Xmax protection of loudspeakers Project Report

Cristian Felipe MANCO OSORIO
Yuyang LIU

International Masters Degree in Electro Acoustics, M2

2023 – 2024

Contents

1 Introduction **1**

2 Objectives **1**
2.1 General Objective . . . 1
2.2 Specific Objectives . . . 1

3 Literature Review **2**

4 Methodology **2**
4.1 Overall Protection Framework . . . 2
4.2 Displacement Estimator . . . 4
4.3 Dynamic High-Pass Filter Implementation . . . 5
4.3.1 High-Pass Filter Setup . . . 5
4.3.2 Cut-Off Frequency Modulation . . . 6
4.4 Definition of Xmax and Margin Value M . . . 7
4.5 Direct Form I Topology . . . 7

5 Experimental Setup **8**
5.1 Transducer Parameters Measurement . . . 8
5.2 Teensy Board . . . 10
5.3 The Teensy code . . . 10
5.3.1 `Bilinear_coeff(Re, Bl, Mms, Kms, Rms, fs)` Function . . . 10
5.3.2 `Operations(void)` Function . . . 10
5.3.3 `LS_Model` Class . . . 10
5.3.4 `DynaFreq(x_est, Xmax)` Function . . . 11
5.3.5 `High_Pass_BW(fc, fs, input)` Function . . . 11
5.4 Gain Amplification Measurement . . . 11
5.5 Measurements Setup . . . 11

6 Results and Analysis **12**
6.1 Evaluation of the Estimated Model . . . 13
6.2 Representation and Analysis of the Protection System Results . . . 14

7 Discussion **16**
7.1 Evaluation of the Protection Mechanism . . . 16
7.2 Limitations of the system . . . 17
7.3 Exploration of 'Adaptive' . . . 18

8 Conclusion **19**

Appendices **20**

A State-Space Model **20**
A.1 Linear State Space Model . . . 20
A.2 Non-Linear State Space Model . . . 21

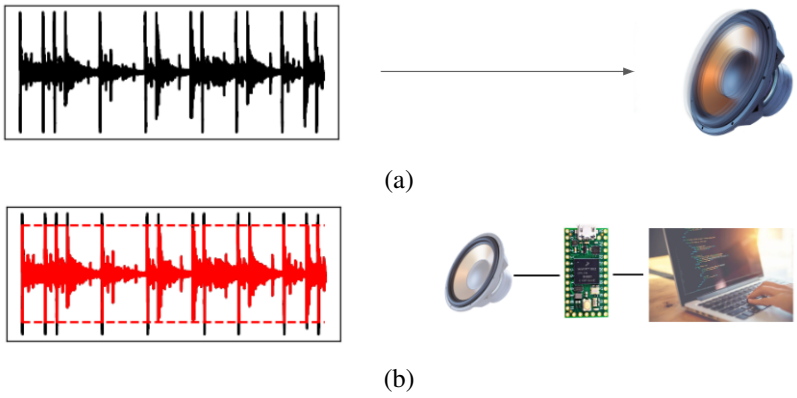
B Peerless HDS-P830860 Datasheet **22**

C Kilpple LSI Mesurement Result **23**

D Arduino Code **23**

1 Introduction

The report focuses on ensuring the longevity and optimal performance of loudspeakers in audio systems, addressing the risks associated with high displacement signals. It evaluates the effectiveness of a protection mechanism in mitigating potential damage by estimating membrane displacements and comparing it with the maximum allowed values like the Xmax, and by doing real-time experiment measurements using various signal types. Figure 1a illustrates a simple example in which a signal (in black) might generate very large displacements causing the membrane to move outside the safety limits, reducing its useful life. The protection mechanism would do something similar to what is shown in Figure 1b, in which a large displacement was detected and so the audio signal is attenuated at those moments to get the signal in red.



(a)

(b)

Figure 1: Example diagrams of displacement signals and vibration sketches of a loudspeaker without a protective mechanism (a), example diagram of the displacement signal with protection device (red line) and sketch of the implementation of the protection device (b).

This study thoroughly examines scenarios with and without the protection system engaged, demonstrating its ability to maintain displacements within safety margins while highlighting differences between estimation and measurement. The discussion section deeply analyzes displacement behavior, cutoff frequency modulation, and observed dynamic trends during signal processing, while also addressing system limitations, such as the impact of environmental factors on system parameters and the instability induced by dynamic high-pass filters.

Additionally, the report explores adaptivity within the protection system, considering adjustable setups for attack and release times to alter system parameters.

Finally, this text aims to provide a complete understanding of the protection system's capabilities, limitations, and potential ways to improve loudspeaker resilience against high displacement signals.

2 Objectives

2.1 General Objective

Design an adaptive digital system that would protect a loudspeaker (woofer) at very low frequencies against excessive displacement in real-time.

2.2 Specific Objectives

- Model a loudspeaker using different methods and evaluate them to choose the most convenient to implement in real-time.

- Measure the displacement and compare it with the estimated one to validate its accuracy, especially in low frequencies.
- Implement an adaptive high-pass filter in real-time whose cut-off frequency is modulated depending on the amplitude of the estimated displacement every time it approaches the Xmax value.

3 Literature Review

There has been a constant search for ways to protect the movement of a loudspeaker from exceeding its maximum displacement. The loudspeaker is a complex multi-physics system, and its motion involves many aspects such as electromagnetism-mechanics-acoustics. In the process of protecting its displacement, each physical model should be consistent with the real changes and the overall logic is clear. Many researchers have explored the criteria and ideas of protection from different aspects.

In the field of audio technology, a common approach is to use **compressor** or **limiter**. **Compressor** and **limiter** have been around since the mid-1900s as recording technology evolved to avoid overloading the signal. **Compressor** serves to reduce the dynamic range of the input signal amplitude. The recorder can increase the amplitude of smaller amplitude signals while reducing larger amplitude signals by setting a series of parameters such as the operating threshold, attack, and release time. The **limiter** can be thought of as a special kind of **compressor**. It can limit the amplitude of a signal below a specified threshold and reduce the amplitude of any over-valued signal. However, this method is only used to control the input signal, independent of the displacement of the loudspeaker, and does not guarantee the effectiveness of the protection. In addition, direct compression of the time-domain signal distorts the output signal.

With the advent and popularization of DSPs, more digital protection circuits are present. In 1996, Wolfgang Klippel introduced the conceptual framework and systematic methodology for the prediction of loudspeaker displacement as a means of loudspeaker protection in his patent [1]. This approach relies upon the input signal. There are generally feedforward and feedback controls to predict the loudspeaker displacement, detect its envelope, and control the input signal according to the detection results. Displacement prediction can be implemented by displacement sensors or can rely on accurate real-time modeling. Two different methods of displacement envelop detection are mentioned here, one uses the Hilbert transform to understand the total displacement envelope but cannot be used in real-time, and another uses time delays to approximately detect it.

In addition to rigorous protection circuitry, the definition of protection thresholds is also important. Klippel has given a definite and meaningful value of Xmax [2], which is closely related to the mechanical structure of loudspeakers. At the same time, he introduced the concept of overload protection of loudspeaker systems from a mechanical point of view in 2016 [3]. The two above research can be combined with digital processing to realize better protections.

4 Methodology

This section explains and illustrates the overall algorithmic framework for Xmax protection of loudspeakers and the methodological and theoretical basis for the various components that make up the system. The algorithm is combined with an estimator, a detector, and a controller using feedback control.

4.1 Overall Protection Framework

This section details the three main components: the estimator, detector, and controller of the algorithm and explains the logical process of the overall protection framework. The flow diagram of the algorithm is

represented in Figure 2, with the three components surrounded by dashed boxes of different colors. Since the process takes place in real-time, the signal is represented sample by sample.

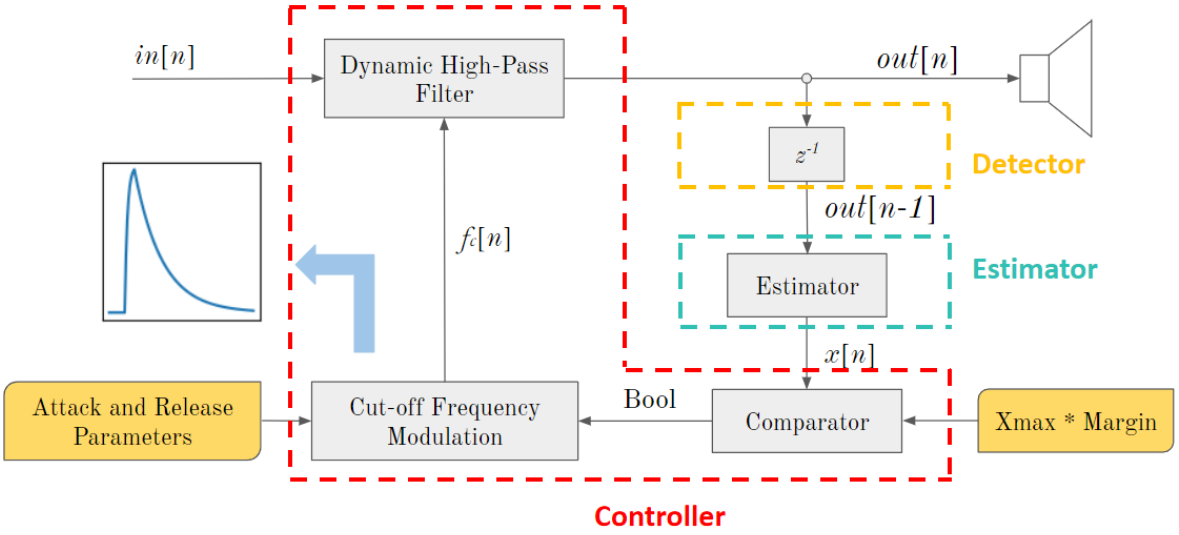


Figure 2: Flow Diagram of the overall protection framework algorithm

Before explaining the details of this process, it is important to note that the transfer function, labeled $H_{XU}(\omega)$, which relates the loudspeaker membrane displacement $X(\omega)$ to the input signal $U(\omega)$, acts like a low-pass filter. This property will be explored further in the section 4.2. This means that for the same amplitude, the lower the input signal frequency range, the greater the amplitude of the loudspeaker's membrane displacement movement. Based on this conclusion, the general idea to protect the loudspeaker is to pass the input signal $in[n]$ through a high-pass filter to cut the low-frequency components and reduce the risk of the loudspeaker membrane exceeding its Xmax.

Nevertheless, this high-pass filter is not static; instead, it is anticipated to dynamically adjust based on the real-time condition of the loudspeaker displacement. The nature of the dynamic of the high-pass filter is the varying cut-off frequency. When the membrane is operated under the threshold **Xmax*Margin**, the cut-off frequency is so low that almost all signals pass through the high-pass filter with the original amplitudes. However, once the membrane displacement exceeds the threshold, the cut-off frequency begins to rise. When the displacement falls below the threshold, the cut-off frequency decreases again. The rate of change of the cut-off frequency modulation depends on the parameter attack time and release time. The principles of cut-off modulation and parameter selection are detailed in Section 4.3.2.

The comparator within the controller is responsible for assessing whether the displacement exceeds the threshold. The other component involved in the comparison is the loudspeaker membrane displacement, computed by the model in the estimator based on the preceding real output signal $out[n-1]$. The selection of the model depends on the operational state of the loudspeaker, a topic explored in detail in Section 4.2.

The detector is used to detect the real-time output signal $out[n]$ of the speaker and issue commands to the controller based on this signal. However, the immediate results reflected by this real-time signal cannot directly influence itself. Therefore, this is why the detector needs to examine previous output signal samples.

In summary, the general process involves the passage of the input signal $in[n]$ through a dynamically adjustable high-pass filter, where the cut-off frequency depends on the outcomes determined by the comparator. The comparator serves the purpose of comparing the threshold magnitude **Xmax*Margin** with the modeled displacement of the loudspeaker based on the preceding output signal $out[n-1]$.

4.2 Displacement Estimator

The displacement estimator consists of an accurate real-time model. This model simulates the displacement of the loudspeaker based on the input signal of the loudspeaker, i.e. the output signal $out[n]$ of the protection system. The accuracy of the model is crucial as it determines whether the protection mechanism operates appropriately and in time.

The key to ensuring accuracy is the choice of model and the accuracy of parameter values. The measurement of the parameters is detailed in Section 5. In terms of the model, there are two options: the transfer function filter model and the state space model. The filter model uses the linear loudspeaker motion equations to derive the transfer function between its displacement and the input voltage signal and regards it as an analog filter. This is then discretized into a digital infinite impulse response (IIR) filter that can operate in real-time. The state space model is a direct discretization of the polynomial equations to derive the state of each sample based on the previous state of the variables.

