

Active Noise Cancellation System

Final Report

BY

Jessica Arbona & Christopher Brady

Department of Electrical and Computer Engineering

Bradley University

Advisors: Dr. Yufeng Lu and Dr. In Soo Ahn

Peoria, Illinois
May 2012

ACKNOWLEDGEMENT

We would like to express our sincere gratitude and appreciation to our advisors who helped us all along the way with our project. Their help was invaluable as we progressed through the project. We would also like to acknowledge all the professors who helped us. Dr. Jose Sanchez provided us with additional resources and support. Dr. James Irwin provided us with helpful insight into the intricacies of acoustics needed to perform real-world active noise cancellation. Finally we would like to thank our parents and families for supporting us in our education careers.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENT	ii
LIST OF TABLES	v
LIST OF FIGURES	vi
ABSTRACT	viii
CHAPTER	
1. INTRODUCTION	1
1.1 OBJECTIVE	1
1.2 SIGNIFICANCE.....	1
1.3 ADAPTIVE FILTER.....	1
1.4 FOUR FUNDAMENTALS CLASSES	2
2. PROJECT APPROACH	5
2.1 HIGH LEVEL BLOCK DIAGRAM	5
2.2 FUNCTIONAL REQUIREMENT LIST	6
2.3 PERFORMANCE SPECIFICATIONS	7
2.4 TECHINICAL METHODS	7
3. PROJECT SIMULATION.....	9
3.1 LMS & RLS SPECIFICATIONS	10
3.2 LMS RESULTS	12
3.3 RLS RESULTS.....	15
3.3 THE DIFFERENCE BETWEEN RLS AND LMS	19
4. PROJECT HARDWARE DESIGN	19
4.1 ACTIVE NOISE CANCELLATION SYSTEM DESIGN.....	20
4.2 DIFFERENT STRUCTURES OF FIR.....	25
4.3 DSP/FPGA IMPLEMENTATION	31

5. SUMMARY.....	36
6. CONCLUSION.....	37
APPENDIX	
A. LMS ALGORITHM CODE	38
B. RLS ALGORITHM CODE	39
C. FILTER USING THE LMS ALGORITHM CODE.....	40
D. FILTER USING THE RLS ALGORITHM CODE.....	41
E. SIMULATE INTERFERENCE FOR REFERENCE SIGNAL CODE...	42
F. DISPLAY RESULTS CODE	43
BIBLIOGRAPHY	45

LIST OF TABLES

Table	Page
Table 1 Number of taps vs. LMS Mean Square Error Evaluation.....	12
Table 2 Number of taps vs. RLS Mean Square Error Evaluation.....	16
Table 3 Difference between RLS and LMS.....	19
Table 4 System Components for the XtremeDSP Board.....	21
Table 5 System Components for the SingalWave Board.....	32

LIST OF FIGURES

Figure	Page
Figure 1 Adaptive Filter	2
Figure 2 Adaptive Identification.....	3
Figure 3 Adaptive Inverse.....	3
Figure 4 Adaptive Predictor.....	4
Figure 5 Adaptive Noise Cancellation	4
Figure 6 High Level Block Diagram of an Adaptive Filter	6
Figure 7 Target Signal	10
Figure 8 Interference Signal.....	10
Figure 9 Reference Signal.....	10
Figure 10 Target Signal Spectrum	11
Figure 11 Interference Signal Spectrum	11
Figure 12 Reference Signal Spectrum	11
Figure 13 Number of taps vs. LMS Mean Square Error.....	13
Figure 14 Converge of LMS Coefficients	14
Figure 15 LMS Desired and Recovered Signals.....	14
Figure 16 LMS Desired and Recovered Signals Spectra.....	15
Figure 17 Number of taps vs. RLS Mean Square Error.....	16
Figure 18 Converge of RLS Coefficients	17
Figure 19 RLS Desired and Recovered Signals.....	18
Figure 20 RLS Desired and Recovered Spectra	18
Figure 21 Active Noise Cancellation System for the XtremeDSP Board	22

