RANC: Real-time Adaptive Noise Cancellation

Ashwin Karthik Tamilselvan (at3103)
Gikku Stephen Geephilip (gg2624)
Richa Glenn Netto (rn2388)
Rishikanth Chandrasekaran (rc3022)

1. Introduction

We propose to implement an adaptive noise canceller that can filter out noise from contaminated sources in real-time. External sound is ubiquitous; an important functionality in audio related applications is to filter out unwanted sound (also termed as noise) to provide high quality audio. The experimental setup will demonstrate functionality similar to a noise cancelling headphone.

2. General Design

The system consists of the following modules: speakers, 2 mics, SoCKit FPGA board and a VGA display. The speakers are used to play the music and mics are used for recording external noise as a reference input. The mic input is recorded using a MATLAB program that records the noise, stores it and sends it to the FPGA for processing. The FPGA includes functionalities that are divided and associated specifically to hardware and software. The software system running on the FPGA is a linux OS with programs to decode the audio input obtained from the PC using appropriate audio codecs. The decoded audio is then fed to the FPGA hardware component on which the noise cancellation algorithm is implemented on. Noise cancellation is done using the Fast Least Mean Square algorithm which is implemented on the FPGA using a state machine. The output of the algorithm is handed over back to the software which encodes it into the required audio format using the appropriate audio codec. This output is then played out via the speaker. Along with the audio output, the weights that are calculated in real time for the LMS algorithm is displayed on the VGA display. The magnitude and the sign of the weights are displayed on the VGA display in the form of bars. Figure 1 shows the block diagram of this implementation.