Both of these models come with their strengths and limitations in practical application. The filter model can neither take into account eddy currents generated by electromagnetic components nor their nonlinear characteristics. However, it offers the advantage of simplicity and ease of implementation on the Teensy board in C++ language as they can utilize the same topology as the dynamic high-pass filter. On the other hand, the state-space model can simulate analog eddy currents in real-time using models like **R2L2** and can also incorporate nonlinear characteristics through implicit or explicit methods [4]. However, factoring in these aspects introduces more parameters and equations, consequently increasing the code complexity.

This project is focused on researching and designing a functional protective demonstrator rather than a mature product. The protection mechanism primarily functions in the low-frequency range, as demonstrated later, with the loudspeaker predominantly handling small signals in most cases. Hence, utilizing a linear model and disregarding eddy currents is sufficient. As a result, a filter model was employed for the estimator in both simulation and practical applications throughout the project. The operational principle is illustrated below, while the details of the state-space model are provided in Appendix A.

The equations that characterize the motion of loudspeakers in the electrical and mechanical domains are expressed in Equations 1 respectively. Here, **Re** and **Le** denote resistance and inductance, while **Mms**, **Rms**, and **Cms** represent the moving mass of the loudspeaker incorporating the acoustic radiation mass, the mechanical resistance including the acoustic radiation resistance, and the mechanical compliance, respectively. The variables U, I, and X represent the voltage at the end of the voice coil, the current flowing through the voice coil, and the membrane displacement, respectively.

$$\begin{cases} U = \mathbf{R_e} I + j\omega \mathbf{L_e} I + j\omega \mathbf{Bl} X, \\ \mathbf{Bl} I = -\omega^2 \mathbf{M_{ms}} X + j\omega \mathbf{R_{ms}} X + \frac{X}{\mathbf{C_{ms}}} \end{cases}. \tag{1}$$

In the low-frequency range, the component $j\omega\mathbf{L_e}$ can be neglected. By replacing the current by the displacement, the transfer function between the loudspeaker membrane displacement $X(\omega)$ and the input signal $U(\omega)$ is shown in Equation (2), where $\mathbf{A}_{XU} = \frac{\mathbf{B}\ell\mathbf{C_{ms}}}{\mathbf{R_e}}$, $\omega_0 = \sqrt{\frac{1}{\mathbf{M_{ms}C_{ms}}}}$ and $Q_0 = \frac{1}{\omega_0 \mathbf{R'_{ms}C_{ms}}}$. $\mathbf{R'_{ms}}$ represents the total resistance in the mechanical domain, which equals to $\mathbf{R_{ms}} + \frac{\mathbf{B}\ell^2}{\mathbf{R_e}}$.

$$H_{XU} = \frac{X(\omega)}{U(\omega)} = \mathbf{A}_{XU} \frac{1}{1 + j\frac{\omega}{\omega_0 Q_0} - \left(\frac{\omega}{\omega_0}\right)^2}. \tag{2}$$

This expression shows that the transfer function is a second-order low-pass filter with a cut-off frequency of $f_0 = \frac{\omega_0}{2\pi}$. The general expression of the second-order filter in the Laplace domain is shown in Equation (3), where $s = j\omega$. The transfer function H_{XU} can also be shown by this expression with $B_0 = B_1 = 0$,

$B_2 = \mathbf{A}_{XU}$, $A_0 = \frac{1}{\omega_0^2}$, $A_1 = \frac{1}{\omega_0 Q_0}$ and $A_2 = 1$.

$$H(s) = \frac{B_0 s^2 + B_1 s + B_2}{A_0 s^2 + A_1 s + A_2}. \tag{3}$$

To implement the filter in real-time, it is necessary to discretize the equations from the Laplace domain to the Z domain. In this project, the bilinear discretization method is employed, and the relationship between s and z is depicted in Equation (4), where f_s is the sampling rate of the system.

$$s = 2f_s \frac{z-1}{z+1}. \tag{4}$$

Upon substituting s with z, the general expression for the second-order digital filter is provided in Equation (5). According to the properties of the Digital Time Fourier Transform (DTFT), considering the transfer function $H_{d_{XU}}(z)$ as an example, the calculation of the output signal displacement $x[n]$ in the time domain can be expressed in terms of the input voltage signal $u[n]$ and the preceding samples. This relationship is detailed in Equation (6). Typically, the value of a_0 is set equal to 1. Knowing the values of the coefficients of the digital filter, it can be implemented into the Teensy board in C++ with the Direct Form I topology, as described in Section 4.5.

$$H_d(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}}. \tag{5}$$

$$a_0 x[n] = b_0 u[n] + b_1 u[n-1] + b_2 u[n-2] - a_1 x[n-1] - a_2 x[n-2]. \tag{6}$$

4.3 Dynamic High-Pass Filter Implementation

The controller's dynamic high-pass filter is used to limit the passage of low-frequency, high-amplitude input signals. This section describes the general setup and cut-off frequency modulation configurations of the high-pass filter, particularly the functions and selection principles for the attack and release time.

4.3.1 High-Pass Filter Setup

Two parameters, order, and type, are crucial in determining the characteristics of the high-pass filter. In this project, a second-order Butterworth high-pass filter is selected. The analog filter function of this high-pass filter is given in Equation (7), where ω_c is the angular cut-off frequency of the high-pass filter. After determining the cut-off frequency value, the coefficients of the filter are established, aligning with the general second-order filter equation given in Equation (3).

$$H_{BW}(s) = \frac{s^2}{s^2 + s\sqrt{2}\omega_c + \omega_c^2}. \tag{7}$$

The analog filter can undergo the same discretization method - bilinear discretization as the filter model. However, to avoid errors in the high-frequency range, frequency warping is employed in this context [5], which is carried out to correct the cut-off frequency so that the digital filter matches with the analog filter in the high-frequency. Equation (8) shows the expression of the corrected cut-off frequency ω_d. The digital filter coefficients in the Teensy board are calculated using frequency warping techniques.

$$\omega_d = 2f_s tan\left(\frac{\omega_c}{2f_s}\right). \tag{8}$$

4.3.2 Cut-Off Frequency Modulation

The dynamic of the high-pass filter is influenced by the variation of the cut-off frequency. Specifically, the cut-off frequency increases when the estimated displacement surpasses the threshold, and conversely, it decreases when the estimated displacement is below the threshold. This section aims to elucidate the modulation of the cut-off frequency: the function it follows for upward or downward adjustments and the impact of parameters, such as attack and release time, on the outcomes.

There are key parameters that define the variation of the cut-off frequency: the maximum cut-off frequency $f_{c_{max}}$, the minimum cut-off frequency $f_{c_{min}}$, the attack time τ_a, and the release time τ_r. The cut-off frequency undergoes modulation within the range of $[f_{c_{min}}, f_{c_{max}}]$, taking the attack time to transition from $f_{c_{min}}$ to close to $f_{c_{max}}$, and conversely, the same for the release time.

The modulation function can be any monotonically increasing or decreasing function, as precisely explained in the study by Clément Richard [6]. The intuition of the principle of choice is to have a short attack time and a long release time, ensuring that loudspeaker displacements exceeding the threshold are promptly reflected for protection while returning gradually to $f_{c_{min}}$ to prevent the occurrence of multiple consecutive pulse signals.

However, in some cases, too short an attack or release time may result in increased distortion. This occurs because the time-varying filter may not adequately react in time, resulting in a significant error. Also, the long release time lets the high-pass filter keep cutting the low-frequency components, which continuously degrades the sound quality. Hence, the selection of the attack time and release time depends on the signal type. In the project experiments, appropriate values for τ_a and τ_r were chosen to align with the characteristics of the test signals. Nevertheless, further research is required on algorithms that aim for adaptive adjustments in τ_a and τ_r.

The modulation function used in this project is analogous to the voltage signals of the components in the **LC** circuit, which is a kind of exponential function. The general functions are represented in Equation (9), where e^{-1} is a key limiting value which is the rate of change of the attack and release time control.

$$\begin{cases} \text{Increase function: } f_c = f_{c_{min}} + (f_{c_{max}} - f_{c_{min}}) \left(1 - e^{-\frac{t}{\tau_a}}\right) \\ \\ \text{Decrease function: } f_c = f_{c_{min}} + (f_{c_{max}} - f_{c_{min}})\, e^{-\frac{t}{\tau_r}}. \end{cases} \tag{9}$$

In the digital domain, changes occur on a sample-by-sample basis. The attack and release time are substituted with the attack number $N_a = f_s \times \tau_a$ and release number $N_r = f_s \times \tau_a$. At each instant, the cut-off frequency value changes $e^{-\frac{1}{N_a}}$ for attack or $e^{-\frac{1}{N_r}}$ for release. The computational formulas employed in the Teensy board are illustrated in Equation (10).

$$\begin{cases} \text{If x>threshold: } f_c[n] = \left(\left(\frac{f_c[n-1]-f_{c_{min}}}{f_{c_{max}}-f_{c_{min}}} - 1\right) \times e^{-\frac{1}{N_a}} + 1\right) * (f_{c_{max}} - f_{c_{min}}) + f_{c_{min}} \\ \\ \text{If x<threshold: } f_c[n] = \frac{f_c[n-1]-f_{c_{min}}}{f_{c_{max}}-f_{c_{min}}} \times e^{-\frac{1}{N_r}} * (f_{c_{max}} - f_{c_{min}}) + f_{c_{min}}. \end{cases} \tag{10}$$

In Figure 3, an illustrative example of the cut-off frequency modulation function is presented, with parameters set as $f_{c_{max}} = 200Hz$, $f_{c_{min}} = 5Hz$, $\tau_a = 0.05s$, and $\tau_r = 0.2s$. The blue dashed line represents the estimated loudspeaker displacement signal, while the orange dashed line signifies the threshold. The green line corresponds to the cut-off frequency. It is evident that each time the displacement surpasses the threshold, f_c increases rapidly, and it gradually decreases when the displacement decreases, aligning with the intended behavior.

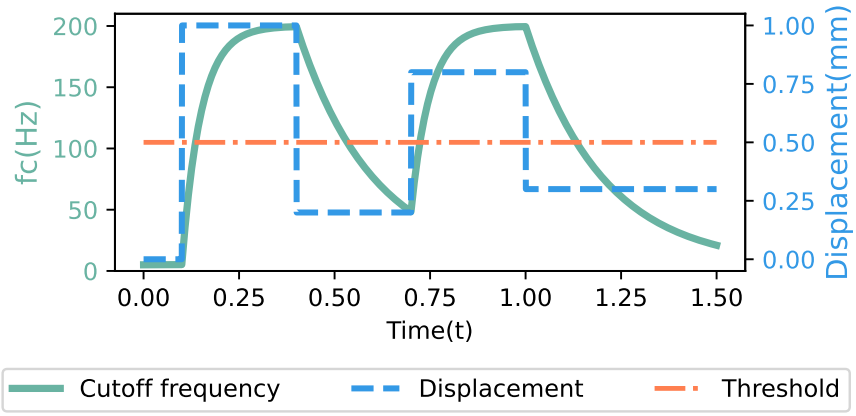


Figure 3: Example of cut-off modulation where the displacement varies with time above or below the threshold, where the attack time is 0.05 seconds and the release time is 0.2 seconds.

4.4 Definition of Xmax and Margin Value M

Typically, the maximum displacement of a loudspeaker is dictated by the mechanical properties of its materials. The manufacturer specifies the Xmax value on the datasheet when the loudspeaker is made. Theoretically, this value serves as the threshold for determining whether the protection system is operational or not. However, as the protection system requires a certain amount of time to engage, using this value directly as a judgment threshold would be too delayed for effective protection. Therefore, it is essential to define a safety margin.

The precise definition of the safety margin is elaborated in Klippel's document [3]. The primary objective of this project is to finalize the Xmax protection demonstrator, and extensive additional research on the margin is not required. Additionally, the allowed maximum input voltage of the acquisition board is 10 V, and the loudspeaker does not reach Xmax at this voltage. Therefore, moderate Xmax and margin values were selected for the project, with $Xmax$ =1.5mm and $M = 0.7$.

4.5 Direct Form I Topology

An appropriate topological approach is essential when employing digital filters in a real-time MCU board or programming language. In the algorithm, the displacement estimation model and the dynamic high-pass filter are digital filters, and the results of the filters are described in the above sections.

The computation process of the digital filter in real-time follows Equation (6). There are four typical topologies for implementing the filter: **Direct Form I**, **Direct Form II**, **Transposed Direct Form I**, and **Transposed Direct Form II**. The project ultimately adopts the topology which is optimal for the varying coefficients: direct form I topology. Due to the time-varying coefficients of the dynamic high-pass filter, instability may occur because of the float of the previous sample's value. Hence, a topology that maximizes the retention of fixed points is necessary.

Taking the model filter H_{XU} as an example, the schematic diagram of the direct form I topology is illustrated in Figure 4, where a_0 is set to 1, and the values of the remaining coefficients adjust accordingly. Four temporary variables, d_0, d_1, d_2, and d_3, are employed to store the values of $u[n-1]$, $u[n-2]$, $x[n-1]$, and $x[n-2]$, respectively. The following C++ code implements this topology, where u and x represent the values at moment n, seen as $u[n]$ and $x[n]$, respectively.

```
volatile float d0, d1, d2, d3;
''' At sample n '''
x = b0 * u + b1 * d0 + b2 * d1 - a1 * d2 - a2 * d3;
```

```
d1 = d0;
d0 = u;
d3 = d2;
d2 = x;
```

Listing 1: Example code of the direct form I in C language.