Figure 22 LMS Adaptive for the XtremeDSP Board.....	22
Figure 23 Adaptive Coefficients for the XtremeDSP Board	23
Figure 24 FIR Filter for the XtremeDSP Board.....	23
Figure 25 Oscilloscope Results.....	24
Figure 26 Comparison of the Output Signal Plots	24
Figure 27 Standard Form	26
Figure 28 Standard Form Results	26
Figure 29 Transpose Form	27
Figure 30 Transpose Form Results	28
Figure 31 Systolic Form.....	29
Figure 32 Systolic Form Results.....	29
Figure 33 Systolic Pipeline Form	30
Figure 34 Systolic Pipeline Form Results.....	31
Figure 35 Active Noise Cancellation System for the XtremeDSP Board	33
Figure 36 LMS Adaptive for the SignalWave Board	34
Figure 37 Adaptive Coefficients for the SignalWave Board	34
Figure 38 FIR Filter for the SignalWave Board	35
Figure 39 LMS Desired and Recovered Signals.....	35
Figure 40 LMS Desired and Recovered Signals Spectra	36

ABSTRACT

An active noise cancellation system has been designed and implemented. Both speech and ultrasound data were used to verify the system. MATLAB/Simulink was used to design and test a least mean square (LMS) and a recursive least square (RLS) adaptive filter for the project. Once the filters were successfully simulated and verified, the Xilinx block set was used for hardware/software co-simulation and hardware implementation. This Xilinx filter model was subject to finite precision due to fixed-point arithmetic. It required careful verification via numerous simulations. Results obtained with the finite precision Xilinx model were compared with those from the MATLAB model to fine-tune the filter. Four types of FIR structures were investigated. After testing and validation using hardware/software co-simulation, the system was downloaded to a DSP/FPGA board for real-time processing of various signals.

CHAPTER 1

INTRODUCTION

1.1 OBJECTIVE

The goal of the project was to design and implement an active noise cancellation system using an adaptive finite impulse response (FIR) filter. This active noise cancellation system would be used to increase the signal-to-noise ratio (SNR) of a signal by decreasing the power of the noise. Two applications studied in this project were ultrasonic data and an audio signal with simulated interference.

1.2 SIGNIFICANCE

The study of active noise cancellation is a rapidly developing area. With the concern for noise pollution on the rise, methods of reducing noise are in greater demand. Active noise cancellation systems with adaptive filters are considered an effective method for reducing unwanted information (i.e., noise).

1.3 ADAPTIVE FILTERS

Adaptive filters consist of the three basic components: the adaptive filter, $h(n)$; the error, $e(n)$; and the adaptation function: $y(n) = x(n) * h(n)$ and $e(n) = d(n) - y(n)$ as shown in Figure 1. The goal of the system in Figure 1 is to adapt the filter in such a way that the input digital signal, $x(n)$, is filtered to produce an output signal, $y(n)$, that will minimize the error signal, $e(n)$, when subtracted from the desired signal, $d(n)$. The arrow through the adaptive filter is standard notation to indicate that the filter is adaptive. This means that all of the filter coefficients can be adjusted in such a way that the mean square error is to be minimized. The

adaptive filter can be an FIR or IIR filter or even a non-linear system. To ensure the stability of the adaptive algorithm, most adaptive filters use an FIR type.^[1]

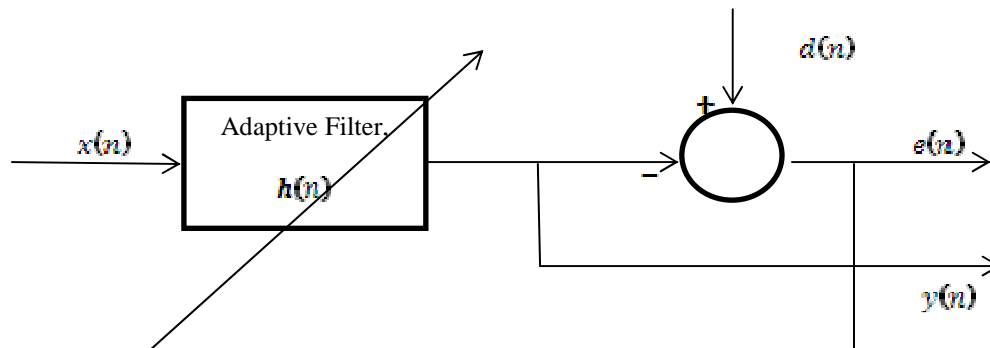


Figure 1. Adaptive Filter

The adaptive filters are widely used in areas such as control systems, communications, signal processing, acoustics, and others to deal with random signals with stationary or quasistationary statistics. Although these applications are quite different, they have input, output, error, and reference signals. The applications of the adaptive filters can be classified into four fundamental classes based on the architecture of the implementation: adaptive identification, adaptive inverse, adaptive prediction, and active noise cancellation.^[1]