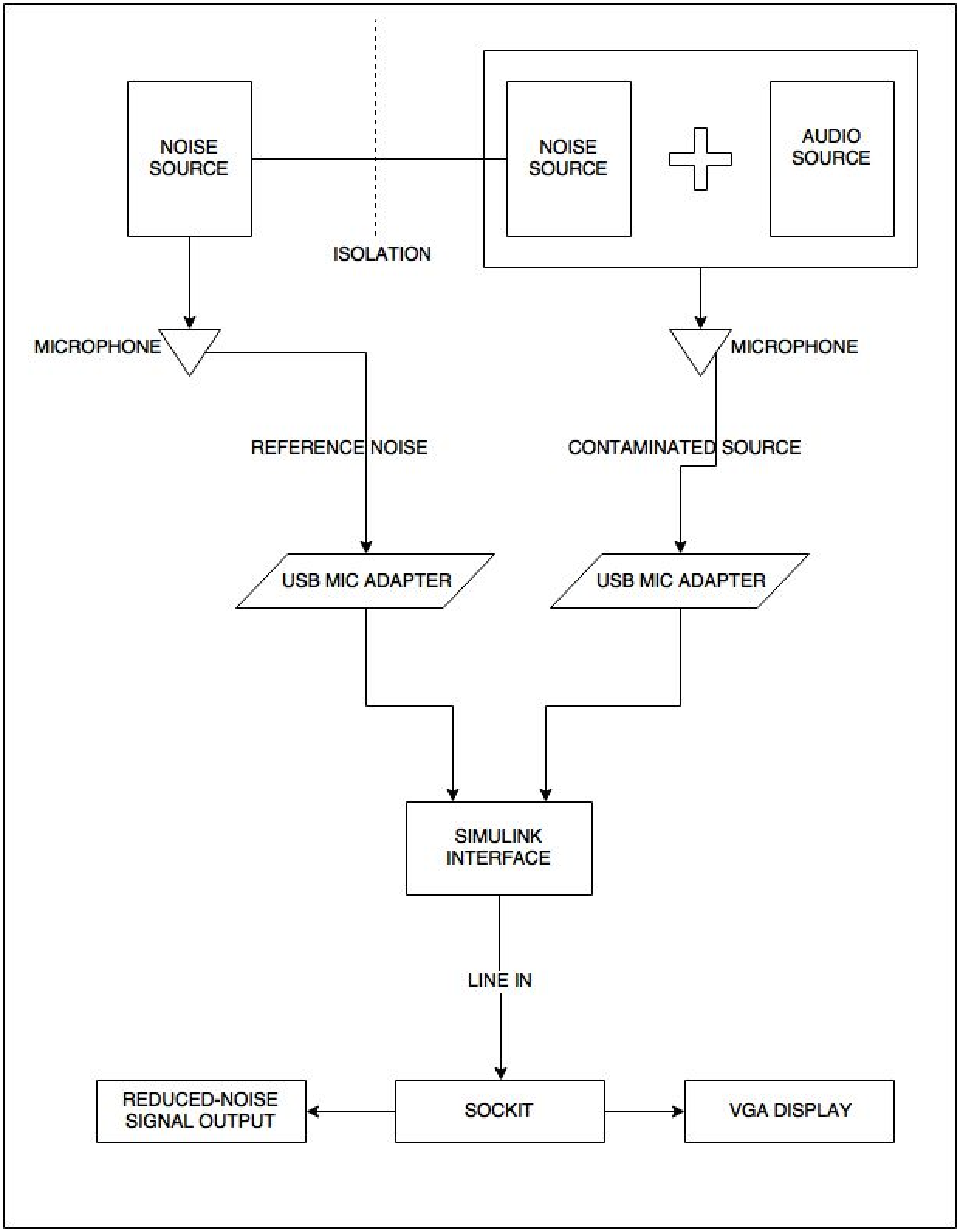

Fig 1: Block Diagram of the experimental setup

3. Algorithms - Least Mean Square & Fast Least Mean Square Algorithm

   We will be using the Least Mean Square Algorithm in this project, mainly owing to the its computational simplicity and ease of implementation. The LMS algorithm is a class of adaptive filter used to mimic a desired filter by adjusting the filter coefficients in such a way that it produces the least mean square of the error signal, ie, if we consider the error to be a cost function, the LMS algorithm basically finds filter coefficient values that can minimise the cost function. The error signal here is the difference between the desired and the actual signal. We will implement an adaptive filter on the FPGA whose weights will be computed in real time, along with noise cancelling schemes to verify the performance of the adaptive noise canceller.

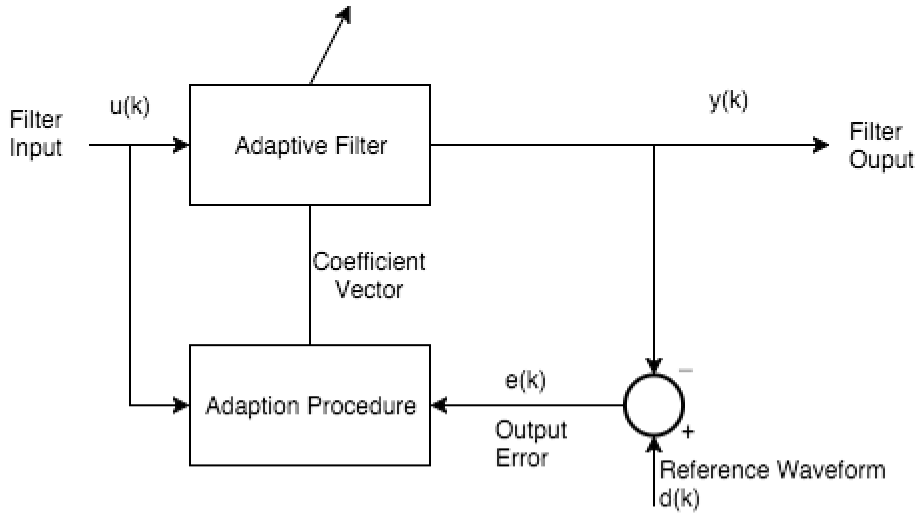

Fig 2: Block Diagram of Adaptive Filter using LMS

The signals shown in Fig 2 are described as follows:

1. u(k) is the input vector
2. d(k) is the reference waveform, ie, the desired response
3. y(k) is the output of the filter, which gives us an estimate of the desired response. It is calculated as the convolution of the weight vector and input vector.
4. e(k) is the estimated error, which is the difference between the output of the adaptive filter (ie, estimated response) and the desired response

The following equations give the basic structure of relations in the LMS algorithm:

I. Filter Output: $y(k) = u^T W$ (where W is a vector of weights applied to the filter coefficients, and T is the Transpose operation)
II. Estimated Error: $e(k) = d(k) - y(k)$
III. Weight Update: $W_{k+1}=W_k+2\mu*e(k)*u_k$

The estimation error, along with the input vector are applied to the feedback adaptive procedure.

