REAL TIME ADAPTIVE NOISE CANCELLATION ON A FPGA

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

1. Overview

Our project involves implementation of an adaptive noise canceller that takes environmental noise as an input and reduces it in real-time. External sound is ubiquitous; an important requirement of certain environments such as vehicles and aircrafts, is to minimize the noise external to the system in consideration. We have implemented our project using an adaptive filter based on the fast-LMS algorithm and interfaced the FPGA with hardware peripherals, a microphone and a speaker, to capture the noise and play the reduced noise result respectively. We have utilized the audio codec on the FPGA to process the sound signals.

2. General Description

a. Adaptive Noise Cancellation

Noise cancellation has always been one of the most fascinating and consumer market-driven area of research. In an ideal setup, an efficient communication is one in which the complete message produced from the source is reproducible in the destination. An ideal setup would be expected to be free of any extra noise that could disrupt the communication leading to the loss of a part of the message or the complete message transmitted. But an ideal setup is very difficult to achieve practically. Communication has been an integral part of the human life and have been utilized in various forms. Media transmission is one such widely used type of communication.

A special case of audio transmission has been discussed in this project. Noise Cancellation is a technique developed with an aim of providing the best possible transmission of audio signals without considerable loss of date or disruption of the original signal. Noises are an inherently present as part of the environment. Noise cancellation in theory would aid in the replication/reproduction of the audio signal in the same state, it was transmitted from the source. Various techniques have been developed over time and adaptive noise cancellation is a popular technique used in the market.

Adaptive Noise Cancellation primarily deals with the generation of anti-noise which in theory would cancel out the ambient noise. Adaptive noise cancellation can be broadly divided into three different setups depending on the position of microphone/s, with each setup having its advantage over the other - Feedforward Noise Cancellation, Feedback Noise Cancellation and Hybrid Noise Cancellation.

We have selected Feedforward Noise Cancellation to realize the working of Adaptive Noise Cancellation on a SoCkit Cyclone V board. In this setup, a microphone is placed outside the ear cup of a headphone giving the advantage of extra response time over the Feedback Noise Cancellation setup. This implementation of ANC fares better at reducing high frequency noise up to 1-2 kHz. We have used a single microphone and an audio output(speakers/headphones) in our setup. We have modelled the system such that the microphone feeds the environment/ambient noise as the input to the system and a reduced noise is obtained at the output. The anti-noise produced is superimposed on the input noise signal. This leads to the partial cancellation of the noise signal resulting in a reduced noise signal.

b. Least Mean Square Algorithm:

We have used the Least Mean Square Algorithm in this project, mainly owing to its computational simplicity and ease of implementation. This algorithm is based on the underlying concept of Least Mean Squares and the Steepest Descent Algorithm, however, it eliminates the need for exact measurements of the gradient vector and does not involve matrix inversion. 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 minimize the cost function. The error signal in this case is the difference between the desired and actual signal. Our project implements an adaptive filter using a more efficient variation of the LMS algorithm to update the weights of the filter in real-time while simultaneously ensuring that the computational load on the FPGA is reduced.