1.4 FOUR FUNDAMENTAL CLASSES

Adaptive Identification

The adaptive identification is an approach to model an unknown system. As seen in Figure 2, the unknown system is in parallel with an adaptive filter, and both are receiving the input signal. The output of the unknown system provides the reference signal for the adaptive digital filter. Applications for adaptive identification include room acoustic identification, channel estimation, echo cancellation and so on.^[2]

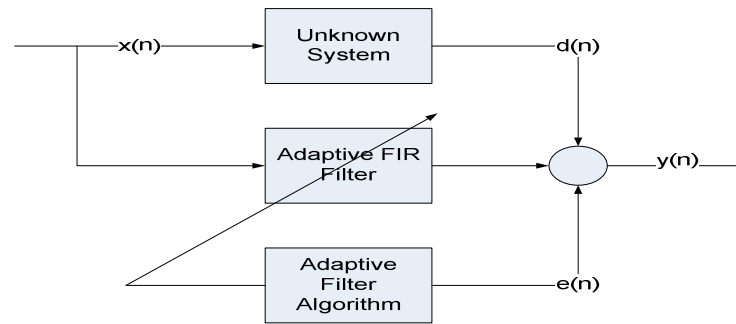


Figure 2. Adaptive Identification

Adaptive Inverse

In the architecture of adaptive inverse as shown in Figure 3, the adaptive digital filter is used to provide the inverse model for an unknown system. The inverse model realizes the reciprocal of the unknown system's transfer function. The combination of the two would then constitute an ideal transmission medium. Applications that use adaptive inverse include equalization in digital communications, predictive deconvolution, blind equalization, adaptive control systems, and others. ^[2]

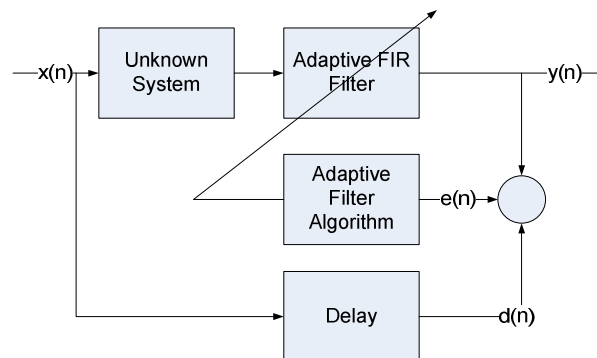


Figure 3. Adaptive Inverse

Adaptive Predictor

In the prediction architecture as shown in Figure 4, the adaptive filter is used to provide a prediction of the value of a random input signal. Depending on the application, the system can operate as a predictor if the output of the adaptive filter predicts the output of the system in advance. However, the system can also operate as a prediction error filter if the prediction error

signal is used as the output of the system. Applications of adaptive predictors include predictive noise suppression, periodic signal extraction, linear predictive coding, and others.^[2]

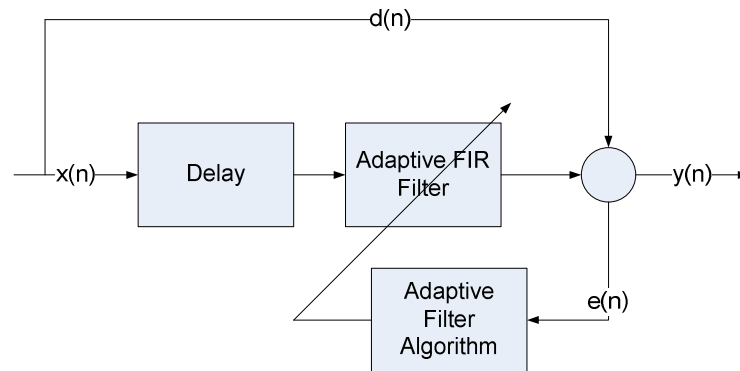


Figure 4. Adaptive Predictor

Active Noise Cancellation

Active noise cancellation increases the signal-to-noise ratio of a signal by decreasing the noise power in the signal by attempting to cancel noise signals. Applications consist of adaptive noise cancellation, echo cancellation, adaptive beamforming, biomedical signal processing, and others.^[2]

