Adaptive Filtering of Audio Signals and Proposed Applications within Neuromorphic Hardware

Ankur Sharma
University of California, San Diego
9500 Gilman Drive, San Diego, CA, 92122
ankursharma@ucsd.edu

Jonathan Somayajulu
University of California, San Diego
9500 Gilman Drive, San Diego, CA, 92122
jsomayaj@ucsd.edu

Abstract

Silicon cochlea is one of the most sought after products in the medical industry and neuromorphic engineering is one of the ways to achieve an efficient design for the same. However, implementing signal processing in hardware for low power applications is difficult and has several limitations because of the noise tolerance of the hardware components. In this project we looked for ways to implement adaptive filters to reduce the noise from devices whose noise fingerprint cannot be known before the process of chip fabrication. We look into the linear filter architecture (Kalman Filter), frequency domain adaptive filters (Freq. Domain Kalman Filter) and non linear filters as well (Collaborative Link Adaptive Filter). We then try to implement the same architecture in silicon and conclude on a design that can work for a specific device - the comparator.

1. Introduction

As edge devices for machine listening are becoming more ubiquitous, there is a push to perform audio signal processing under tighter power constraints on IoT devices. Mixed-mode approaches inspired by the biology of hearing offers a promising pathway for realizing efficient solutions for audio processing in-silico. Audio applications in silicon are highly susceptible to noise and with the decreased power constraint there is a need to look into real time adaptive filters that can counteract the noise.

One of the main noise sources in audio processing ICs is process noise. This is the noise that arises from various components on chip that do not behave as per the simulations due to the mismatch in transistor sizes. A primary culprit in this scenario is the comparator. It is a small element that makes the decision whether one signal is greater than the other (Fig 1.) [3]. The transistors M12 and M13 in Fig 1 contribute to this process noise the most and their mismatch can be catastrophic to the working of the chip.

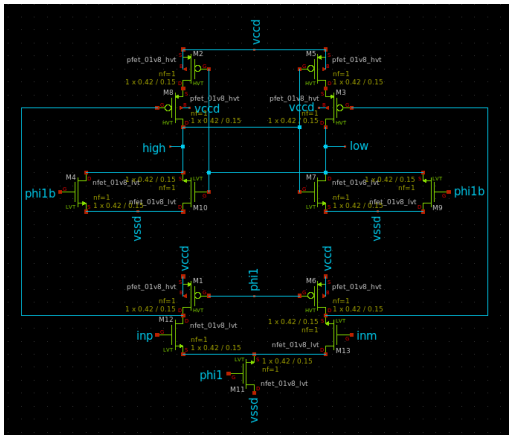

Figure 1. Example of model of comparator used

However, there is a way to fix the mismatch by applying an adaptive filter that can correct this process noise on board the chip. Considering that the mismatch can never be known before fabrication process is completed, and the each comparator will have it's own mismatch and it's own noise fingerprint, it is important that this filter be adaptive to the outputs of the comparator and correct the offset introduced by it periodically. This will also contribute towards removing any thermal noise from the comparator as well.

2. Background and Related Work

2.1. Adaptive noise cancellation algorithms

The current state of the art in adaptive audio cancellation are machine learning models that use extensive architectures [1]. The most common architectures use several hidden layers to be effective, and there is a steep tradeoff between good neural networks for noise filtering and them being usable in real time applications. For example Mozilla's rnnoise [2] is very fast and can be used in real time applications, but it is not very effective.

Hence, we look into various other types of filter architectures that we think can be translated to hardware and used for real time audio noise filtering.

2.2. Impulsive noise filter

Fig. 3 shows the impulsive noise filter [12]. The functioning of the filter is optimized for applications in FPGA and real time audio processing. We replicated this idea on Simulink to see if this would perform well in hardware. The idea is centered around estimating the distribution of noise adaptively but being wary of the outliers and not letting them affect the noise estimation.