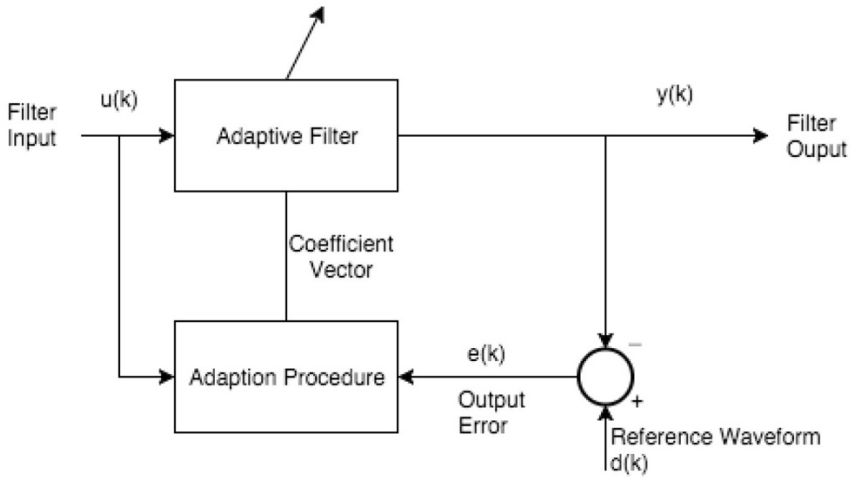


Fig X: Block Diagram of Adaptive Filter using LMS Algorithm

The signals shown in the Fig X are described as follows:

i. u(k) is the input vector (pure noise signal)
ii. d(k) is the reference signal, ie, the contaminated input signal
iii. y(k) is the output of the adaptive filter, which gives us anti-noise or reduced noise
iv. e(k) is the estimated error obtained by subtracting the anti-noise from the contaminated signal. This is essentially the final reduced-noise output of the system.

The following equations define the working of the LMS algorithm:

i. Filter Output: $y(k) = u^T W$
   where, W = vector of weights applied to the filter coefficients
   T = 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 estimated error and the input noise vector are applied as feedback to the adaptive procedure.

μ is the step size and is inversely proportional to the settling time constant of the convergence behavior. For smaller values of step size, the adaptive process slows down but the mean-square error is minimized, ie, it takes longer to converge but the results are better. Similarly, for larger values of step size, the adaptive process converges quickly, but the results are not as good as those obtained with a smaller step size.

c. Fast-LMS Algorithm

In our system, the adaptive filter is implemented using the Fast-LMS Algorithm. It follows the following equation for updating the weights:

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

The Fast-LMS algorithm replaces step size with a shift operation, where n represents the number of shifts. Also, Fast-LMS only uses the sign bit of the reference input, u(k), instead of using its value.

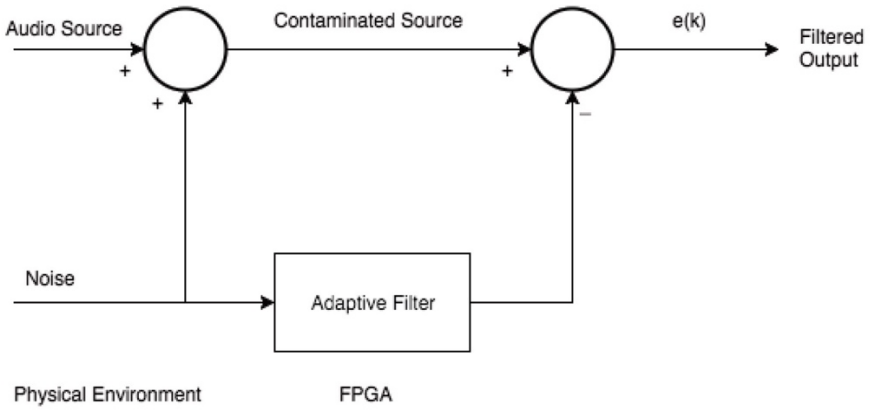


Fig X:

d. Audio Codec:

To record and play audio on the FPGA, we have to build an interface to the audio codec. This bidirectional interface will enable us to capture audio from and stream audio to the hardware peripherals, such as microphones and speakers.

When sound travels through the air medium, the microphone perceives it as pressure waves that create alternating areas of compression and rarefaction as it travels from the source to the microphone. The microphone then converts the pressure waves into an electrical signal, using a voltage level to represent the pressure at a point in the wave. The voltage levels are continuous in time, so now we have analog signals, which the FPGA cannot process since it accepts digital quantities and runs at a fixed clock rate. The audio codec on the FPGA converts the analog voltage values to discrete quantities by capturing voltage levels at regular intervals based on a sampling rate, and then performing analog-to-digital conversion (ADC) to discretize the captured voltage levels. In the reverse process, when we want to play an audio signal from the FPGA through the speakers, the codec performs digital-to-analog (DAC) conversion, which results in a slight loss of information due to reconstruction of a continuous signal from a bunch of discrete values.