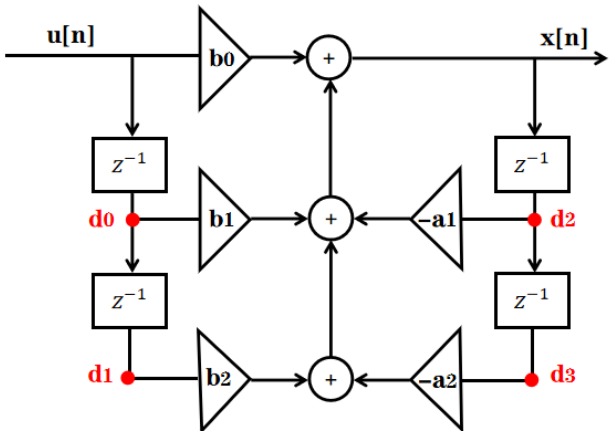


Figure 4: Schematic diagram of the direct form I topology of the digital filter.

5 Experimental Setup

5.1 Transducer Parameters Measurement

The device under test is a woofer Peerless HDS-P830860 (Appendix B) which is a 133 mm transducer, initially, it was measured in the Klippel system with the Linear Parameter Measurement (LPM) [7] and the Large Signal Identification (LSI) [8] modules that are included dB-Lab Software to get the loudspeaker parameters under linear and no-linear behavior.

The linear loudspeaker parameters estimated from this set of measurements are the ones used on the Teensy board algorithm to estimate the a and b coefficients of the displacement estimator according to Equation (6). These parameters are shown in Table 1.

$\mathbf{B\ell}$	$\mathbf{R_e}$	$\mathbf{L_e}$	$\mathbf{K_{ms}}$	$\mathbf{M_{ms}}$	$\mathbf{R_{ms}}$
5.816	6.76 Ω	0.29 mH	1800 N/m	9.257 g	1.866 Ns/m

Table 1: Transducer Parameters estimated with the Klippel system in LPM mode.

To verify these parameters, it was necessary to measure the loudspeaker response and then compare its actual response with a loudspeaker model using the parameters from Table 1 to see how accurate they are. The setup for this measurement is shown in Figure 5 where three signals were captured: Voltage U_R on the 1 Ω reference resistor, voltage U_{HP} on the loudspeaker, and the voltage given by the displacement sensor. This sensor is a Panasonic HG-C1030-P with a sensitivity of 2 mm/V and latency of 1 ms approximately, and it was used to measure the membrane displacement, note that this sensor has a frequency range of up to 500 Hz. The signal used for measuring the loudspeaker response was a synchronized swept sine from 10 Hz to 20 kHz. Additionally, the device used to reproduce the testing signal and capture the three measured signals was a data acquisition module NI USB-4431. From these measurements, the current and the displacement are shown in Figure 6.

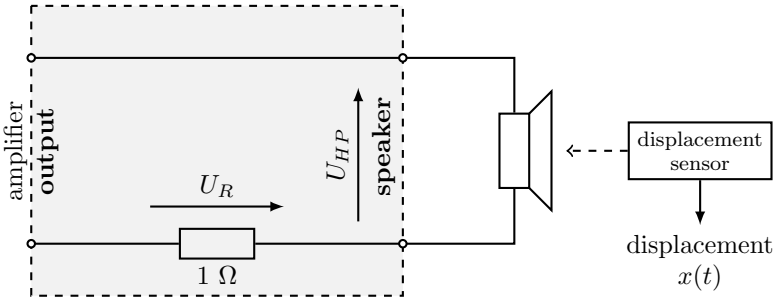


Figure 5: Setup used for measuring the loudspeaker response.

For comparing the measured displacement, and also for verifying the loudspeaker parameters, different models were computed using these parameters: Linear model using IIR filters with bilinear discretization, and state space Euler Forward method, linear and no linear, with and without considering eddy currents respectively (Appendix A). For considering eddy currents, the R2L2 method was implemented, the R2 and L2 values are shown in Table 2 these values were obtained in the LPM results with the Klippel system. The offline simulations are depicted in Figure 6, from where it can be seen that the three models coincide at low frequencies between them but there are some discrepancies with the actual measurement.

$\mathbf{R_2}$	$\mathbf{L_2}$	$\mathbf{R_3}$	$\mathbf{L_3}$
1.39 Ω	0.28 mH	6.96 Ω	0.13 mH

Table 2: Resistor and inductor values for the R2L2 model.

Since the critical frequencies to control are the low frequencies below the resonance (where displacement is considerable), and since the linear and nonlinear models are very close within this frequency range, the implementation of the linear model using the transfer function as described in Section 4.2 is a good approach for the displacement estimation method.

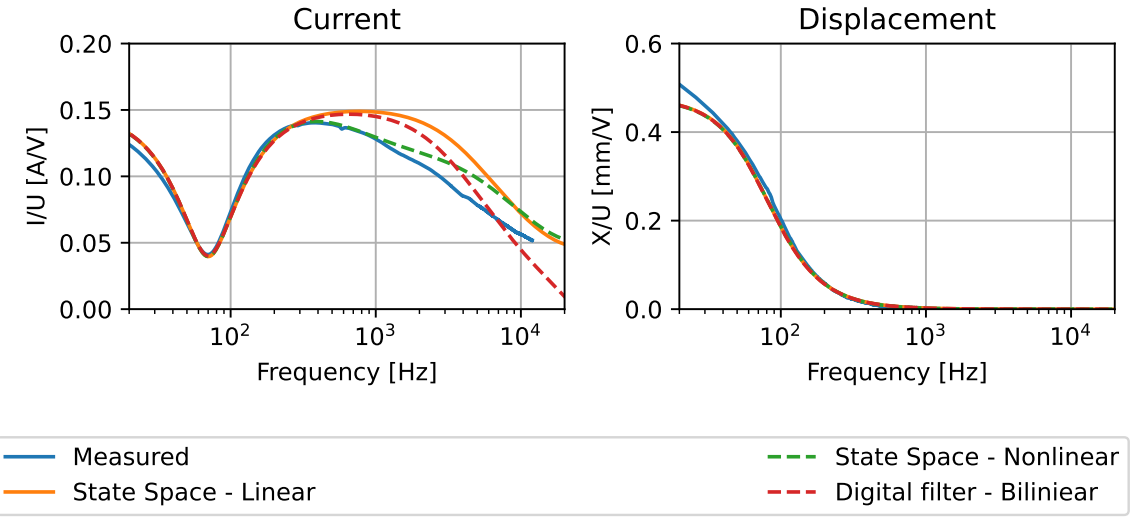


Figure 6: State space and IIR filter model simulation of the current and displacement of the loudspeaker. In blue is the actual measurement, in orange is the state space linear model, in green is the state space nonlinear model considering $\mathbf{B}\ell(x)$, $\mathbf{L_e}(x)$, $\mathbf{K_{ms}}$, and R2L2 for eddy currents, and in red is the IIR filter model.

The main differences are very visible at high frequencies when not taking into account the effect of eddy

currents, where the current tends to be higher than the actual measurement. Another difference is present at very low frequencies, where the current tends to be higher, and the displacement lower than the actual measurement. This phenomenon is caused by the process 'Creep effect', which means that the material undergoes deformation when it moves in the low-frequency range. However, because of the goal of this project, it will be considered that these models are just as good as they can get, and trying to improve them could involve more complex analysis that could be part of a continuation of this study.

5.2 Teensy Board

The microcontroller used for processing the audio is a Teensy 3.6 development board in conjunction with a MAX5717 DAC through SPI communication protocol to allow having a 16-bit audio resolution at the main output, since the DACs integrated in the Teensy board only have up to 12-bit resolution. Two analog outputs with this 12-bit DAC were utilized to output the displacement estimated by the implemented algorithm and for measuring and visualizing the change in the cut-off frequency of the adaptive filter, this is only for monitoring and validation purposes.

5.3 The Teensy code

The code implemented in the Teensy board can be seen in Appendix D. Here only the main sections of the code are going to be described.

5.3.1 `Bilinear_coeff(Re, Bl, Mms, Kms, Rms, fs)` Function

This function can be found in **line 365** in the main code and is implemented once in the `void setup()` method to retrieve the discretized filter a and b coefficients to model the loudspeaker behavior. Once it is called, the coefficients are calculated and stored in global variables to be used later in the code.

5.3.2 `Operations(void)` Function

This is the main function, found in **line 204**, since all of the other methods are called here every cycle loop. After managing the SPI communication, the input value of the ADC is read and stored in the variable `input1`. With this value, the loudspeaker model filter coefficients, and the amplification gain, the displacement estimator can be implemented thanks to the method `x_predictor()` in the class `LS_Model`. With the displacement estimation, the cut-off frequency of the adaptive filter is updated depending on the displacement amplitude by calling the function `DynaFreq(x_est, Xmax)`, which is used in the next step to call the function `High_Pass_BW(fc, sampleRateHz, input1)`, which finally applies the high-pass filter to the signal to protect the membrane against wide displacements. Next in this function is simply updating the analog outputs of the DAC outputs of the Teesny + SPI DAC MAX5717 board, that is, the main audio output with the processed signal, the estimated displacement in the Aux1 output, and the modulated cut-off frequency in the Aux2 output.

5.3.3 `LS_Model` Class

This class is saved into a **.cpp** file and has the method `x_predictor(b_0, b_1, b_2, a_1, a_2, input1, Gain)` for estimating the membrane displacement. It was coded in a class form since it can make it easier to compute independently the displacement of different signals, in this case, it was used to compute the displacement signal without any modification, and the displacement signal as the protection system was controlling it, but it will be useful for working with multiple channels for instance. The function has an IIR

filter structure (direct form I) that applies the linear response of the loudspeaker displacement to the signal using the a and b coefficients from its arguments.

5.3.4 `DynaFreq(x_est, Xmax)` Function

This function takes the estimated displacement signal `x_est` and compares it to the minimum safety value, which in this case is 70% of the Xmax value. The cut-off frequency is modulated with the method mentioned in section 4.3.2. It can be found in **line 403** of the main code.

5.3.5 `High_Pass_BW(fc, fs, input)` Function

This function applies a second-order Butterworth high-pass filter with frequency-warping with a direct form I topology to the audio input signal, in the main code it is found in the **line 419**. Its cut-off frequency can change depending on the amplitude of the estimated displacement thanks to the `DynaFreq()` function, so this filter will be constantly adapting to filter out the low frequencies with high amplitudes to avoid high displacements on the membrane.

5.4 Gain Amplification Measurement

Given that between the audio signal at the main output of the Teensy board, and the loudspeaker there is a gain stage due to the amplifier, this gain level has to be taken into account inside the estimator model to match both amplitudes, the real and estimated one. To do this, a signal generator was used to obtain a sine signal with 1 Vrms that was passed through the whole system (Teensy + amplifier) and then measured at the loudspeaker terminals with an oscilloscope, then the amplifier gain knob was manually adjusted until a voltage of 8 Vrms, meaning that the gain was 8, this gain value is applied to the input signal just before applying the displacement estimator.

5.5 Measurements Setup

The testing and measurements of the system were done by implementing the setup shown in Figure 7 using a data acquisition module **NI USB-4431**, a **Panasonic HG-C1030-P** sensor displacement, and a **Canford Compact Power Amplifier**. From the computer, different signals were reproduced through the acquisition board's output AO0 and sent directly to the Teensy input 1, the main two signals that were studied were a Reversed Synchronized Swept Sine and drum kit loop, some other signals were tested like pure sines or more general signals like music, but they are not analyzed here, nevertheless, in the conclusion section there is a link to a video where it can be shown the system working with different types of signals.

The input connections of the acquisition board are as follows: Input AI0 was used to obtain the data voltage of the loudspeaker voice coil, input AI1 to get the output of the displacement sensor, input AI2 to capture the modeled displacement, and AI3 to obtain the cut-off frequency changes. Bear in mind that the output of the Teensy board is a signal that goes from 0 V to 3.3 V so it has to be shifted by -1.65 V to be properly represented when doing plot comparisons.