The algorithm works in the following steps:

1. Calculation of y(n) for L = 8:

$$y(n) = \sum_{k=0}^{L-1} x(n-k).w_n(k) = \mathbf{x_n^t}.\mathbf{w_n} \quad (1)$$

2. Calculation of e(n):

$$e(n) = d(n) - y(n) \quad (2)$$

3. Calculation of β. As explained in [12], $\beta = 1/\sigma^2$. We then calculate the variance using the following equation:

$$\sigma^2(n) = 0.95 * \sigma^2(n-1) + \frac{e^2(n)}{n} \quad (3)$$

4. Now, we calculate the weights for the next iteration using the following equations:
   if $|e(n)| > \sigma^2$

$$w_{n+1}(k) = w_n(k) + \mu.sign[e(n)].x(n-k), \quad (4)$$

   if $|e(n)| <= \sigma^2$

$$w_{n+1}(k) = w_n(k) + \mu.\left[\frac{2}{\sigma^2} - \frac{|e(n)|}{(\sigma^2)^2}\right].e(n).x(n-k) \quad (5)$$

2.3. Time Domain: Kalman Filter

The time domain Kalman filter is a seminal filter design as one of the first adaptive filters invented [11]. This filter is comprised of hidden state variable (system) and observed variable (measurement) equations as shown in Figure 5 [8].

The process of Kalman filtering is summarized in the equations as shown in Figure 4 [8].

A major difficulty in the Kalman filter solution is the computation of the Kalman filter gain [8]. Under the classical Kalman filter design, the state error correlation matrix must be known to calculate the Kalman gain [8]. As this matrix is quite difficult to compute, the Riccati equations are used to solve for this matrix. This is a drawback of the classical Kalman filter design, as errors in the calculation of the Riccatti equations can cause instability in the Kalman filter solution [8].

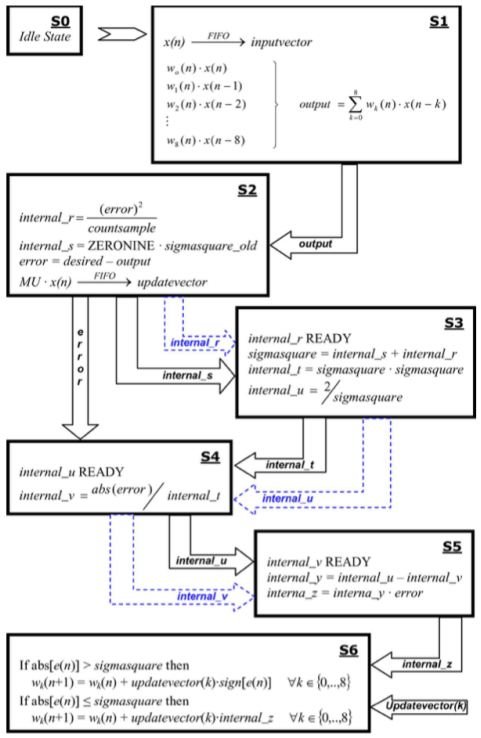

Figure 2. Impulsive Noise Filter Architecture [12]

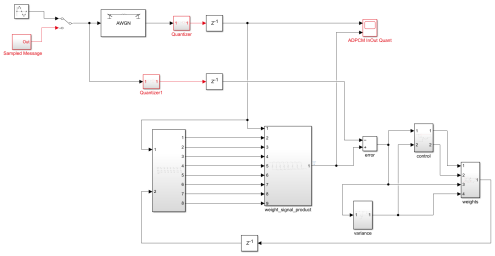

Figure 3. Simulink Implementation of Impulsive Noise Filter Architecture

2.4. Frequency Domain: Frequency Domain Kalman Filter

The frequency domain Kalman filter is an extension of the Kalman filter into the frequency domain, and the generalized architecture of this process is as shown in Figure 6 [6]. This is done by transforming the time domain Kalman filter by FFT to frequency domain, changing the weighting from time domain to frequency domain [7]. The properties of this filter are close to the time domain Kalman filter, but applying weights in the frequency domain provides better performance for frequency specific filtering [6]. However, this filter has the same shortcomings as the time domain Kalman filter, in that the measurement noise covariance as well as the Kalman gain should be well characterized for the filter to operate optimally [8].