The audio codec on this FPGA (Analog Devices SSM2603) is configured with parameters such as sampling rate and sampling width using the I^2C protocol.

3. Setup

a. Adaptive Noise Canceller – Physical Setup

The setup of our project requires one microphone, one speaker and a noise source. We are using a white noise source played out of a cell phone as the noisy input. The microphone is held near the noisy source and is plugged into the mic-in port on the FPGA. The line-out port on FPGA is connected to a speaker and it gives the reduced noise as an output.

b. Fast-LMS

We have decided to use Fast-LMS instead of LMS for our project because Fast-LMS only uses the sign bit of the input in the formula where weights are updated and hence, it significantly reduces the number of multiplications required and simplifies the implementation of the LMS filter in hardware. It also replaces step size with a shift operation, further simplifying the algorithm.

We have modified the block diagram of Adaptive Noise Cancellation to account for the fact that our input to the system is noise (in the inherent presence of silence) recorded through a microphone. The Adaptive Filter also gets the same noisy input, and it generates a reduced noise which is represented in the diagram as Filter Output. Then, we subtract the Filter Output from the Noisy Input to generate the error output, which is the reduced noise signal. This is then sent back to the adaptive filter so that it can be used to update the weights of the filter coefficients. Finally, the reduced error output is sent out via a speaker.

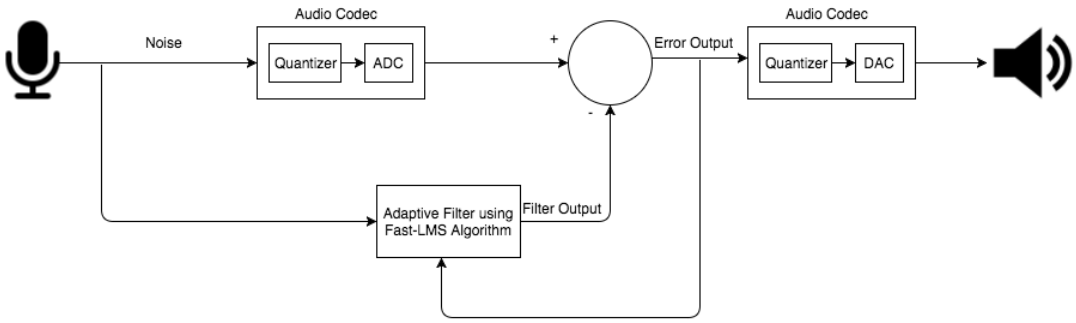


Fig X:

c. Audio Codec

The SSM2603 Audio Codec takes 16-bit data words. Each transmission will include 3 acknowledgements before the stop symbol must be sent. The maximum frequency of SCLK is 526 kHz. SCLK is divided down by 128, giving a frequency of

about 390 kHz. The codec only checks for acknowledgements after all 24-bits (including address, data and r/w bit) are transmitted. The ack signal is set high if all the acknowledgements have been received.

Using the data sheet of the SSM2603 Audio Codec, we have assigned values to the registers that we require in our system. The audio codec organizes its configuration variables into 19 9-bit registers. The first seven bits of the transmitted data are register address, the last 9 bits are the register contents.