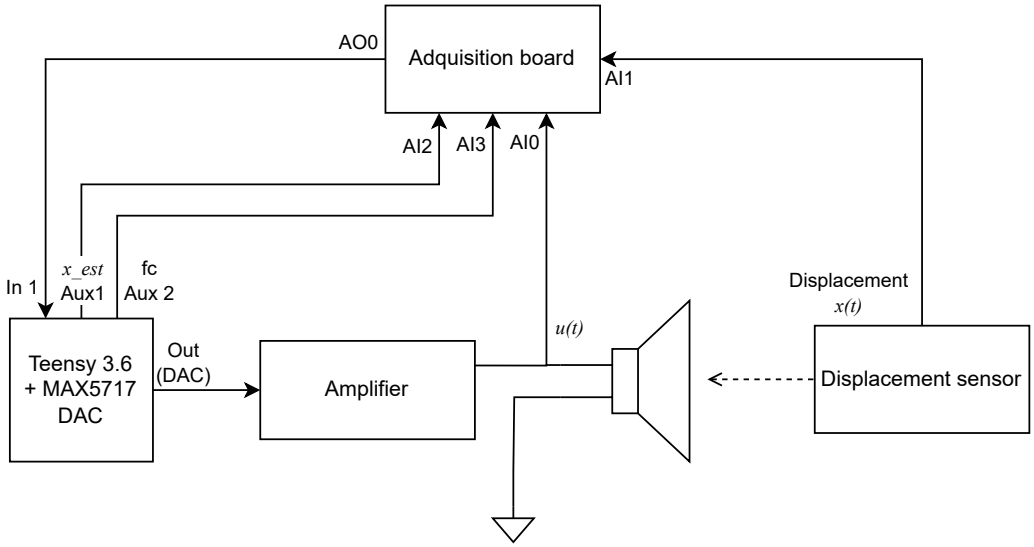


Figure 7: Setup used for measuring the system.

Figure 8 is an image of the physical setup of the system without the acquisition board, just as a customer user would use it: the Teensy controller with an audio input, the output connected to the amplifier, and finally the loudspeaker.

Figure 8: Simplest setup of the system with one audio input channel and one output channel connected to an amplifier to drive the loudspeaker

6 Results and Analysis

With the experimental setup explained in the previous section, the simulation and experimental results have been conducted and analyzed. As mentioned before, two main signals are investigated in this study: a **Reversed Synchronized Swept Sine** (Rsss) ranging from 1 kHz to 20 Hz over 5 seconds, and **Drum Kit Loop**. These signals represent two typical types. The first exemplifies a slowly and gradually increasing displacement signal, while the second shows signals with fast impulse displacements. This classification is essential for investigating the parameters of interest in this project, specifically the rate at which signals exceed the safety margin. In this section, the adaptive Arduino code setup for the attack and release time mentioned in Section 5.2 demonstrates the operational protection processes. The attack and release time values for the two signals are detailed in Table 3.

In this section, we first discuss the validation of the estimated model for both signals to ensure the effec-

	Rsss	Drum Kit Loops
Attack Time τ_a [s]	0.1	0.05
Release Time τ_r [s]	0.1	0.05

Table 3: Attack and release time setup for the Rsss and Drum Kit Loop signals, respectively.

tiveness of the protection system. Subsequently, we present and analyze the results from both simulation and measurement of the protected displacement signals, and talk about the errors between them.

6.1 Evaluation of the Estimated Model

The estimated and measured displacement results for the two signals are depicted in Figure 9 and Figure 10, respectively. The Teensy Board computes the estimated results, while the measurement results are obtained using the displacement sensor. These data are recorded into the input channels **AI2** and **AI1** of Figure 7, respectively. The experiments were performed with the protection mechanism turned off and only the estimator operating.

In both cases, differences are observed between the estimated and measured results. The results for Rsss show that the peak-to-peak amplitude of the estimated displacement aligns well with the measured one in the high and middle-frequency range but is larger in the low-frequency. This occurrence arises since the frequency response of the amplifier is not perfectly flat. However, this difference proves advantageous for the protection system as it allows the controller to detect warnings of high displacement earlier. On the other hand, the measured displacement result is asymmetrical, unlike the estimated one. This asymmetry arises from the absence of non-linear parameters in our estimator model, the measurement curves of which are presented in Appendix C. Additionally, it is possible that the offset situation of the rest displacement X_{rest}, as explained in Klippel's document[2], contributes to this phenomenon.

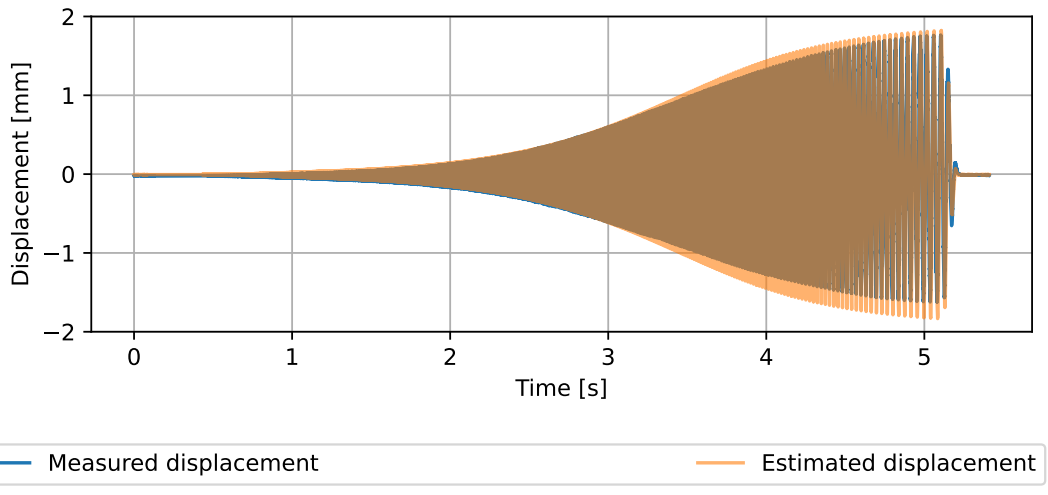


Figure 9: Comparison between the actual displacement and the estimated one using a reversed swept sine signal without Xmax protection in real-time .

Similar phenomena are observed in the case of the drum loop signals, where the estimated displacement covers the measured results and has the larger peak-to-peak amplitude, triggering necessary precautions. These results prove the effectiveness of our estimator model, ensuring the successful implementation of the protection system.

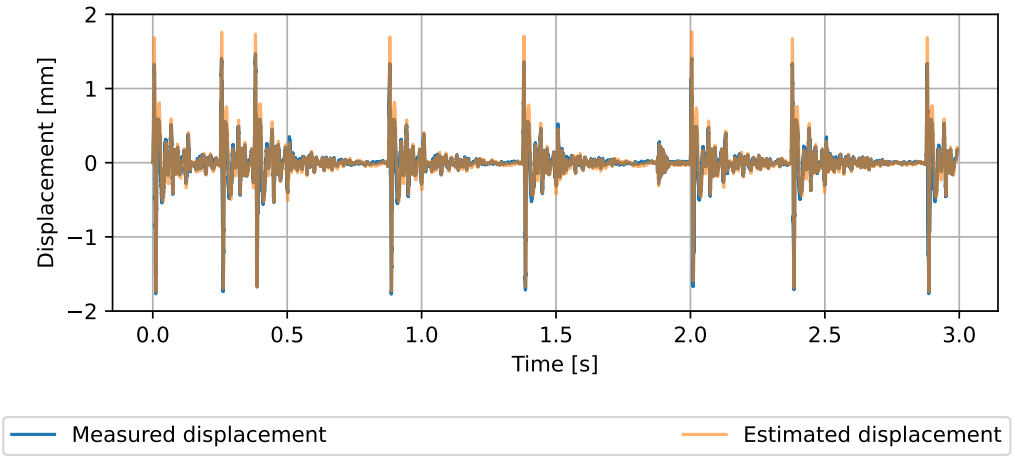


Figure 10: Comparison between the actual displacement and the estimated one using a drum kit loop signal without Xmax protection in real-time .

6.2 Representation and Analysis of the Protection System Results

The estimated and measured results of the protected displacement for both signals are depicted in Figure 11 and Figure 12, respectively. The recording processes are the same as the last subsection but with the protection system turned on. The Xmax value and the safety margin Xmax*M are also represented on the graphs, which evaluate the quality of the protection.

Both the estimated and measured protected Rsss displacement work well. The displacements are almost limited inside the safety margin. Due to the precautions mentioned in the last sections, the measured displacement works better. It is worth noting that the DC offset appears in both estimated and measured results for the Rsss signal, especially at the moment the protection mechanism starts working. This is because of the instability of the dynamic high-pass filter, which is mentioned in Subsection 4.5.

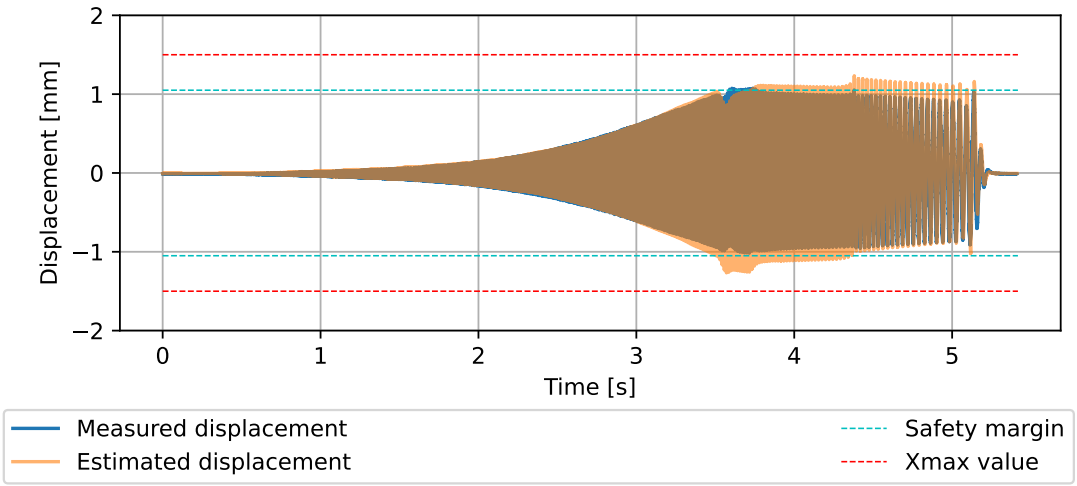


Figure 11: Comparison between the actual displacement and the estimated one using a reversed swept sine signal with Xmax protection in real-time.

Figure 12 illustrates the outcomes of the drum loop with the protection engaged. In general, it can be observed that they do not react as swiftly as the Rsss system. Both the estimated and measured results surpass

the safety margin, but they are nearly within the range of the Xmax. This implies that the protection system can also function for rapidly changing signals but is less effective compared to slower signals. Similar to the Rsss result, the measurement results outperform the estimated ones because of the higher estimated peak-to-peak amplitude.

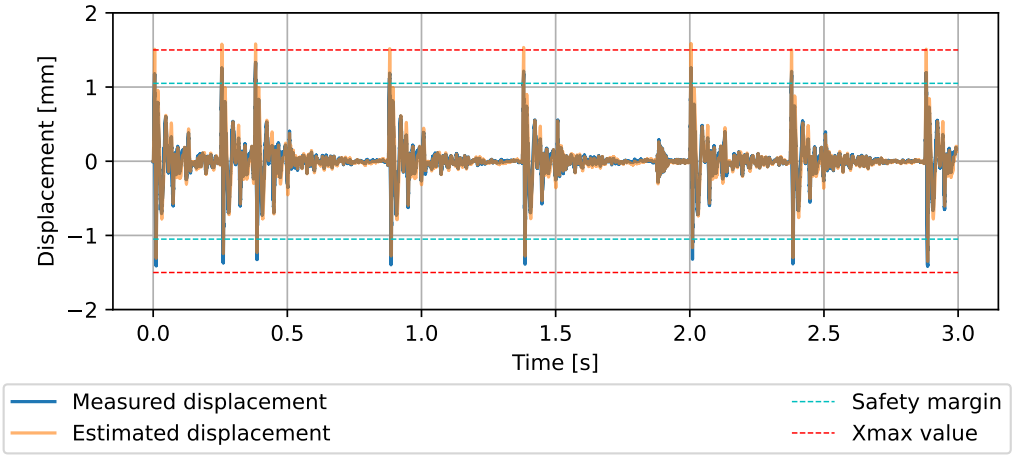


Figure 12: Comparison between the actual displacement and the estimated one using a drum kit loop signal with Xmax protection in real-time.

Zoomed results of one of the impulse components are depicted in Figure 13, illustrating both unprotected and protected outcomes. The figure demonstrates the validation of the estimator model for fast-changing signals in both scenarios—with and without protection. Additionally, the asymmetric phenomenon is evident in all situations, whether protected or unprotected, and in both measurement and estimation results. This asymmetry could be attributed not only to nonlinear factors as before but also to differences in the attack and release processes during the recording of batterie instruments' sounds. For instance, when drums are struck, the membrane descends faster than it releases, leading to an asymmetry in acoustic pressure that is also captured in the recording.