Variable	Definition	Dimension
$\mathbf{x}(n)$	State at adaptation cycle n	M by 1
$\mathbf{y}(n)$	Measurement at adaptation cycle n	N by 1
$\mathbf{F}(n+1,n)$	Transition matrix from adaptation cycle n to adaptation cycle $n+1$	M by M
$\mathbf{C}(n)$	Measurement matrix at adaptation cycle n	N by M
$\mathbf{Q}_1(n)$	Correlation matrix of system noise $\boldsymbol{\nu}_1(n)$	M by M
$\mathbf{Q}_2(n)$	Correlation matrix of measurement noise $\boldsymbol{\nu}_2(n)$	N by N
$\hat{\mathbf{x}}(n\|\mathcal{Y}_{n-1})$	Predicted estimate of the state at adaptation cycle n given the measurements $\mathbf{y}(1), \mathbf{y}(2), \ldots, \mathbf{y}(n-1)$	M by 1
$\hat{\mathbf{x}}(n\|\mathcal{Y}_n)$	Filtered estimate of the state at adaptation cycle n, given the measurements $\mathbf{y}(1), \mathbf{y}(2), \ldots, \mathbf{y}(n)$	M by 1
$\mathbf{G}(n)$	Kalman gain at adaptation cycle n	M by N
$\boldsymbol{\alpha}(n)$	Innovations vector at adaptation cycle n	N by 1
$\mathbf{R}(n)$	Correlation matrix of the innovations vector $\boldsymbol{\alpha}(n)$	N by N
$\mathbf{K}(n,n-1)$	Correlation matrix of the error in $\hat{\mathbf{x}}(n\|\mathcal{Y}_{n-1})$	M by M
$\mathbf{K}(n)$	Correlation matrix of the error in $\hat{\mathbf{x}}(n\|\mathcal{Y}_n)$	M by M

Input vector process:
Measurements $= \{\mathbf{y}(1), \mathbf{y}(2), \ldots, \mathbf{y}(n)\}$

Known parameters:
Transition matrix $= \mathbf{F}(n+1,n)$
Measurement matrix $= \mathbf{C}(n)$
Correlation matrix of system noise $= \mathbf{Q}_1(n)$
Correlation matrix of measurement noise $= \mathbf{Q}_2(n)$

Computation: $n = 1, 2, 3, \ldots$

$$\mathbf{G}(n) = \mathbf{F}(n+1,n)\mathbf{K}(n,n-1)\mathbf{C}^{\mathrm{H}}(n)[\mathbf{C}(n)\mathbf{K}(n,n-1)\mathbf{C}^{\mathrm{H}}(n) + \mathbf{Q}_2(n)]^{-1}$$
$$\boldsymbol{\alpha}(n) = \mathbf{y}(n) - \mathbf{C}(n)\hat{\mathbf{x}}(n|\mathcal{Y}_{n-1})$$
$$\hat{\mathbf{x}}(n+1|\mathcal{Y}_n) = \mathbf{F}(n+1,n)\hat{\mathbf{x}}(n|\mathcal{Y}_{n-1}) + \mathbf{G}(n)\boldsymbol{\alpha}(n)$$
$$\mathbf{K}(n) = \mathbf{K}(n,n-1) - \mathbf{F}(n,n+1)\mathbf{G}(n)\mathbf{C}(n)\mathbf{K}(n,n-1)$$
$$\mathbf{K}(n+1,n) = \mathbf{F}(n+1,n)\mathbf{K}(n)\mathbf{F}^{\mathrm{H}}(n+1,n) + \mathbf{Q}_1(n)$$

Initial conditions:

$$\hat{\mathbf{x}}(1\,|\,\mathcal{Y}_0) = \mathbb{E}[\mathbf{x}(1)]$$
$$\mathbf{K}(1,0) = \mathbb{E}[(\mathbf{x}(1) - \mathbb{E}[\mathbf{x}(1)])(\mathbf{x}(1) - \mathbb{E}[\mathbf{x}(1)])^{\mathrm{H}}] = \Pi_0$$

Figure 4. Kalman Filtering Process as described in Haykin[8]

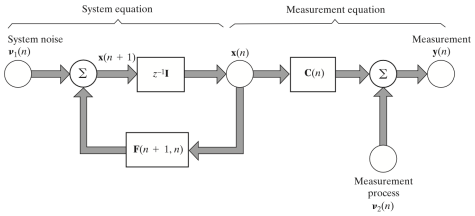

Figure 5. Kalman Filter [8]