Register	Register Name	Value
R0	**Left Channel ADC Input Volume**	**000010111**
R1	**Right Channel ADC Input Volume**	**000010111**
R2	**Left Channel DAC Volume**	**001111001**
R3	**Right Channel DAC Volume**	**001111001**
R4	**Analog Audio Path**	**011010100**
R5	**Digital Audio Path**	**000000100**
R7	**Digital Audio Interface**	**000000001**
R8	**Sampling Rate**	**000100000**

REGISTER MAP

Table 10. Register Map

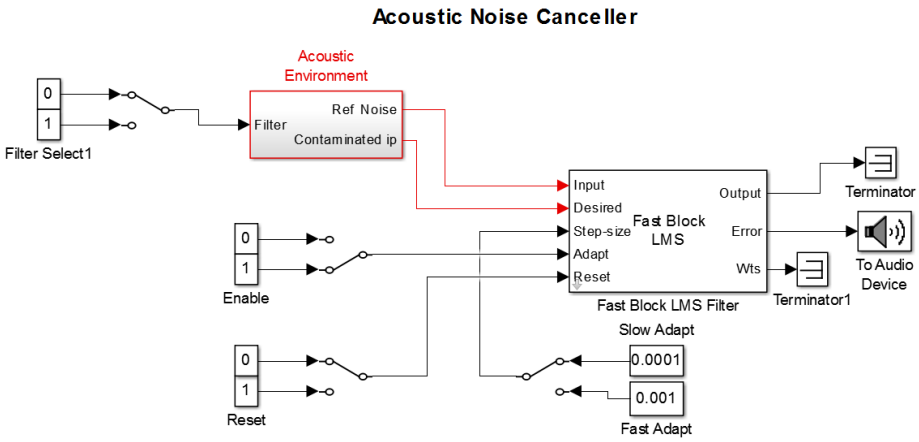
Reg.	Address	Name	D8	D7	D6	D5	D4	D3	D2	D1	D0	Default
R0	0x00	Left-channel ADC input volume	LRINBOTH	LINMUTE	0	LINVOL[5:0]						010010111
R1	0x01	Right-channel ADC input volume	RLINBOTH	RINMUTE	0	RINVOL[5:0]						010010111
R2	0x02	Left-channel DAC volume	LRHPBOTH	0	LHPVOL[6:0]							001111001
R3	0x03	Right-channel DAC volume	RLHPBOTH	0	RHPVOL[6:0]							001111001
R4	0x04	Analog audio path	0	SIDETONE_ATT[1:0]		SIDETONE_EN	DACSEL	Bypass	INSEL	MUTEMIC	MICBOOST	000001010
R5	0x05	Digital audio path	0	0	0	0	HPOR	DACMU	DEEMPH[1:0]		ADCHPF	000001000
R6	0x06	Power management	0	PWROFF	CLKOUT	OSC	Out	DAC	ADC	MIC	LINEIN	010011111
R7	0x07	Digital audio I/F	0	BCLKINV	MS	LRSWAP	LRP	WL[1:0]		Format[1:0]		000001010
R8	0x08	Sampling rate	0	CLKODIV2	CLKDIV2	SR[3:0]				BOSR	USB	000000000
R9	0x09	Active	0	0	0	0	0	0	0	0	Active	000000000
R15	0x0F	Software reset	Reset[8:0]									000000000
R16	0x10	ALC Control 1	ALCSEL[1:0]		MAXGAIN[2:0]			ALCL[3:0]				001111011
R17	0x11	ALC Control 2	0	DCY[3:0]				ATK[3:0]				000110010
R18	0x12	Noise gate	0	NGTH[4:0]					NGG[1:0]		NGAT	000000000

4. Implementation

We developed the system model with Simulink and evaluated the measured data using MATLAB. We then used ModelSim to verify the results of the hardware simulation of the adaptive filter using Fast-LMS algorithm and plotte d the obtained output using MATLAB to verify the noise attenuation. The final synthesis of the SystemVerilog code has been done on the provided Sockit Cyclone V FPGA.

a. Simulink and MATLAB

The LMS algorithm was modelled on Simulink and Matlab to verify operation. We also used matlab to iterate over the design of the algorithm to make it more efficient. We mathematically modelled the algorithm and reduced the control block diagrams to make it suitable for implementation on the FPGA. We then tested the designed algorithm with actual audio input from a mic interfaced via USB and played back the output. The Simulink block diagram designed is shown below:



b. ModelSim

The RTL or hardware design was done in SystemVerilog and first simulated on ModelSim. A state machine was designed to take care of different sequence of processings involved. The state machine has 6 states. The state machine transitions occur on the positive edge of the main clock which runs at 50 MHz, but the triggering of the state machine is on the positive edge of the DAC clock (the Idle state waits for the positive level of the DAC clock).