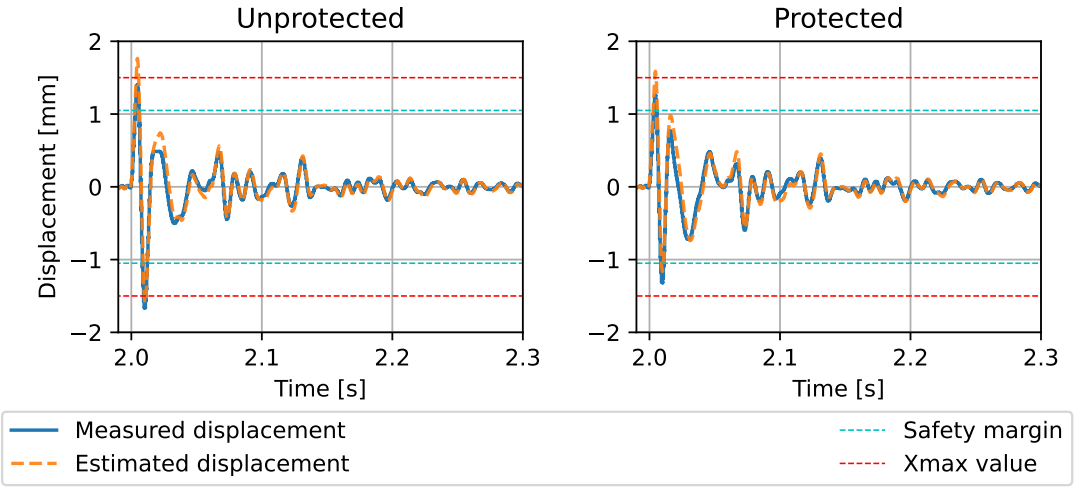


Figure 13: Comparison between the actual displacement and the estimated one using a drum kit loop signal with and without Xmax protection in real-time.

7 Discussion

7.1 Evaluation of the Protection Mechanism

Once the effectiveness of the protection system is confirmed, the measurement results, with and without the protection system, are represented in a single graph for comparison. Additionally, the dynamic cut-off frequency is illustrated, revealing the cut-off frequency modulation in response to both slow and fast-changing signals. This insight can facilitate the optimization of controller parameters.

The results for the Rsss signal are presented in Figure 14. As mentioned earlier, the protected displacement consistently remains within the safety margin range. Moreover, the protected segment displacement value closely aligns with the value of the safety margin, which ensures radiation efficiency simultaneously. This is achieved through the careful selection of attack and release times. Regarding its cut-off frequency, in general, it increases when the displacement exceeds the margin and then decreases gradually.

Upon closer inspection, due to the signal phase, the cut-off frequency briefly experiences both an attack and a release within each cycle. It is known that the frequency of the Rsss signal decreases over time. The later the individual signal cycle appears, the lower the frequency it is, and the higher the maximum displacement amplitude without the protection, making the attack and release process of the cut-off frequency more evident. This discovery suggests potential research opportunities to explore the relationship between adaptive attack and release time and frequency. However, it's important to note that this topic falls outside the scope of the current project.

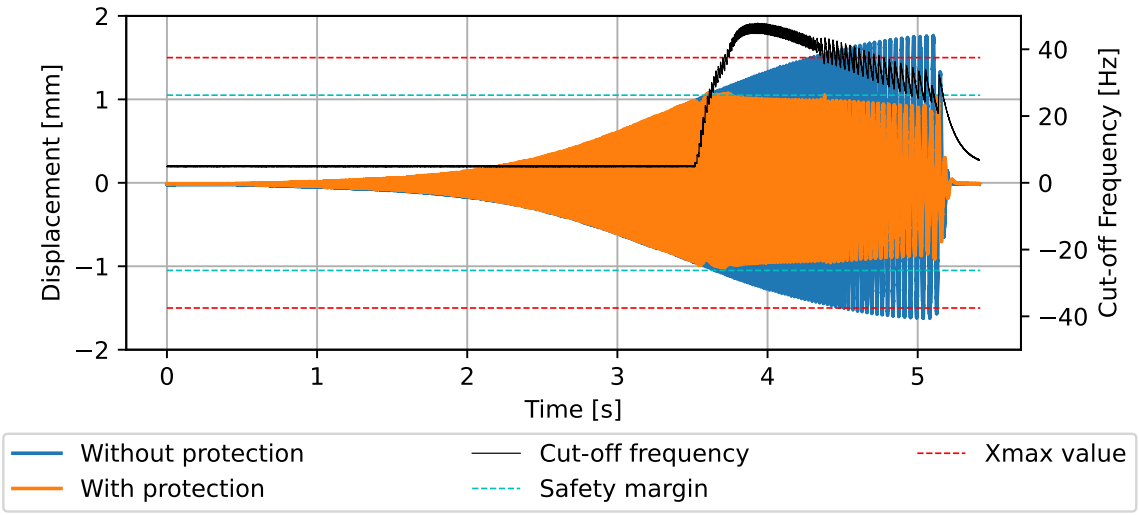


Figure 14: Comparison of displacement with and without the real-time Xmax protection with a reversed swept sine as the input signal.

Figure 15 illustrates the results of the drum loop signal. For signals with numerous impulses, such as this one, even though its displacement is mostly lower than the safety margin, there is an instantaneous risk of damaging the loudspeakers. Fortunately, the results indicate that the protection system effectively balances a rapid response to dangerous impulses while ensuring sound quality for the rest of the signal, though with a less effective result compared to the Rsss signal.

The cut-off frequency exhibits similar dynamic trends as the displacement signal, rapidly increasing to safeguard against fast impulses and subsequently decreasing gradually to maintain sound quality. Upon examining the cut-off frequency axis, it is evident that the maximum cut-off frequency here is approximately 25 Hz, sig-

nificantly lower than $f_{c_{max}}$ =200 Hz. The decision not to define a faster τ_a is due to the higher instability induced by a more rapid time-varying system. The resultant DC offset leads to an even higher amplitude on the positive or negative side. The observed maximum cut-off frequency value is determined by the relationship between the lasting time of the impulse and the attack time τ_a. Selecting attack and release times based on the duration and amplitude of impulses is a broader consideration within the previously mentioned future topic.

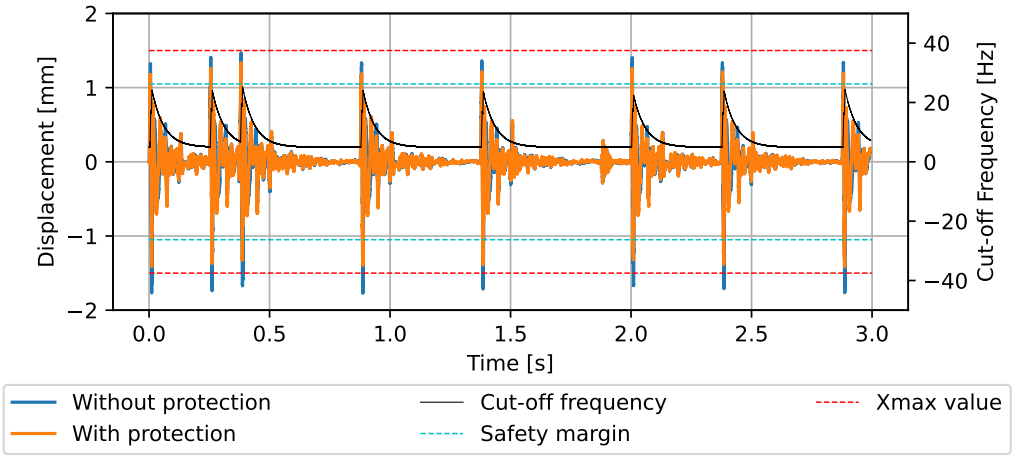


Figure 15: Comparison of displacement with and without the Xmax protection in real-time with a drum kit loop as the input signal.

7.2 Limitations of the system

The protection demonstrator has certain limitations. Firstly, the loudspeaker is not a constant system; its properties fluctuate with changes in the atmosphere. For example, variations in environmental temperature can affect the stiffness of the surroundings, subsequently influencing mechanical compliance values. These alterations in parameters can lead to modifications in the estimator model, crucially impacting the effectiveness of the protection system. Unfortunately, the system designed for this project lacks the setup to measure the model parameters dynamically, which is essential for real-time adaptation.

The instability induced by the dynamic high-pass filter stands as the primary reason for the deterioration in sound quality, one limitation consistently highlighted in preceding sections. The following provides a detailed explanation of this limitation. Consider the input signal as x and the corresponding output as y. At the moment 'n-1', the cut-off frequency is denoted as $f_c[n-1]$, with the calculated output value being $y_{n-1}[n-1]$. Upon transitioning to the next moment 'n', the cut-off frequency changes to $f_c[n]$. However, the computation process for the output value $y_n[n]$ requires the correct previous value $y_n[n-1]$, while only $y_{n-1}[n-1]$ is available. In the context, for the second-order filter, the corrected coefficients α and β are defined as the ratio of the actual applied output values to the correct previous output values, as illustrated in Equation 11.

$$\alpha = \frac{y_{n-1}[n-1]}{y_n[n-1]}, \qquad \beta = \frac{y_{n-2}[n-2]}{y_n[n-2]}. \tag{11}$$

The real calculation process of $y[n]$ is shown in Equation 12, where b_i and a_j are the digital coefficients calculated according to the cut-off frequency $f_c[n]$.

$$y_n[n] = \sum_{i=0}^{2} b_i x[n-i] - \sum_{j=1}^{2} a_j y_{n-j}[n-j] = \sum_{i=0}^{2} b_i x[n-i] - a_1 \alpha y_n[n-1] - a_2 \beta y_n[n-2]. \tag{12}$$

The Z-transform result of the digital filter is expressed in Equation 13. Additionally, its zero-pole form is presented, where q_1 and q_2 represent the two zeros, and p_1' and p_2' denote the real poles. However, due to corrected coefficients, the values of the real poles deviate from the desired values. This discrepancy results in

the real filter having a new cut-off frequency value, leading to the induced instability of the system.

$$H_n(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 \alpha z^{-1} + a_2 \beta z^{-2}} = \frac{b_0(1 - q_1 z^{-1})(1 - q_2 z^{-1})}{(1 - p_1' z^{-1})1 - p_2' z^{-1})}. \tag{13}$$

Yet another limitation is the system's lack of sufficient adaptability. The two tested signals presented in the report are illustrative, but in actual customer use cases, audio characteristics can vary significantly. An audio session may include both segments of slow-varying signals, like the Rsss, and segments with impulses, such as the drum kit loop signals. Further research has been conducted to realize the concept of 'adaptivity,' as discussed in Subsection 7.3.

7.3 Exploration of 'Adaptive'

The key to adaptivity lies in the appropriate selection of attack and release times. However, a fixed setup may not be optimal for general cases. Two explorations were tried during the project to achieve an adaptive system.

The initial solution involves designing a manually adjustable setup for the system, which was adopted during the project. Our Arduino code incorporates an input window that allows experienced users to manually modify τ_a and τ_r. Manufacturers have the opportunity to enhance this setup by designing a more aesthetically pleasing software user interface or incorporating a mechanically rotating knob.

However, for many clients lacking professional knowledge and instructions, manually adapting the setup may be challenging. In this context, the concept of attack and release acceleration is introduced. Analogous to the well-known acceleration concept, these two parameters control the attack and release times. A monitoring element is introduced to the protection system to detect if signals consecutively exceed the safety threshold. If so, the attack acceleration is triggered, causing the cut-off frequency to increase more rapidly, which means a decreased attack time. Nevertheless, this introduces new challenges in terms of selection and optimization, requiring further research.

8 Conclusion

This project researched the Xmax protection for loudspeakers and designed a protection demonstrator using the Teensy board. The system protects loudspeakers by employing a dynamic high-pass filter when there is a risk of exceeding the maximum displacement.

The project began with the design and analysis of the logical structure of the protection circuit, comprising a detector, estimator, and controller. The estimator predicts the loudspeaker's displacement, the detector calculates preceding signal input to the loudspeaker, and the controller judges whether the signal is dangerous, activating the protection mechanism. The project chose the linear filter model due to its simplicity. The controller incorporates a comparator and a dynamic high-pass filter. The estimated displacement is compared with a safety threshold, and if the displacement exceeds the threshold, the cutoff frequency of the high-pass filter increases. An analogous electrical LC circuit with exponential cutoff frequency modulation is designed and utilized, with the changing rate determined by the attack and release time.

The linear characteristics of the loudspeaker were accurately measured using the Klippel machine, as these parameters play an important role in the efficacy of the protection system, shaping the estimator model. The implemented protection mechanism is integrated into the Teensy board, compiled on Arduino using the C language. In addition to delivering the protected signal to the loudspeaker, the Teensy board also provides output for calculated values such as estimated displacement and cutoff frequency. The actual loudspeaker displacement was captured using a displacement sensor. These results were analyzed and utilized in the evaluation of the protection system.

A comparison between the calculated and measured displacements of the Teensy board in the absence of protection revealed the effective performance of the estimator model. The slight differences observed can contribute positively to the protection mechanism. When the protection system is turned on, the estimated and measured results demonstrate effective performance for both slow-changing signals and signals with rapid impulses with the appropriate chosen attack and release times. A demonstration video featuring a variety of tested signals has been recorded and is available for viewing on the website[1].