μ is the step size, and is inversely proportional to the settling time constant of the convergence behaviour. For small step size values, the adaptive process slows down but the mean-square error is minimised, and vice-versa for large values of step size. The selection of the value of step size is extremely important as it greatly affects how the algorithm updates the filter coefficients. Generally, for stable behaviour and convergence of the LMS algorithm, the step size must be a small, positive value.

The noise canceller is implemented using *fast least mean square algorithm*. The fast LMS algorithm reduces the number of multipliers needed to implement weight adaption in hardware, thereby reducing the resources required for hardware implementation. The weight update expression used by fast LMS is as follows:

$$W_{k+1}=W_k+e(k)*\text{sign}(u_k)>>>n$$

The fast LMS algorithm replaces step size with a shift operation, and n represents the number of shifts. Another difference between LMS and fast LMS is that fast LMS uses only the sign bit of the reference input, u(k), instead of using its value. This is done to significantly reduce the number of multiplications, therefore, it simplifies the implementation of the LMS filter. However, there is an inaccuracy of the adaptation process caused by this simplification that degrades the learning performance.

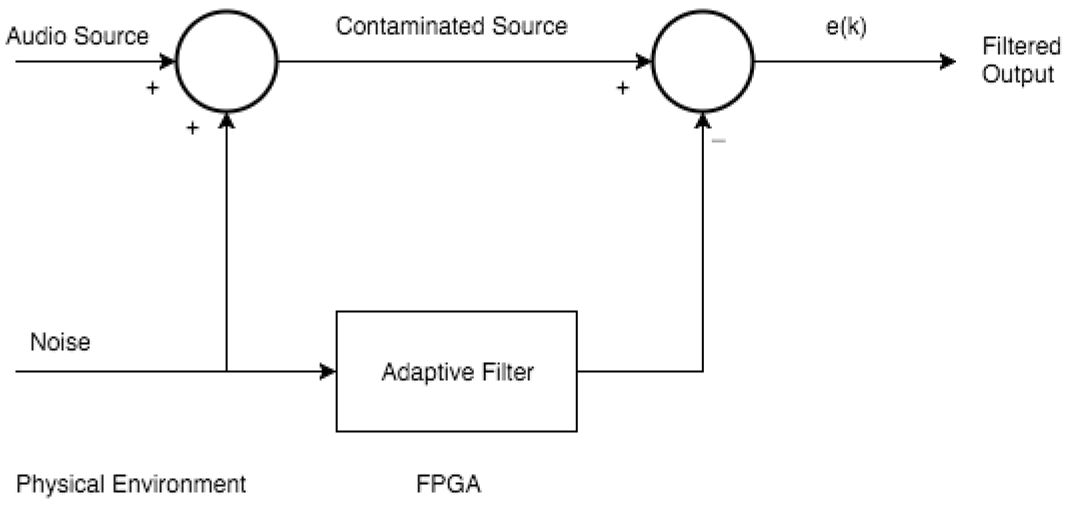

Fig 3: Block Diagram of Adaptive Noise Canceller

4. SoCKit

The FPGA implementation consists of software and hardware components each programmed to perform specific operations to coordinate the system. The software component handles interfacing with the PC to obtain the input data which is the contaminated sound data. This is then decoded using the audio codec implemented on the linux platform running on the ARM core. The processor then feeds the pre-processed bits of data to the FPGA hardware which runs a state machine to handle the different processing events. The weights are calculated in real-time according to the data which is then used for running LMS. The computed weights are then fed to the VGA handler which displays the weights on the VGA display. The implementation overview is shown in Figure 4.