The first state: Waits for noise input sample

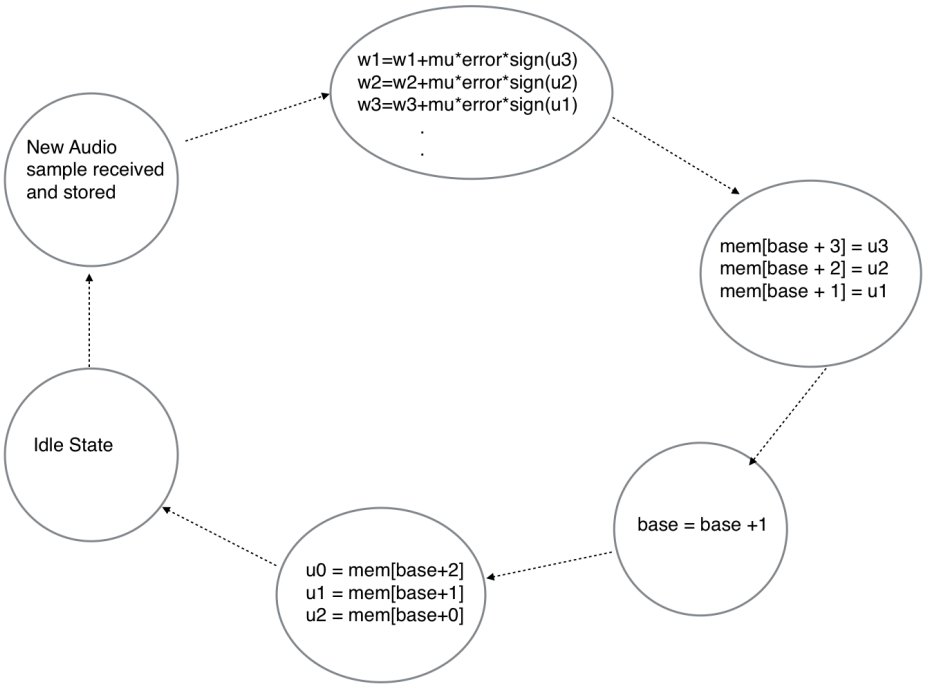
Second State: Compute filter weights

Third State: Store noise input in circular buffer

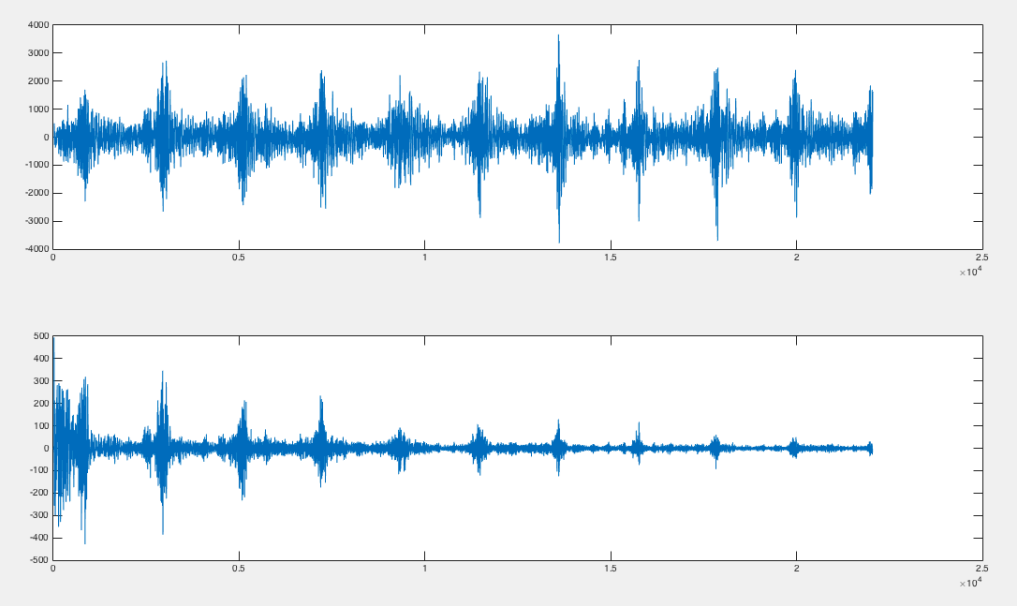
Fourth State: Increment base address pointer of circular buffer