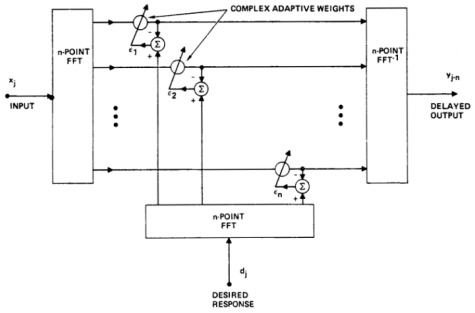

Figure 6. Frequency Domain Based Filter Methodology [6]

2.5. Nonlinear filter: Collaborative Functional Link Adaptive Filter

We also tested an collaborative functional link filter (CLAF) as a non-linear design extension of the Kalman filter, as is shown in Figure 7 and recently proposed in Comminiello et al. [5]. A collaborative functional link filter differs from the previous filter designs in that it combines a nonlinear functional link adaptive filter and linear adaptive filter design into one filtering process that operates in tandem [5]. The benefit of using this architecture on acoustic noise is that the filter response can be adjusted depending on the input noise estimate as the non-linear and linear filters can be turned off and on via weighting, allowing for a truly adaptive noise estimation and subsequent cancellation [5].

A more recent generalized version of the CLAF filter architecture as proposed by Zhang et al. is shown in Figure 8 [15].

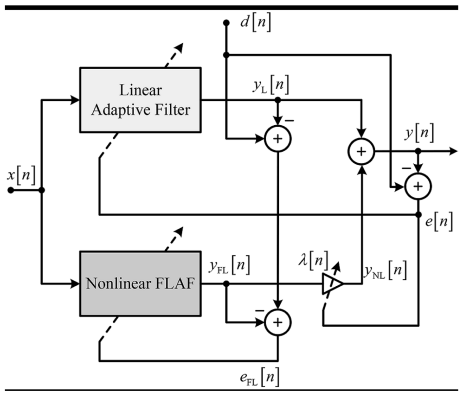

Figure 7. Collaborative Functional Link Filter [5]

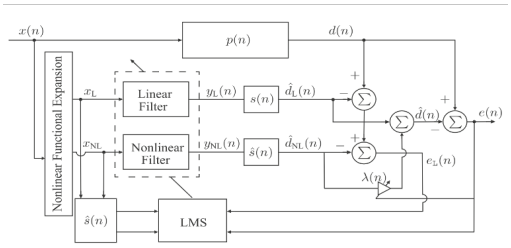

Figure 8. A generalized version of a Collaborative Functional Link Filter [15]

2.6. Hardware feedback adaptive filter

Considering all the above designs, we observed that these designs are very complicated to be implemented as a simple module on an IC. Adding a single filter to each comparator requires excessive space and high power consumption. Hence, we use a method that is based on a few digital cells in the circuit and the implementation is as follows:

1. We use the outputs of the comparator on a known that fires 50% of the time to find the tilt of the comparator.
2. This is done using a counter that counts in gray fashion to conserve energy.
3. The counter is 16 bits long and starts from

'1000000000000000'. If we see a series of zeros we go down and if we see a series of ones, we go higher.

4. Once this counter has settled, we send a signal to multiply the multiplexed gray output and send it through a low pass filter to remove all high frequencies.

5. This smooths out the feedback and gives us an analog signal as we needed.

This process corrects the offset and removes the noise introduced by the comparator. It consumes 90 femto Joules of energy per clock cycle of 2.56 MHz. Fig 10. shows this system working.