The protection system, while effective, does have certain limitations. Variations in loudspeaker parameters across time, space, and environmental conditions may introduce errors in the estimator. The use of dynamic IIR high-pass filters can potentially lead to system instability. A more adaptive attack and release times should be proposed for the various audio styles. To address these challenges, a further approach was proposed and discussed in the report to improve the adaptability. The concept of attack and release acceleration has been suggested to automatically adjust these times. However, it is crucial to note that the limitations and explorations mentioned above require further in-depth analysis, experimental exploration, and engineering studies for a comprehensive understanding and potential solutions.

[1] https://youtu.be/tgGlCQ0jw64

Appendices

A State-Space Model

The equivalent electrical circuit of the loudspeaker considering the eddy current is represented in Figure 16. Compared to the linear filter model, the linear state space model directly discretizes the derivative components within the equations and formulates the equation for the next state. All derivations are performed in the time domain. Four variables are involved in the differential Equation 14: the total current $i(t)$ flowing through the electrical circuit, the current i_2 flowing through the resistor $\mathbf{L_2}$, the displacement $x(t)$, and the velocity $v(t)$ of the membrane. The voltage $u(t)$ represents the supplied input to the system. To ensure a unique solution for the next state, the system of equations should be full rank, indicating that the number of variables must equal the number of equations. The additional differential equation required to achieve a third-order full-rank system is the relationship between $x(t)$ and $v(t)$. The system used for the linear state space model is described by Equations 14.

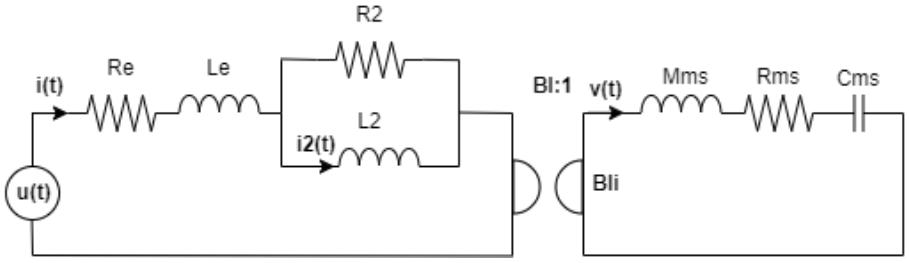


Figure 16: Electrical circuit of the electrical and mechanical part of the loudspeaker considering the eddy current

$$\begin{cases} u(t) = \mathbf{R_e} i(t) + \mathbf{L_e} \frac{di(t)}{dt} + \mathbf{R_2}(i(t) - i_2(t)) + \mathbf{Bl} v(t), \\ \mathbf{L_2} \frac{di_2(t)}{dt} = \mathbf{R_2}(i(t) - i_2(t)), \\ \mathbf{Bl} i(t) = \mathbf{M_{ms}} \frac{dv(t)}{dt} + \mathbf{R_{ms}} v(t) + \frac{x(t)}{\mathbf{C_{ms}}}, \\ \frac{dx(t)}{dt} = v(t). \end{cases} \tag{14}$$

A.1 Linear State Space Model

Equations 15 show the more clear expressions of the differential equations. All derivations are on the left side of the equals and the others are on the right side.

$$\begin{cases} \frac{di(t)}{dt} = -\frac{\mathbf{R_e}+\mathbf{R_2}}{\mathbf{L_e}} i(t) + \frac{\mathbf{R_2}}{\mathbf{L_e}} i_2(t) - \frac{\mathbf{Bl}}{\mathbf{L_e}} v(t) + \frac{1}{\mathbf{L_e}} u(t), \\ \frac{di_2(t)}{dt} = \frac{\mathbf{R_2}}{\mathbf{L_2}} (i(t) - i_2(t)), \\ \frac{dx(t)}{dt} = v(t), \\ \frac{dv(t)}{dt} = \frac{\mathbf{Bl}}{\mathbf{M_{ms}}} i(t) - \frac{1}{\mathbf{M_{ms}} \mathbf{C_{ms}}} x(t) - \frac{\mathbf{R_{ms}}}{\mathbf{M_{ms}}} v(t). \end{cases} \tag{15}$$

To enhance the visualization and manipulation of the entire equation, it is essential to express the complete equation in matrix form. In this context, the variable vector $(i(t), x(t), v(t))^T$ is denoted by a unified variable $\mathrm{x(t)}$. The input variable, or voltage variable, is represented by $\mathrm{U(t)}$. The matrix representation of the system of equations is illustrated in Equation 16. Given that there is only one input variable $u(t)$ across the three equations, the matrix $\mathbf{B}$ is of size (3×1), and the input vector is of size $(1\times\mathrm{N})$, where N is the number of

signals.

$$\frac{d}{dt}\begin{pmatrix} i(t) \\ i_2(t) \\ x(t) \\ v(t) \end{pmatrix} = \begin{pmatrix} -\frac{\mathbf{R_e}+\mathbf{R_2}}{\mathbf{L_e}} & \frac{\mathbf{R_2}}{\mathbf{L_e}} & 0 & -\frac{\mathbf{Bl}}{\mathbf{L_e}} \\ \frac{\mathbf{R_2}}{\mathbf{L_2}} & -\frac{\mathbf{R_2}}{\mathbf{L_2}} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{\mathbf{Bl}}{\mathbf{M_{ms}}} & 0 & -\frac{1}{\mathbf{M_{ms}C_{ms}}} & -\frac{\mathbf{R_{ms}}}{\mathbf{M_{ms}}} \end{pmatrix} \begin{pmatrix} i(t) \\ i_2(t) \\ x(t) \\ v(t) \end{pmatrix} + \begin{pmatrix} \frac{1}{\mathbf{L_e}} \\ 0 \\ 0 \\ 0 \end{pmatrix} u(t) \Rightarrow \frac{dx(t)}{dt} = \mathbf{A}x(t)+\mathbf{B}U(t). \tag{16}$$

Initially, three straightforward discretization methods—**Euler Forward**, **Euler Backward**, and **Bilinear**—are introduced and employed to derive the state equation. The discretization process for these three methods is depicted in Equation 17. In the case of the bilinear transform, the outcome is not direct and can be explained from two perspectives: 1. the transition from the Laplace domain to the z domain, followed by utilizing z-transform properties to obtain the result. 2. the determination of the derivative result at a given moment involves averaging this moment with the next.

$$\begin{cases} \text{Euler Forward: } \frac{x[n+1]-x[n]}{T_s} = \mathbf{A}x[n] + \mathbf{B}U[n], \\ \text{Euler Backward: } \frac{x[n+1]-x[n]}{T_s} = \mathbf{A}x[n+1] + \mathbf{B}U[n+1], \\ \text{Bilinear: } \frac{x[n+1]-x[n]}{T_s} = \frac{\mathbf{A}}{2}\left(x[n+1]+x[n]\right) + \frac{\mathbf{B}}{2}\left(U[n+1]+U[n]\right). \end{cases} \tag{17}$$

The state equation in the three methods is expressed in Equation 18. While not directly derived from Equation 17, the input value $U[n+1]$ is approximated as $U[n]$. This approximation holds under the condition of a slow signal or a high sampling rate. The unified form of the state equation for the three methods is given by $x[n+1] = \mathbf{A_d}x[n] + \mathbf{B_d}U[n]$, where the expression of $\mathbf{A_d}$ and $\mathbf{B_d}$ changes with the method.

$$\begin{cases} \text{Euler Forward: } x[n+1] = \left(\mathbf{I} + T_s\mathbf{A}\right)x[n] + \mathbf{B}T_sU[n], \\ \text{Euler Backward: } x[n+1] = \left(\mathbf{I} - T_s\mathbf{A}\right)^{-1}x[n] + \left(\mathbf{I} - T_s\mathbf{A}\right)^{-1}\mathbf{B}T_sU[n], \\ \text{Bilinear: } x[n+1] = \left(\mathbf{I} - T_s\frac{\mathbf{A}}{2}\right)^{-1}\left(\mathbf{I} + T_s\frac{\mathbf{A}}{2}\right)x[n] + \left(\mathbf{I} - T_s\frac{\mathbf{A}}{2}\right)^{-1}\mathbf{B}T_sU[n]. \end{cases} \tag{18}$$

A.2 Non-Linear State Space Model

The non-linear parameters in the loudspeaker system include the force factor $\mathbf{Bl}(x)$, mechanical compliance $\mathbf{C_{ms}}(x)$, and the electrical inductance $\mathbf{L_e}(x)$. These parameters can be modeled as polynomial equations, such as $\mathbf{Bl}(x) = \sum_{n=0}^{N}\mathbf{Bl_n}x^n$. The values of the non-linear parameters are determined by the real displacement x at a given moment. However, in loudspeaker modeling, only the preceding displacement values are known. To handle the non-linear state space calculation, both explicit and implicit methods can be employed. In this project, the explicit method is utilized and explained.

Equation 19 presents the non-linear differential formula, highlighting that the coefficient matrices $\mathbf{A}$ and $\mathbf{B}$ are also dependent on the displacement.

$$\frac{d}{dt}\begin{pmatrix} i(t) \\ i_2(t) \\ x(t) \\ v(t) \end{pmatrix} = \begin{pmatrix} -\frac{\mathbf{R_e}+\mathbf{R_2}}{\mathbf{L_e}(x)} & \frac{\mathbf{R_2}}{\mathbf{L_e}(x)} & 0 & -\frac{\mathbf{Bl}(x)}{\mathbf{L_e}(x)} \\ \frac{\mathbf{R_2}}{\mathbf{L_2}} & -\frac{\mathbf{R_2}}{\mathbf{L_2}} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{\mathbf{Bl}(x)}{\mathbf{M_{ms}}} & 0 & -\frac{1}{\mathbf{M_{ms}C_{ms}(x)}} & -\frac{\mathbf{R_{ms}}}{\mathbf{M_{ms}}} \end{pmatrix} \begin{pmatrix} i(t) \\ i_2(t) \\ x(t) \\ v(t) \end{pmatrix} + \begin{pmatrix} \frac{1}{\mathbf{L_e}} \\ 0 \\ 0 \\ 0 \end{pmatrix} u(t) \tag{19}$$

The calculation processes of the state values are almost the same as those of the linear state space model. The difference lies in the necessity to calculate the values of the non-linear parameters at each moment. With the explicit method, the values are based on the previous displacement value $x[n-1]$. For example, $\mathbf{Bl}[n] = \sum_{i=0}^{N}\mathbf{Bl_i} \times (x[n-1])^i$. Using these non-linear parameter values, the coefficient matrices $A[n]$ and $B[n]$ can be constructed. Subsequently, the next state value can be calculated using Equation 18 with various discretization methods. The nonlinear parameters measured with Klippel LSI system are shown in Section C.

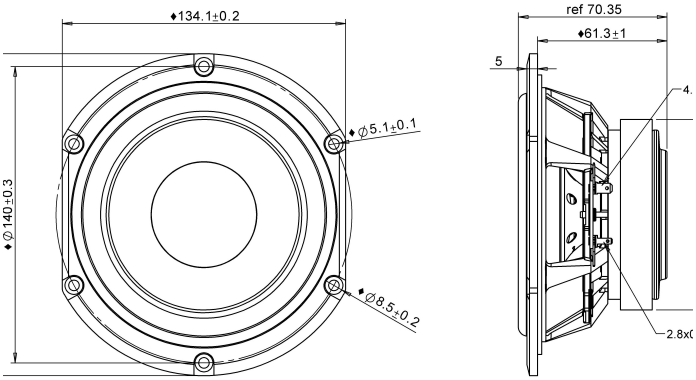
B Peerless HDS-P830860 Datasheet

Peerless

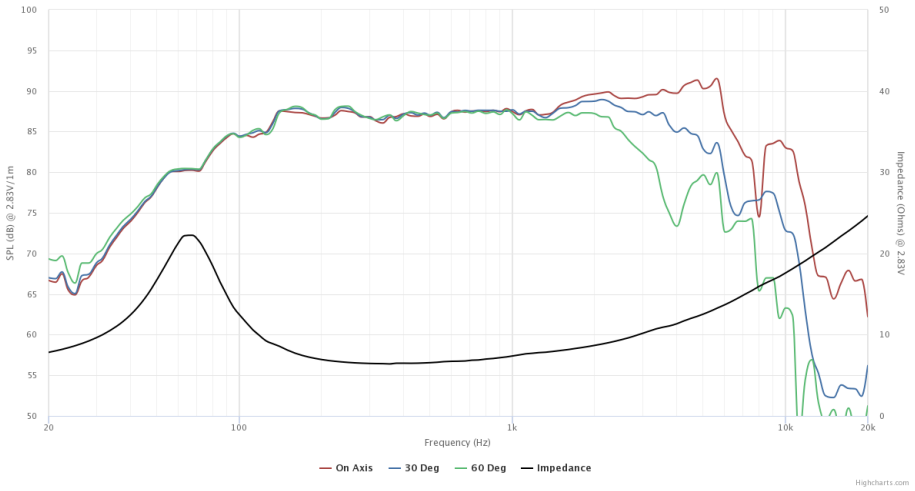
HDS-P830860

WOOFER

- Cast Aluminum Frame
- Vented Cone Neck
- Coated Paper Cone
- Ferrite Magnet
- Large Excursion



