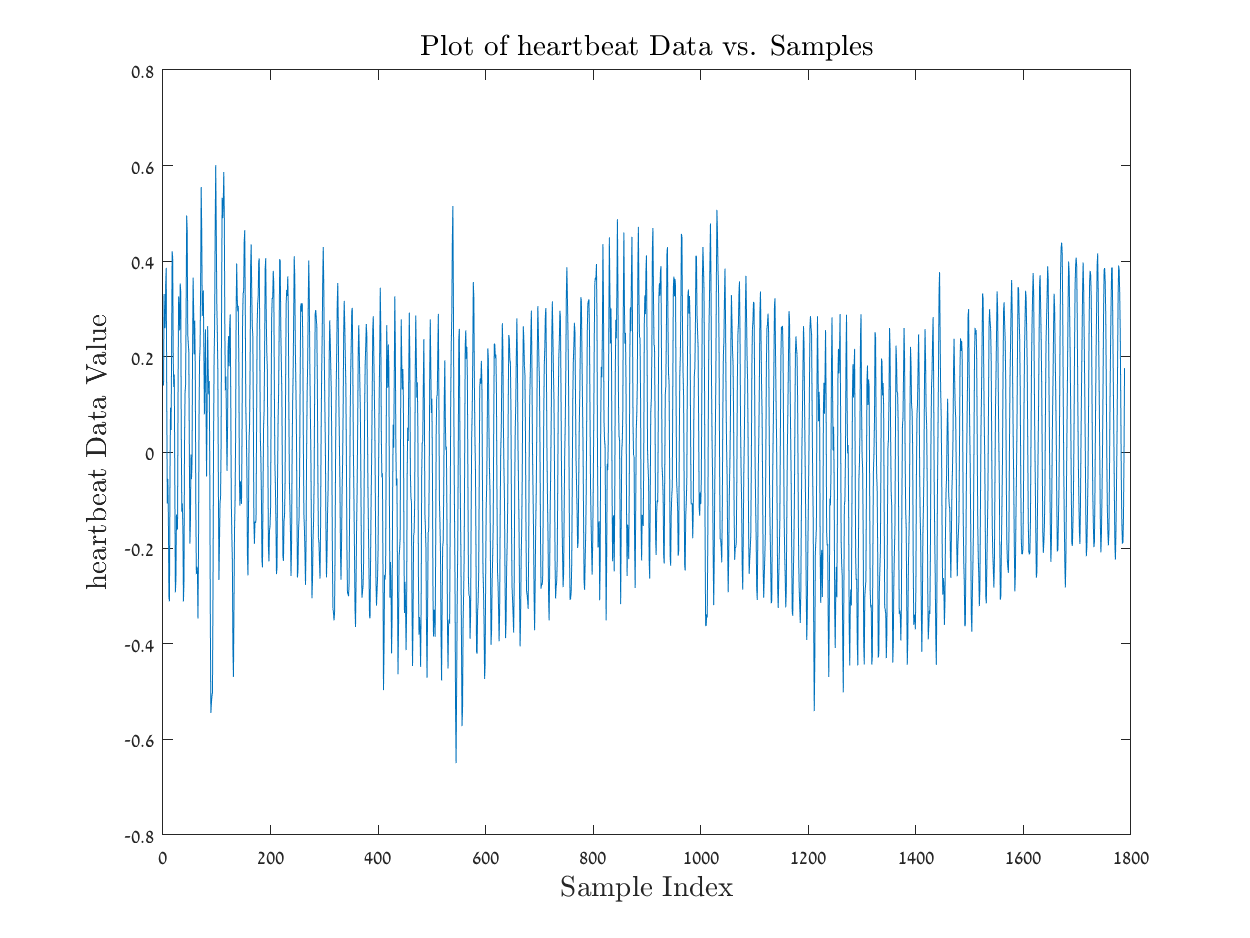


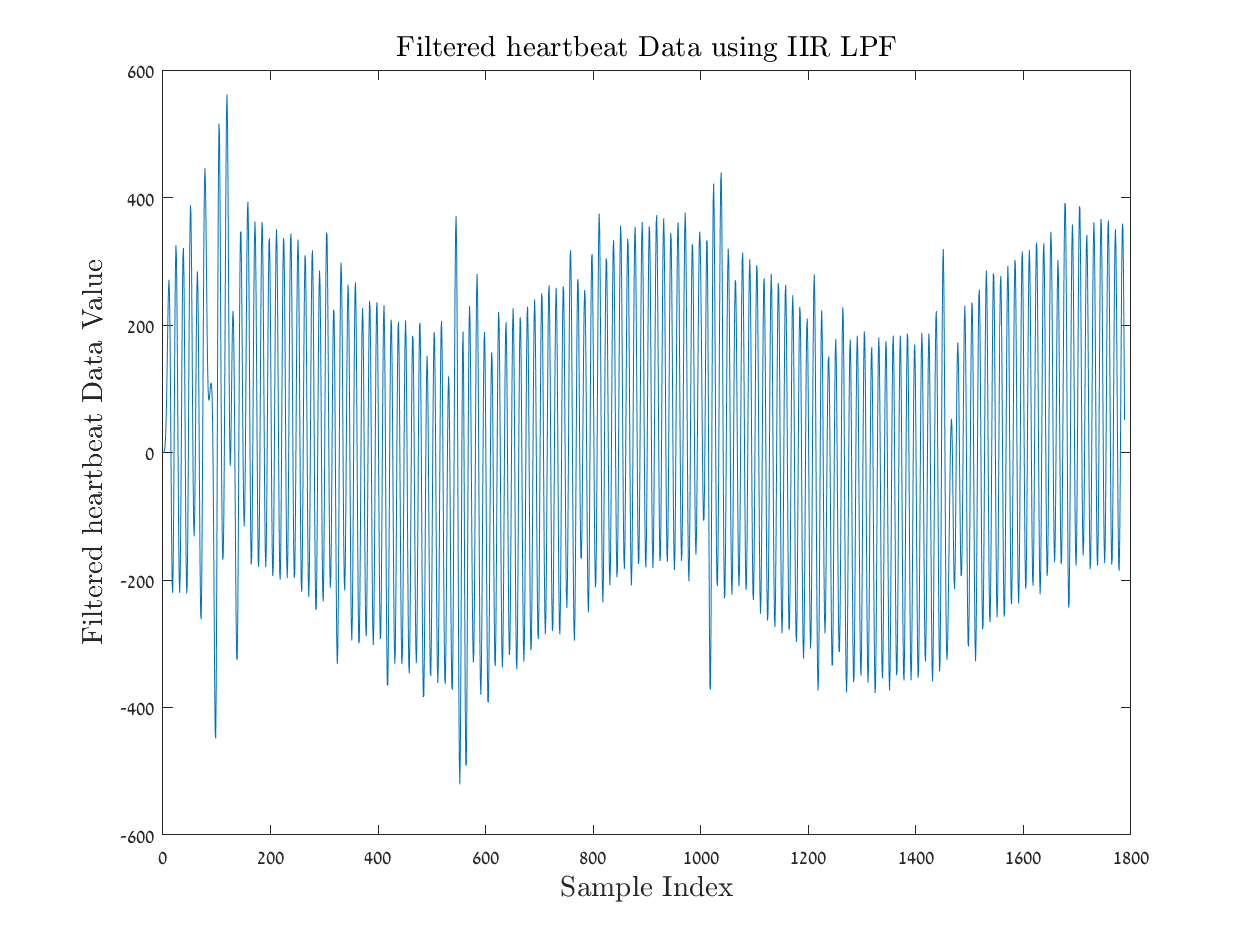
Matlab simulation:

Before starting to apply the given project in the CCS environment , we first and foremost tried to simulate the whole project in a Matlab environment.

We will not present here the full code we used in order to simulate , albeit we will present the results both in time domain and in frequency domain – which can help the reader of that document to understand our thinking and working processes.

The process will be numbered – as our project's process in Matlab to understand the project we were given:

1. 
2. A graph of a person's body

   Description automatically generated with medium confidence
3. 

A graph of a graph

Description automatically generated with medium confidence

**Sources used in that report:**

**https://health.ucdavis.edu/sports-medicine/resources/heart-rate**