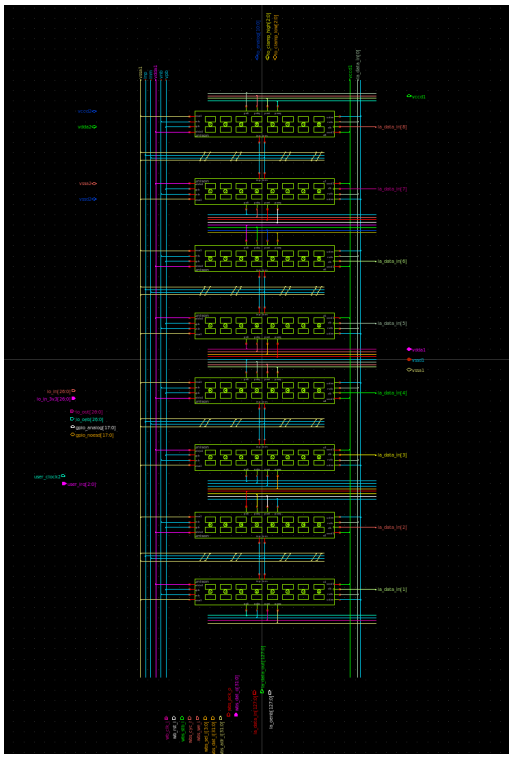

Figure 9. Top level design of the chip. Each small 'H' block used one comparator.

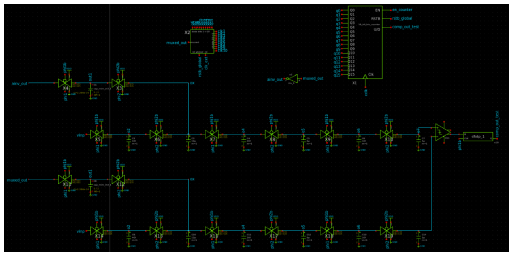

Figure 10. Feedback Design

3. Methods

3.1. Exploratory Software Adaptive Filter Test in Python

As the methods for the hardware and Simulink design implementations are discussed in prior sections, the methods section will focus on the software implementation.

In order to run the filters on real world data, we use audio from Noizeus [9, 10] and process this dataset using a Python based adaptive filter implementation, passing each audio sample through the adaptive filters described above [14]. This website has calibrated audio files for 30 different audio sample types and various real world noise sources such as in exhibition halls, trains and supermarkets. For our simulations, we tested the restaurant, babble, train, supermarket, and exhibition hall samples at 3 different noise levels: 10dB, 5dB and 0dB SNR. For the Kalman filter, 64 weights were used. The frequency domain Kalman filter and collaborative functional link filter used default parameters from the implementation for the sake of consistency.

For each filtered audio example, in order to quantify noise reduction, we calculated the signal to noise ratio by finding the average power of the additive noise and dividing the average power of the denoised signal by this value.

4. Results and Discussion

4.1. Results from software based filters

This summarizes our results from our exploratory experiment testing a variety of adaptive filtering techniques. A selection of results is presented below, specifically for the exhibition hall noise signal. For each figure, the original noisy signal's spectrogram is shown above the denoised spectrogram.

It is worth noting that for audio applications a more qualitative listening assessment of audio quality may be suitable to include in certain circumstances, as purely characterizing audio fidelity in SNR may not be suitable to adequately express audio quality to the listener. For instance, a SNR level may be deceptively passable whilst in reality the original signal has been comprised and is unpleasant to listen to. Hence, for each test in this exploratory experiment, we will include a qualitative assessment of the overall audio quality.

With regards to the Kalman time domain filter shown in Figures 11-13, based on the spectrogram results, it appear that the noisy audio was denoised quite well across all SNR levels. This is audibly apparent in the denoised audio as well, as the audio is noticeably cleaner. This is also backed by the measured SNR levels of 14.47 dB, 15.48 dB, and 18.29 dB for the noisy SNRs of 0 dB, 5 dB, and 10 dB.

For the frequency domain Kalman filter shown in Figures 14-16, the spectrogram for each SNR level appears less well denoised. The denoising algorithm appears to have

overly denoised the noisy signal to the point that the fidelity of the signal has been comprised, which is audible to the listener. This is apparent in the SNR measurements of the denoised signal, which are -1.85 dB, -0.370 dB, and 2.87 dB respectively in comparison to the noisy signal SNRs of 0 dB, 5 dB, and 10 dB.

The collaborative functional link filter shown in Figures 17-19 performed quite similarly to the Kalman time domain based filter and better than the frequency domain based Kalman filter, as evidenced by the spectrogram results across all SNR levels. This is apparent in the SNR measurements of the denoised signal, which was consistency 17.20 dB for each sample in comparison to the noisy signal SNRs of 0 dB, 5 dB, and 10 dB. This indicates that this filter architecture is quite stable across all SNRs. However, given the complexity of this implementation, the tradeoff of filter complexity to performance may not be suitable, especially for hardware oriented design.