SPECIFICATIONS			
Transducer Size		5.25	in
Impedance		8	Ω
Frequency Range [1]		85 - 4000	Hz
Sensitivity [2] **(2.83V | 1W @ 1m)**		86.5 | 86.5	dB
Power Rating (IEC 268-5)		30	W
Voice Coil Size		25.7	mm
Air Gap | Winding Height	H_{ag} | H_{vc}	6 | 13	mm
Net Weight		1.06	kg
PARAMETERS [3]			
Eff. Piston Area	S_d	89.9	cm^2
DC Resistance	R_e	6	Ω
Minimum Impedance	Z_{min}	6.4	Ω
Inductance	L_e	0.278	mH
Resonance Frequency [4]	F_s	72	Hz
Mechanical Q Factor	Q_{ms}	2.08	-
Electrical Q Factor	Q_{es}	0.725	-
Total Q Factor	Q_{ts}	0.54	-
Moving Mass	M_{ms}	8.88	g
Compliance	C_{ms}	560	µm/N
Equivalent Volume	V_{as}	6.36	L
Motor Force Factor	Bl	5.74	Tm
Motor Efficiency	β	5.51	$(Bl)^2 / R_e$
Linear Excursion [5]	X_{max}	5.5	mm
Max Mechanical Excursion [6]	X_{mech}	-	mm



Details on this spec sheet are for reference only and should not be used for setting production limits. Specifications and product cosmetics are subject to change without notice. Peerless is a registered trademark of Tymphany Enterprises. All measurements conducted in test lab at 25°C ±10°C, 50%RH ±10%. [1] Specified by Engineering as linear working range of transducer. [2] Measured at 2.83V at 1m and normalized to 1W with respect to nominal impedance. [3] Measured in Free Air without preconditioning, therefore subject to some deviation. [4] Impedance and Fs value measured under different conditions. [5] Equal/Overhung: $(H_{vc} - H_{ag})/2 + H_{ag}/3$. Underhung: $(H_{ag} - H_{vc})/2 + H_{vc}/3$. [6] Mechanically limited excursion (e.g. bottoming, spider crash).

C Kilpple LSI Mesurement Result

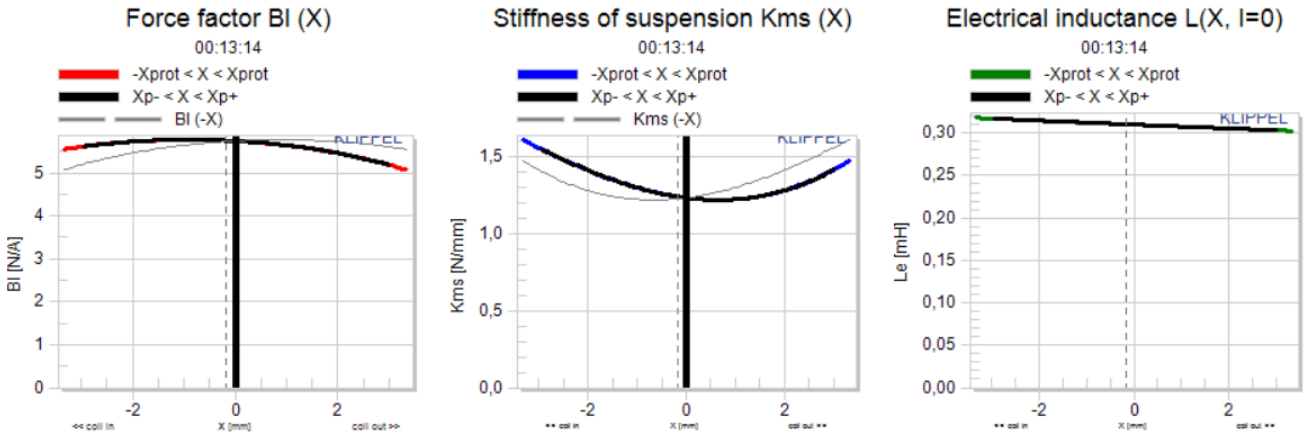


Figure 17: Nonlinear parameters curve with the loudspeaker membrane displacement

Power Series			
Bl0 = Bl (X=0)	5.7409	N/A	constant part in force factor
Bl1	-0.063162	N/Amm	1st order coefficient in force factor expansion
Bl2	-0.035682	N/Amm^2	2nd order coefficient in force factor expansion
Bl3	-0.00077597	N/Amm^3	3rd order coefficient in force factor expansion
Bl4	-0.00032431	N/Amm^4	4th order coefficient in force factor expansion

Figure 18: Nonlinear polynomial coefficients in force factor expansion

C0 = Cms (X=0)	0.81326	mm/N	constant part in compliance
C1	0.024614	1/N	1st order coefficient in compliance expansion
C2	-0.017441	1/Nmm	2nd order coefficient in compliance expansion
C3	-0.0015458	1/Nmm^2	3rd order coefficient in compliance expansion
C4	0.00023521	1/Nmm^3	4th order coefficient in compliance expansion

Figure 19: Nonlinear polynomial coefficients in mechanical compliance expansion

L0 = Le (X=0)	0.30999	mH	constant part in inductance
L1	-0.0023433	mH/mm	1st order coefficient in inductance expansion
L2	3.7761e-005	mH/mm^2	2nd order coefficient in inductance expansion
L3	3.9252e-006	mH/mm^3	3rd order coefficient in inductance expansion
L4	3.8780e-008	mH/mm^4	4th order coefficient in inductance expansion