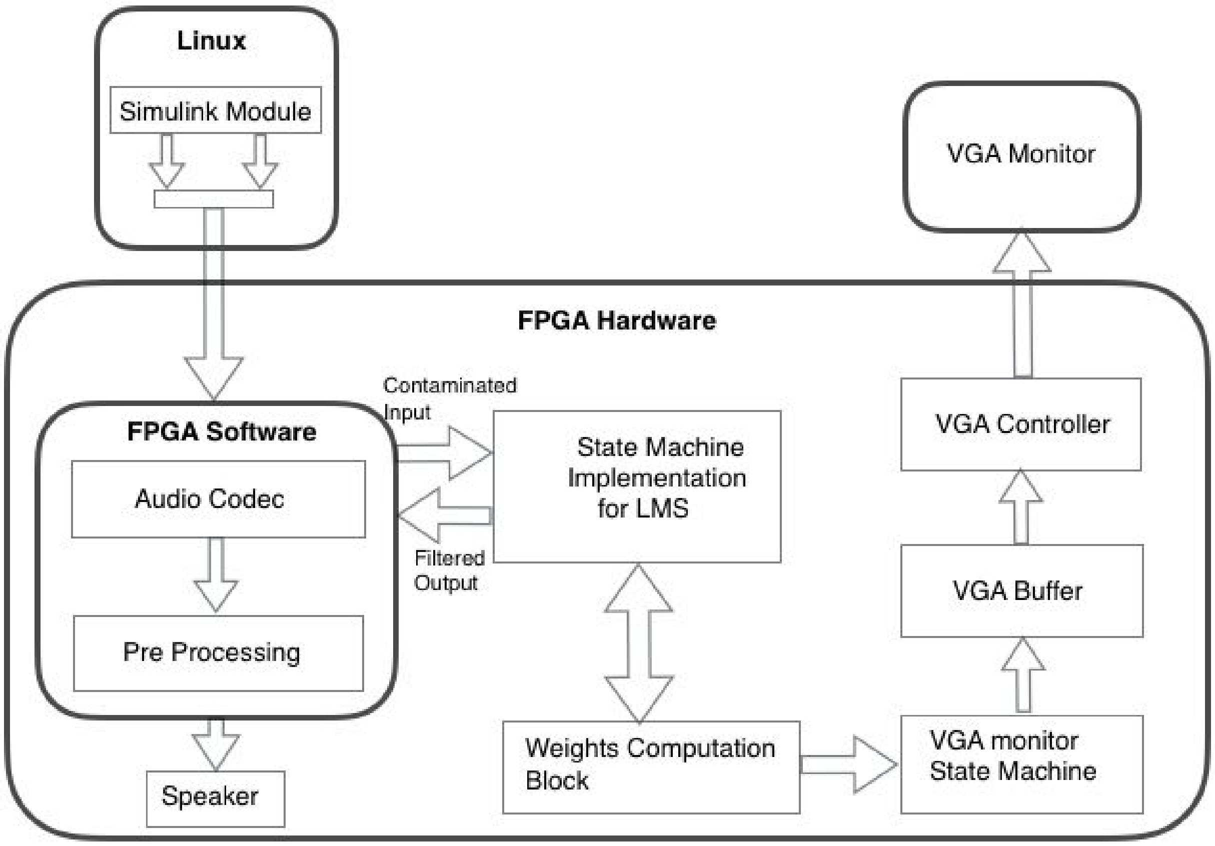

Fig 4: Block Diagram of Adaptive Noise Canceller

5. VGA

   We plan to implement a VGA display that shows the filter weights. The weights are displayed using their magnitude and sign bit. This information is further processed and each bit of the VGA display is set according to the obtained weights of the adaptive filter thus designed.

6. Milestones

Milestone 1:

1. Simulate and verify the correctness of the algorithm on MATLAB
2. Finalize on the hardware components required

Milestone 2:

1. Finalize the audio codecs required to decode the input and reference signals
2. Implement the audio codecs in software
3. Interfacing with audio jack on SoCKit for audio output

Milestone 3:

1. Implement the adaptive noise filter in hardware.
2. Verify the result of the hardware implementation with the software test run and make necessary adjustments
3. Display the filter weights on VGA

Final Project:

The aim is to be creating the final presentation at this stage, but this could also be used as buffer time to implement anything that is leftover from a previous stage.

References:

1. http://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2011/jy554_jc2636/jy554_jc2636/_fpgaimplementation.htm
2. http://publik.tuwien.ac.at/files/pub-et_9883.pdf
3. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&cad=rja&uact=8&ved=0ahUKEwiiu7PK8drLAhXB9h4KHVDFAsEQFggiMAE&url=https%3A%2F%2Fwww.asee.org%2Fpublic%2Fconferences%2F20%2Fpapers%2F6147%2Fdownload&usg=AFQjCNGsH-hDjoU1HvvNJilVyekswtFOYg
4. https://en.wikipedia.org/wiki/Least_mean_squares_filter