Based on these results, we found that the collaborative adaptive filter and Kalman filter operated the best on the dataset used. However, as described previously, a major issue with the Kalman filter is that the Kalman gain must be derived correctly and known for the Kalman filter to behave effectively [8, 13]. In cases where the statistics of the noise process are continually changing, the performance of the Kalman filter may be highly inconsistent [8, 4]. Subsequent modern versions of the Kalman filter and beyond attempt to address this shortcoming, such as with the architecture of CFLAF incorporating both nonlinear and linear filter responses [5].

A alternative implementation of non-linear systems would utilize neural networks, however for practical hardware implementation (especially within IC designs) this is not possible due to the constraints such as board size and component amounts imposed by design in hardware [5]. As such this was excluded in our testing.

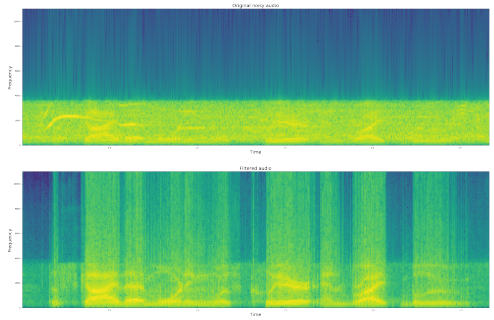

Figure 11. Kalman Time Domain Filtered Spectrogram of signal with noise of Exhibition Hall (SNR: 0 DB)

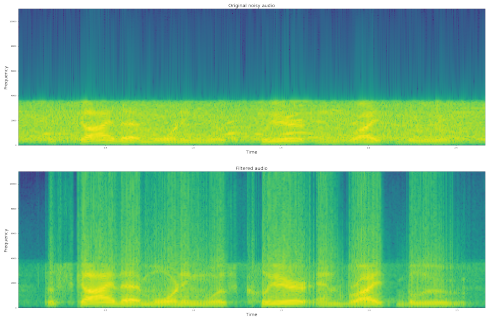

Figure 12. Kalman Time Domain Filtered Spectrogram of signal with noise of Exhibition Hall (SNR: 5 DB)

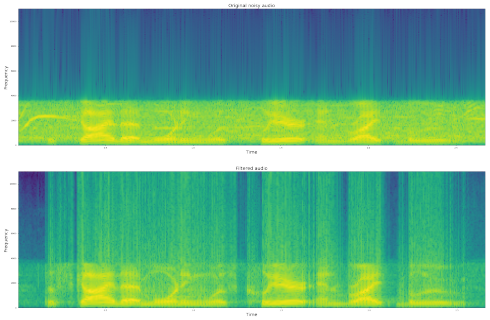

Figure 13. Kalman Time Domain Filtered Spectrogram of signal with noise of Exhibition Hall (SNR: 10 DB)

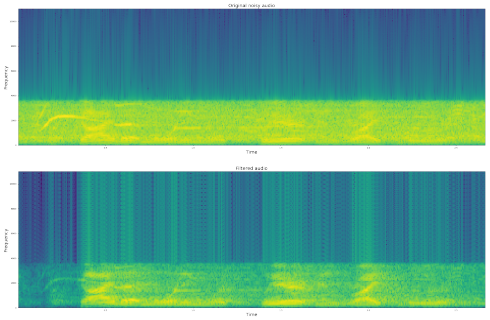

Figure 14. Frequency Domain Filtered Spectrogram of signal with noise of Exhibition Hall (SNR: 0 DB)

4.2. Results from Simulink models

When attempting to transfer these software based filters to hardware, it was found that these architectures did not translate quite well, as the physical size constraints associated with IC design make it difficult for these filters to be directly applied. Thus, the architecture described in Rosado et al. was tested in Simulink, as this was a filter designed for hardware implementation. The results of this design are shown below in Fig. 20.