Figure 20: Nonlinear polynomial coefficients in electrical inductance expansion

D Arduino Code

```
#include <ADC.h>
#include <ADC_util.h>
#include <SPI.h>
```

```
4
5  #include "LS_Model.h"
6
7  LS_Model lsModel_p;
8  LS_Model lsModel_np;
9
10
11
12 // ------------------------------------------------
13 // Loudspeaker
14 // ------------------------------------------------
15 // #Parameters - Basic Linear Model
16 // At low frequencies Le is approximately 0
17
18 // Previous parameters
19 // const float Mms = 8.897e-3;     //#[Kg]
20 // const float Rms = 1.75;         //#[N.s/m]
21 // const float Kms = 1.73 * 1000;  //#[N/m]
22 // const float Bl = 5.97;          //#[N/A]
23 // const float Re = 5.72;          //#[ohm]
24 // const float Le = 0.29e-3;  // [H]
25
26 // Peerless Loudspeaker
27 const float Mms = 9.257e-3;  //#[Kg]
28 const float Rms = 1.866;     //#[N.s/m]
29 const float Kms = 1800;      //#[N/m]
30 const float Bl = 5.816;      //#[N/A]
31 const float Re = 6.76;       //#[ohm]
32
33 // Audax Loudspeaker
34 // const float Mms = 20.4e-3;  //#[Kg]
35 // const float Rms = 1.379;     //#[N.s/m]
36 // const float Kms = 520;      //#[N/m]
37 // const float Bl = 9.2;      //#[N/A]
38 // const float Re = 6.06;       //#[ohm]
39
40 // Amplifier + Teensy board Gain
41 float Gain = 8;
42 // Maximum displacement definition (Xmax)
43 float Xmax = 1.5e-3;
44 // Estimated displacement with protection
45 volatile float x_est;
46 // Estimated displacemet without protection
47 volatile float x_est_nc;
48 // Protected signal, output of the adaptive filter
49 volatile float out;
```

```
50
51 ////////////////////////////
52 // Filter initialization //
53 ////////////////////////////
54
55 // Linear coefficients of the discretized transfer
56 // function of the loudspeaker for implementing the
57 // IIR filter
58 float b_0, b_1, b_2, a_0, a_1, a_2;
59
60 // Coefficients for the adaptive high-pass filter
61 volatile float b0_hp, b1_hp, b2_hp, a0_hp, a1_hp, a2_hp;
62
63 // Adaptive filter cut-off frequency initialization
64 volatile float fc = 5;
65
66 // Attack and release time in seconds for the adaptive filter.
67 // tao_a: Attack time, tao_r: Release time
68 float tao_a = 0.01, tao_r = 0.1;
69 // float tao_a = 0.001, tao_r = 0.01;
70
71 // Maximum and minimum cut-off frequency for the adaptive filter
72 float fcmax = 200;
73 float fcmin = 5;
74
75 // High-pass filter buffer
76 volatile float d0_hp = 0, d1_hp = 0, d2_hp = 0, d3_hp = 0;
77
78 bool itsOn = true, itsFirst = true;
79 int incomignByte = 0;
80 bool modRelease = false;
81 bool modAttack = false;
82
83 /*
84  * IN1: A14
85  * IN2: A15
86  * IN3: A12
87  * IN4: A13
88  */
89 const int readPin_In1 = A14;
90 const int readPin_In3 = A12;
91 const int readPin_In4 = A13;
92
93 const int outPin_Aux1 = A21;
94 const int outPin_Aux2 = A22;
95
```

```
96  const int resolutionADC = 16;
97  const int resolutionDAC = 16;   //MAX 5717 DAC
98  const int resolutionDACT = 12;  //Teensy DAC
99
100 const float VrefADC = 3.3;    // Teensy ADC VRef
101 const float VrefDACT = 3.3;   // Teensy DAC VRef
102 const float VrefDAC = 4.096;  //Modified from 10.0 19/04/22
103
104 const int slaveSelectPin = 9;
105 const int LDAC = 10;
106
107 const int DIO = 2;
108
109 const float conversionConstADC = VrefADC / ((1 << resolutionADC) - 1);
110 const float conversionConstDAC = ((1 << resolutionDAC) - 1) / (2.0 *
      VrefDAC);
111 const float conversionConstDACT = ((1 << resolutionDACT) - 1) / VrefDACT;
112 const int ADCAverages = 0;
113
114
115 /*
116  * Timing
117  */
118 const float sampleRateHz = 48000.0;
119 const float dt = 1.0 / sampleRateHz;
120 const float SPIMHz = 22.5;
121 const long SPIHz = SPIMHz * 1000 * 1000;
122
123
124 /**
125  * Initialisation I/O
126  */
127 //Input (ADC) initialisation
128 volatile uint16_t valADC0 = 0, valADC1 = 0;
129 volatile uint8_t VALADC0Ready = false, VALADC1Ready = false;
130 volatile float input1 = 0.0, input3 = 0.0, input4 = 0.0;
131
132
133 //Output (DAC) signal initialisation
134 volatile uint16_t valDAC = 0, valDACT = 0, valDACT2 = 0;
135 volatile float val4DACOut = 0.0;
136
137 // Using the Pedvide ADC Library on Github
138 ADC *adc = new ADC();  // adc object
139
140
```

```
141
142 void setup(void) {
143   // Declare needed pins as inputs or outputs
144   pinMode(readPin_In1, INPUT);
145   pinMode(readPin_In4, INPUT);
146   pinMode(slaveSelectPin, OUTPUT);
147   pinMode(DIO, OUTPUT);
148   pinMode(LDAC, OUTPUT);
149
150   /**
151    * Here we initialise the ADCs
152    */
153
154   // When using the Pedvide ADC library we can set more options
155   adc->adc0->setAveraging(ADCAverages);                                    // set
        number of averages
156   adc->adc0->setResolution(resolutionADC);                                 // set
        bits of resolution
157   adc->adc0->setReference(ADC_REFERENCE::REF_3V3);                         // Set
        voltage reference for ADC.
158   adc->adc0->setSamplingSpeed(ADC_SAMPLING_SPEED::MED_SPEED);              //
       change the sampling speed
159   adc->adc0->setConversionSpeed(ADC_CONVERSION_SPEED::MED_SPEED);          //
       change the conversion speed
160
161   adc->adc0->stopPDB();
162   adc->adc0->startSingleRead(readPin_In1);  // call this to setup
       everything before the pdb starts, differential is also possible
163   adc->adc0->enableInterrupts(adc0_isr);
164   adc->adc0->startPDB(sampleRateHz);  //frequency in Hz
165
166   adc->adc1->setAveraging(ADCAverages);                                    // set
        number of averages
167   adc->adc1->setResolution(resolutionADC);                                 // set
        bits of resolution
168   adc->adc1->setReference(ADC_REFERENCE::REF_3V3);                         // Set
        voltage reference for ADC.
169   adc->adc1->setSamplingSpeed(ADC_SAMPLING_SPEED::MED_SPEED);              //
       change the sampling speed
170   adc->adc1->setConversionSpeed(ADC_CONVERSION_SPEED::MED_SPEED);          //
       change the conversion speed
171
172   adc->adc1->stopPDB();
173   adc->adc1->startSingleRead(readPin_In4);  // call this to setup
       everything before the pdb starts, differential is also possible
174   adc->adc1->enableInterrupts(adc1_isr);
```

```
175   adc->adc1->startPDB(sampleRateHz);  //frequency in Hz
176
177
178   NVIC_SET_PRIORITY(IRQ_USBOTG, 200);
179
180   /**
181    * Here we initialise the SPI channel
182    */
183
184   SPI.begin();
185   SPI.beginTransaction(SPISettings(SPIHz, MSBFIRST, SPI_MODE0));
186
187   /**
188    * Incase you need to use the serial monitor/plotter
189    */
190   Serial.begin(115200);
191
192   // Here the loudspeaker filter model coefficients are computed
193   Bilinear_coeff(Re, Bl, Mms, Kms, Rms, sampleRateHz);
194
195   /**
196    * Here we set up the onboard DACs
197    *
198    * Definition de la resolution des DAC Teensy (si besoin)
199    */
200   analogWriteResolution(12);
201 }
202
203
204 void Operations(void) {
205   /**
206    * SPI output is done first.
207    * This ensure the DAC output timing is synchronised with the ADC via
       the PDB.
208    *
209    * La sortie via SPI est fait en premier.
210    * C'est pour synchroniser la sortie DAC SPI avec l'ADC. Le frequence d
       'echantillonage du DAC et ADC est donc gerer par le PDB (Programmable
       Delay Block)
211    */
212   uint8_t SPIBuff[2] = { 0 };  //Initialise the SPI buffer.  Initialiser
       la buffer SPI.
213   SPIBuff[0] = valDAC >> 8;    // Bit Splitting. 16 bit -> |8 bit||8 bit|
214   SPIBuff[1] = valDAC & 0xFF;  // Separation des bits fort et faible. 16
       bit -> |8 bit||8 bit|
215
```

```
216  /**
217   * Sending data to MAX5717 Via SPI. See Data sheet for details
218   */
219  digitalWrite(LDAC, HIGH);
220  // take the SS pin low to select the chip:
221  digitalWrite(slaveSelectPin, LOW);
222  //  send in the address and value via SPI:
223  SPI.transfer(SPIBuff, 2);
224  // take the SS pin high to de-select the chip:
225  digitalWrite(slaveSelectPin, HIGH);
226  digitalWrite(LDAC, LOW);
227
228
229  /**
230   * Here we read the ADC values and convert into a voltage, while
     removing any offsets
231   */
232
233  //Read ADC Value and convert to voltage
234  input1 = (valADC0 * conversionConstADC - 1.65);
235  // input4 = (valADC1 * conversionConstADC - 1.65);
236  //Serial.println(input2);
237  /**
238   * Preparing value for DAC Output - SPI
239   */
240
241  if (itsFirst) {
242    // First displacement sample is calculated with the input signal (
     input1)
243    x_est = lsModel_p.x_predictor(b_0, b_1, b_2, a_1, a_2, input1, Gain);
244    itsFirst = false;
245  } else {
246    // Other displacement samples are calculated with the filtered signal
     (out)
247    x_est = lsModel_p.x_predictor(b_0, b_1, b_2, a_1, a_2, out, Gain);
248  }
249
250  // Estimation of the not corrected displacement
251
252  // Not necessary but useful for comparing displacement estimation with
253  // the original signal.
254  x_est_nc = lsModel_np.x_predictor(b_0, b_1, b_2, a_1, a_2, input1, Gain
     );
255
256  // For changing the cut-off frequency of the adaptive filter
257  DynaFreq(x_est, Xmax);
```

```
258   // Here is where the signal is filtered
259   out = High_Pass_BW(fc, sampleRateHz, input1);
260 
261   // This is for monitoring the cut-off frequency through a digital
       output
262   valDACT2 = ((fc / 200 + VrefDACT * 0.5) * conversionConstDACT);
263 
264   if (itsOn) {
265     // Activate protection
266 
267     // To output estimated displacement according to the output
268     valDACT = ((x_est * 500 + VrefDACT * 0.5) * conversionConstDACT);
269     val4DACOut = out;
270   } else {
271     // Bypass protection
272 
273     // To output the estimated displacement without protection
274     valDACT = ((x_est_nc * 500 + VrefDACT * 0.5) * conversionConstDACT);
275     val4DACOut = input1;
276   }
277 
278   valDAC = (uint16_t)((val4DACOut + VrefDAC) * conversionConstDAC);
279 
280   /**
281    * DAC Output - Teensy 12 bit
282    */
283   // valDACT = ((val4DACOut + VrefDACT*0.5)*conversionConstDACT);
284 
285 
286   analogWrite(outPin_Aux1, valDACT);
287   analogWrite(outPin_Aux2, valDACT2);
288 }
289 
290 void loop(void) {
291   if (VALADC0Ready && VALADC1Ready) {
292     Operations();
293     VALADC0Ready = false;
294     VALADC1Ready = false;
295 
296     ////////////////////////////////////////////////////////////
297     // For controlling parameters through serial communication //
298     ////////////////////////////////////////////////////////////
299 
300     // Send serial monitor messages to modify some parameters:
301     // p: To activate Xmax protection
302     // n: To desactivate protection
```

```
303     // a: To modify attack time
304     //    After sending a, send the attack value in seconds
305     // r: To modigy release time
306     //    After sending r, send the release value in seconds
307
308     if (Serial.available() > 0) {
309       if (modRelease) {
310         float val = Serial.readString(4).toFloat();
311         if (val != 0) {
312           tao_r = val;
313           Serial.print("Release time: ");
314           Serial.print(val, 3);
315           Serial.println(" s");
316           modRelease = false;
317         }
318       } else if (modAttack) {
319         float val = Serial.readString(5).toFloat();
320         if (val != 0) {
321           tao_a = val;
322           Serial.print("Attack time: ");
323           Serial.print(val, 4);
324           Serial.println(" s");
325           modAttack = false;
326         }
327       }
328
329       incomignByte = Serial.read();
330       if ((char)incomignByte == 'p' && (!modAttack && !modRelease)) {
331         itsOn = true;
332         Serial.println("Protection on");
333       } else if ((char)incomignByte == 'n' && (!modAttack && !modRelease)
    ) {
334         itsOn = false;
335         Serial.println("Protection off");
336       } else if ((char)incomignByte == 'r' && !modRelease) {
337         modRelease = true;
338         Serial.println("Type the new release time in seconds");
339       } else if ((char)incomignByte == 'a' && !modAttack) {
340         modAttack = true;
341         Serial.println("Type the new attack time in seconds");
342       }
343     }
344   }
345 }
346
347
```

```
348
349 void adc0_isr() {
350   valADC0 = (uint16_t)adc->adc0->readSingle();
351   VALADC0Ready = true;
352 }
353
354 void adc1_isr() {
355   valADC1 = (uint16_t)adc->adc1->readSingle();
356   VALADC1Ready = true;
357 }
358
359 // pdb interrupt is enabled in case you need it.
360 //void pdb_isr(void) {
361 //        PDB0_SC &=~PDB_SC_PDBIF; // clear interrupt
362 //        //digitalWriteFast(LED_BUILTIN, !digitalReadFast(LED_BUILTIN) )
    ;
363 //}
364
365 void Bilinear_coeff(float Re, float Bl, float Mms, float Kms, float Rms,
    float fs) {
366   // This function discretizes the loudspeaker transfer function using
    the bilinear transform
367
368   // Linear model assuming Le = 0
369   // A and B are the analog coefficients of the transfer function of the
    loudspeaker
370   // a and b are the discrete coefficients
371
372   float Ts = 1 / fs;
373   float B_0 = 1;
374   float A_0 = (Re * Mms) / Bl;
375   float A_1 = (Re * Rms + pow(Bl, 2)) / Bl;
376   float A_2 = Re * Kms / Bl;
377
378   ////////////////////////////
379   // Bilinear discretization //
380   //                        //
381   // s = 2*fs*(z-1)/(z+1)    //
382   ////////////////////////////
383
384   b_0 = B_0;
385   b_1 = 2 * B_0;
386   b_2 = B_0;
387
388   a_0 = A_2 + 2 * A_1 / Ts + 4 * A_0 / pow(Ts, 2);
389   a_1 = 2 * A_2 - 8 * A_0 / pow(Ts, 2);
```

```
  a_2 = A_2 - 2 * A_1 / Ts + 4 * A_0 / pow(Ts, 2);

  // Here all the coefficients are normalized with a_0
  b_0 /= a_0;
  b_1 /= a_0;
  b_2 /= a_0;

  a_1 /= a_0;
  a_2 /= a_0;
  a_0 /= a_0;
};

// '''Compute cut-off frequency for the adaptive filter'''
void DynaFreq(float x_est, float Xmax) {
  float M = .7;   // Xmax safety margin
  float Na = tao_a * sampleRateHz;
  float Nr = tao_r * sampleRateHz;
  if (abs(x_est) > M * Xmax) {
    fc = (((fc - fcmin) / (fcmax - fcmin) - 1) * exp(-1.0 / Na) + 1) * (fcmax - fcmin) + fcmin;
    // Attack
  } else {
    fc = ((fc - fcmin) / (fcmax - fcmin) * exp(-1.0 / Nr) * (fcmax - fcmin) + fcmin);
    // Release
  }
}

////////////////////////////////
// Adaptive High-pass Filter //
////////////////////////////////
float High_Pass_BW(float fc, float fs, float input) {
  //  This a second-order Butterworth high-pass filter with frequency warping
  float y;
  float wc = 2 * PI * fc / fs;
  float Q = 1 / sqrt(2);
  float alpha = sin(wc) / (2 * Q);

  b0_hp = (1 + cos(wc)) / 2;
  b1_hp = -1 - cos(wc);
  b2_hp = (1 + cos(wc)) / 2;
  a0_hp = 1 + alpha;
  a1_hp = -2 * cos(wc);
  a2_hp = 1 - alpha;

```

```
  // Coefficients normalization
  b0_hp /= a0_hp;
  b1_hp /= a0_hp;
  b2_hp /= a0_hp;

  a1_hp /= a0_hp;
  a2_hp /= a0_hp;
  a0_hp /= a0_hp;

  y = b0_hp * input + b1_hp * d0_hp + b2_hp * d1_hp - a1_hp * d2_hp -
   a2_hp * d3_hp;

  // Update filter states
  d1_hp = d0_hp;
  d0_hp = input;
  d3_hp = d2_hp;
  d2_hp = y;

  return y;
};
```

Listing 2: Main Arduino file.

```

#include "LS_Model.h"

// x_predictor is a method of this class that can estimate, using an IIR
   filter structure (direct form I),
// the loudspeaker displacement based on the a and b coefficientes of the
    discretized transfer function,
// the input signal, and the gain.

volatile float LS_Model::x_predictor(float b_0, float b_1, float b_2,
   float a_1, float a_2, float input1, float Gain) {

  // Serial.print(input1);
  float y;
  // Apply filter
  input1 *= Gain;
  y = b_0 * input1 + b_1 * d0 + b_2 * d1 - a_1 * d2 - a_2 * d3;

  // Update filter states
  d1 = d0;
  d0 = input1;
  d3 = d2;
  d2 = y;

```

```
22   return y;
23 }
```

Listing 3: LS_Model.cpp file, class definition.

```
1
2 #ifndef LS_Model_H
3 #define LS_Model_H
4
5 #include <Arduino.h>
6
7 class LS_Model {
8 private:
9   volatile float d0 = 0, d1 = 0, d2 = 0, d3 = 0;
10
11 public:
12   volatile float x_predictor(float b_0, float b_1, float b_2, float a_1,
      float a_2, float input1, float Gain);
13
14 };
15
16 #endif
```

Listing 4: LS_Model.h header file.

References

[1] Wolfgang Klippel, "PREDICTIVE PROTECTION ARRANGEMENT FOR ELECTROACOUSTIC TRANSDUCER," *United States Patent*, June 1996.

[2] Wolfgang Klippel, "Assessment of Voice Coil Peak Displacement Xmax," *Audio Engineering Society Convention Paper*, April 2002.

[3] Wolfgang Klippel, "Mechanical Overload Protection of Loudspeaker Systems," *Journal of the Audio Engineering Society*, pp. 771–783, October 2016.

[4] Bruno Gazengel, "Duffing oscillator simulation," *IMDEA M2 Loudspeaker Modelling course instructor*, pp. 2–3, November 2023.

[5] Antonin Novak, "Digital Filtering, TD4a: Parametric Equalizers," *IMDEA M1 Digital Filtering course instructor*, pp. 7–8, Febraury 2023.

[6] Clément Richard, "Master 2 Project :Mechanical Overload Protection," *IMDEA M2 Project*, pp. 10–11, January 2023.

[7] K. GmbH, *Linear Parameter Measurement (LPM)*. [Online]. Available: https://www.klippel.de/products/rd-system/modules/lpm-linear-parameter-measurement.html (visited on 01/08/2024).

[8] K. GmbH, *Large Signal Identification (LSI)*. [Online]. Available: https://www.klippel.de/products/rd-system/modules/lsi3-large-signal-identification.html (visited on 01/08/2024).