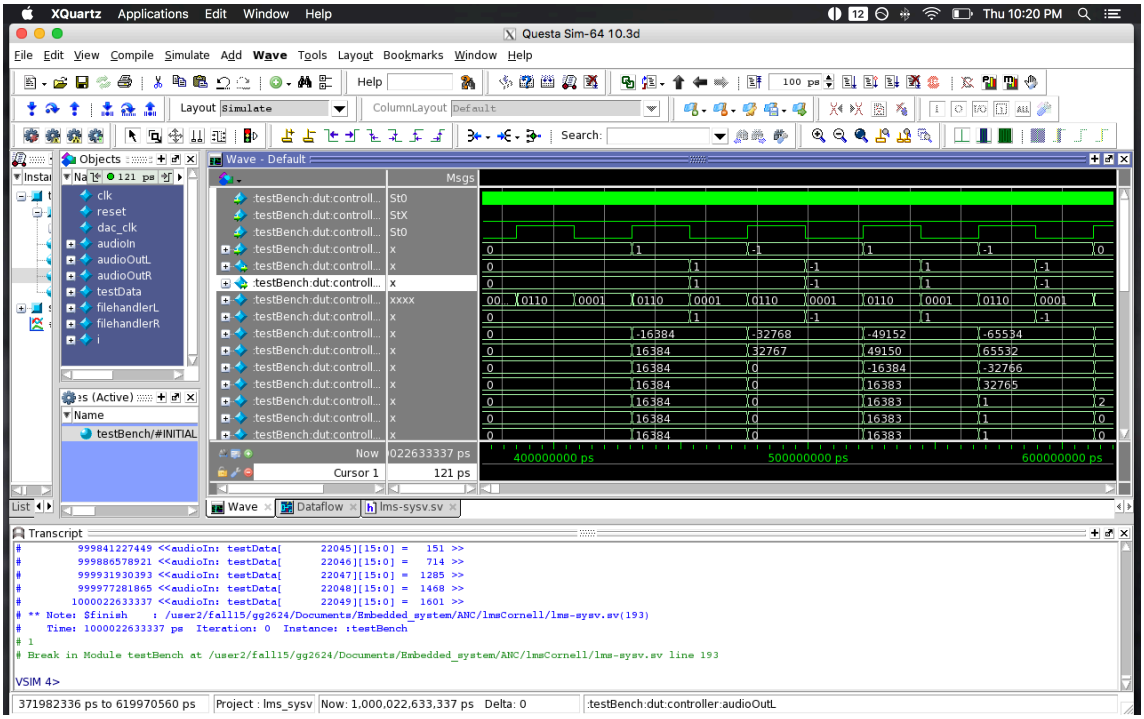
Fifth State: Restore previous history of noise sample from circular buffer for each tap