As shown in Fig. 20, the noise is subdued in the output and the output is observably phase shifted due to the delays

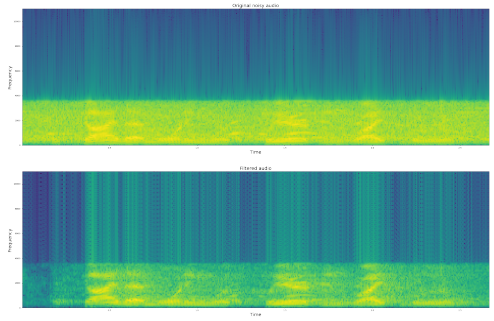

Figure 15. Frequency Domain Filtered Spectrogram of signal with noise of Exhibition Hall (SNR: 5 DB)

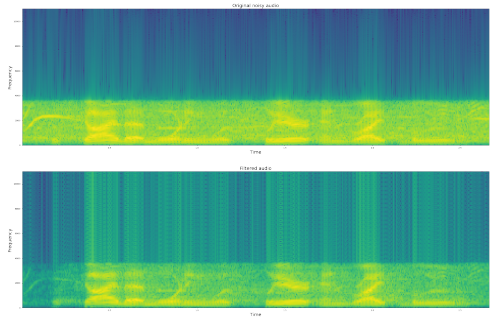

Figure 16. Frequency Filtered Spectrogram of signal with noise of Exhibition Hall (SNR: 10 DB)

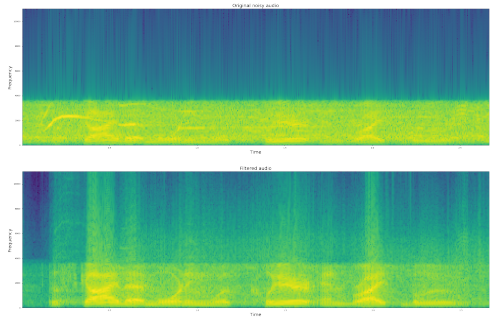

Figure 17. Collaborative Functional Link Filtered Spectrogram of signal with noise of Exhibition Hall (SNR: 0 DB)

added strategically.

4.3. Results from Hardware implementation

Figure 21 shows the corrected offset in the system which is the required functionality for the feedback as expected. With an intentional offset of 50mV in the comparator sizing, we were able to achieve the exact offset that we wanted to compensate. This is a dummy implementation of a real world scenario that we expect to encounter. The 50mV offset compensation doesn't seem too much, but with that offset, the entire IC will not work at all. So it is paramount to our application to have this offset corrected through an

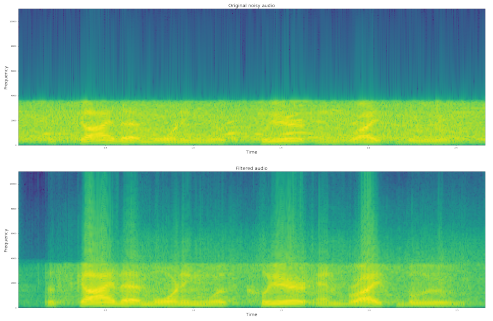

Figure 18. Collaborative Functional Link Filtered Spectrogram of signal with noise of Exhibition Hall (SNR: 5 DB)

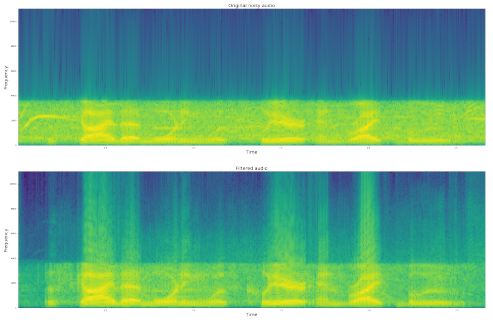

Figure 19. Collaborative Functional Link Filtered Spectrogram of signal with noise of Exhibition Hall (SNR: 10 DB)

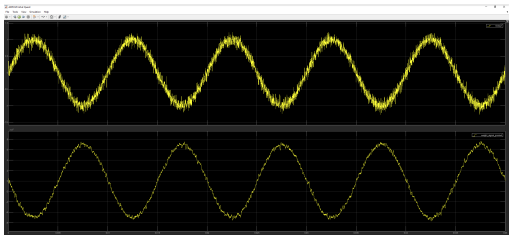

Figure 20. Results from Simulink model

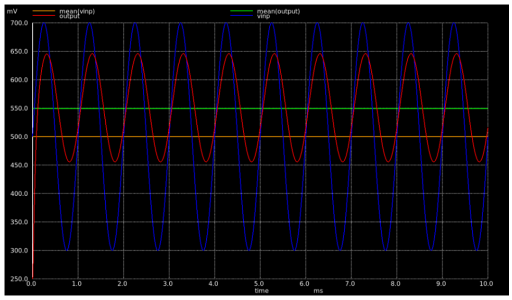

Figure 21. Modified threshold using adaptive feedback

adaptive filter which we achieved here.

4.4. Conclusion

This project was designed to perform research into effective adaptive filter implementations within hardware to-

wards usage in neuromorphic, silicon based cochlear design, intending to test if software based methods underlying adaptive filter designs can translate seamlessly into hardware applications. From our exploratory testing, we find that software defined adaptive filters do not directly translate well into a hardware based design, especially for IC usage. The difficulty of determining noise characteristics of an dynamic audio signal make a direct implementation intractable. As the Simulink implementation of the adaptive filtering for hardware demonstrates, the filter architecture must be heavily modified from software to hardware for it to be functional [12]. Hence, the inference from these two experiments was adapted into the design of the hardware feedback adaptive filter architecture, which used digital cells to adaptively tune thermal component noise from the comparator out of the input audio sequence, removing feedback and effectively denoising the audio signal. Further study is necessary to analyze potential confluences where hardware and software based filter design can overlap to further innovation in this respect.

References

[1] Real-time noise suppression using deep learning. https://developer.nvidia.com/blog/nvidia-real-time-noise-suppression-deep-learning/. Accessed: 2022-05-10.

[2] Rnnnoise by mozilla. https://github.com/xiph/rnnoise. Accessed: 2022-05-10.

[3] Harijot Singh Bindra, Christiaan E Lokin, Daniel Schinkel, Anne-Johan Annema, and Bram Nauta. A 1.2-v dynamic bias latch-type comparator in 65-nm cmos with 0.4-mv input noise. *IEEE journal of solid-state circuits*, 53(7):1902–1912, 2018.

[4] R. Bucy, R. Kalman, and I. Selin. Comment on "the kalman filter and nonlinear estimates of multivariate normal processes". *IEEE Transactions on Automatic Control*, 10(1):118–119, 1965.

[5] Danilo Comminiello, Michele Scarpiniti, Luis A. Azpicueta-Ruiz, Jerónimo Arenas-García, and Aurelio Uncini. Functional link adaptive filters for nonlinear acoustic echo cancellation. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(7):1502–1512, 2013.

[6] M. Dentino, J. McCool, and B. Widrow. Adaptive filtering in the frequency domain. *Proceedings of the IEEE*, 66(12):1658–1659, 1978.

[7] ÅsTRÖM K. J. GRIMBLE, M. J. Frequency-domain properties of kalman filters. *International Journal of Control*, 45(3):907–925, 1987.

[8] Simon Haykin. *Adaptive Filter Theory*. Pearson, 5th edition, 2014.

[9] Y. Hu and P. Loizou. Noizeus. https://ecs.utdallas.edu/loizou/speech/noizeus/ note = Accessed: 2022-05-10.

[10] Y. Hu and P. Loizou. Subjective evaluation and comparison of speech enhancement algorithms. *Speech Communication*, 49:588–601.

[11] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[12] Alfredo Rosado-Muñoz, Manuel Bataller-Mompeán, Emilio Soria-Olivas, Claudio Scarante, and Juan F Guerrero-Martínez. Fpga implementation of an adaptive filter robust to impulsive noise: Two approaches. *IEEE Transactions on Industrial Electronics*, 58(3):860–870, 2009.

[13] Manuela Vasconcelos. Kalman filter lectures 1,2, and 3, May 2022.

[14] Ewan Xu and Jason Zhang. pyaec. https://github.com/ewan-xu/pyaec, 2021. Accessed: 2022-05-10.

[15] Shujie Zhao, Lijun Zhang, and Wenxia Lu. A generalized collaborative functional link adaptive filter for nonlinear active noise control. *Applied Acoustics*, 175, 2020.