Sixth State: Idle State



The error values are calculated using continuous assignment statements which computers the values instantly when any of its input changes.
The output values for the DAC are written at the negative edge of the DAC clock so that the data is ready for pushing to the DAC in the audio codec during the next immediate clock cycle.
We recorded data from a mic via MATLAB had it convered to integer samples to feed into the ModelSIm simulation in real time.
A testbench was also designed in system Verilog which feeds input generated as mentioned above into the designed System Verilog module in real-time according to the required clock. It also records the output to a file in real-time. The output date is then taken out and plotted in Matlab to check the functioning. The plot is shown as seen below:


4000
3000
2000
1000
0
-1000
-2000
-3000
-4000
0
0.5
1
1.5
2
2.5
×10^4
500
400
300
200
100
0
-100
-200
-300
-400
-500
0
0.5
1
1.5
2
2.5
×10^4


XQuartz Applications Edit Window Help
Thu 10:20 PM
Questa Sim-64 10.3d
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Objects
clk
reset
dac_clk
audioIn
audioOutL
audioOutR
testData
filehandlerL
filehandlerR
i
Wave - Default
Msgs
Now
Cursor 1
400000000 ps
500000000 ps
600000000 ps
Wave
Dataflow
lms-sysv.sv
Transcript
VSIM 4>
371982336 ps to 619970560 ps
Project : lms_sysv
Now: 1,000,022,633,337 ps Delta: 0
:testBench:dut:controller:audioOutL

5. **Challenges and Lessons Learnt:**

CHALLENGES:

a. Figuring out Audio Codec : Extensive reading of the datasheet to figure out the different modes and configuration registers
b. Implementing I2C communication for the Audio Codec
c. LMS optimization and reduction : Spent weeks pouring over research papers and math to figure out different operations and how we could reduce them to enable faster convergence as well as reduce multiplications to make it easier to implement on FPGA.
d. Find an alternative to floating point: Ideally the weights are float values which provide better filter performance. Since the SoCKit Altera FPGA does not have a floating point unit, we had to use only integer waits and had to find a way to work with that.
e. Integrating the Audio Codec and the LMS module together for real-time performance

LESSONS LEARNT:

a. Designing real-time embedded systems requires a good knowledge of clock level execution of statements.
b. Clock is god.
c. Time management

6. **Future Work:**

The same setup can be used with a more optimized version of the fast LMS algorithm or a more optimized algorithm to provide better cancellation effect. The model takes in ambient noise as an input and produces a reduced version of the noise signal using a single microphone and a single audio output. In the current setup, silence is inherently mixed with the ambient noise signal to form the input of the system. This system can be extended to take any audio signal mixed with an ambient noise and provide a noise reduced/cancelled audio signal as the output. An enhancement to provide support for mp3 format audio files can be added to improve the flexibility and extend the operation domain of the system.

7. **Contribution:**

a. **System Design:** Richa, Gikku
b. **LMS:** Algorithm Design: Richa, Gikku
FPGA: Gikku, Rishi
c. **Audio Codec:** Rishi, Ashwin

d. **Presentation:** Rishi, Ashwin

d. **Report:** Richa

REFERENCES:

[1] http://www.ti.com/lit/an/spra095/spra095.pdf

[2] Foul, Wolfgang, Jörn Matthies, and Bernd Schwarz. "A FPGA-based adaptive noise cancelling system." Proc. of the 12th Int. Conference on Digital Audio Effects (DAFx-09). 2009.

[3] Software/Hardware Implementation of an Adaptive Noise Cancellation System" Dr. Wagdy H Mahmoud, Dr. Nian Zhang, University of the District of Columbia

[4] https://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2011/jy554_jc2636/jy554_jc2636/_fpgaimplementation.htm

[5] http://blog.jabra.com/anc-headsets-arent-all-the-same-three-types-of-anc/

[6] http://www.analog.com/media/en/technical-documentation/data-sheets/SSM2603.pdf

[7] SoCKit User Manual