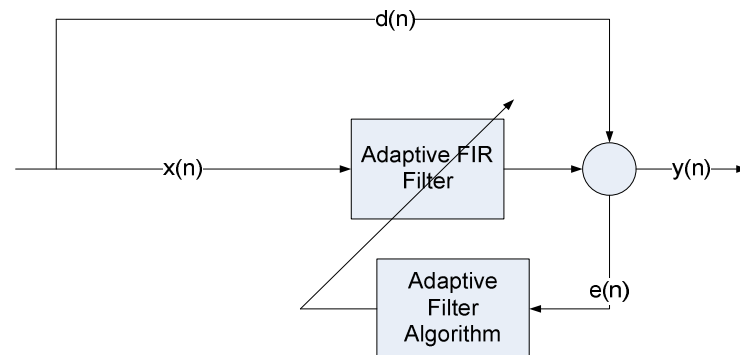


Figure 5. Adaptive Noise Cancellation

CHAPTER 2

PROJECT APPROACH

In order to complete the project, a series of design tasks was undertaken. First, a high level block diagram was made to represent the functionality of the system. After this, a functional requirements list was made to describe how the system would function. Performance specifications were then made to describe the ultimate goal of the system. The basics of two adaptive filters, recursive least square (RLS) and least mean square (LMS), were then researched to provide a method for designing the active noise cancellation system.

2.1 HIGH LEVEL BLOCK DIAGRAM

Figure 6 shows the configuration of the high level block diagram for the system. There are two inputs in the system: reference and interference signals. The reference signal, $d(n)$, contains the target signal and an interference signal. The interference signal, $x(n)$, contains just an interference signal similar to that contained in the reference signal. When the interference signal is passed through the adaptive filter, the output, $y(n)$, is generated so that when it is subtracted from the reference signal the error signal, $e(n)$, is obtained. The error signal is then used to update the coefficients of the filter.

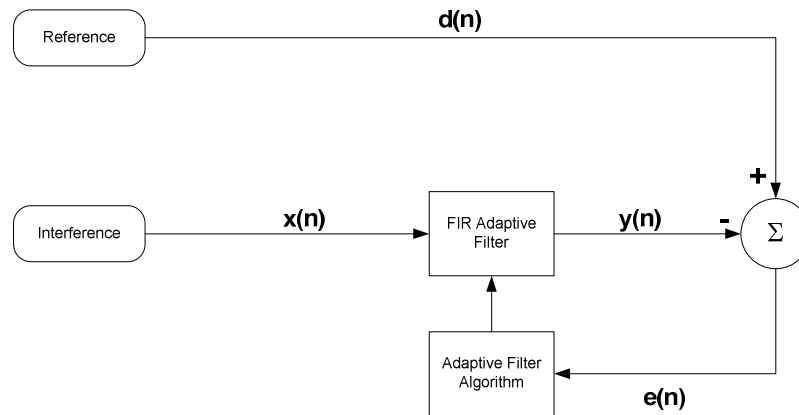


Figure 6. High Level Block Diagram of an Adaptive Filter

2.2 FUNCTIONAL REQUIREMENTS LIST

The project used two different types of data: ultrasound and speech. To process these data, two types of hardware boards were used in the project to process the different types of data. An XtremeDSP board was selected to analyze the results of the ultrasound data. The main reason to use this board was to output the results to an oscilloscope for visual inspection. A SignalWave DSP/FPGA board was used to analyze the audio data tapping to its audio Codec hardware, which allowed the signals to be heard.

The ultrasound data was acquired with a 5 MHz transducer and 100 MSPS sampling rate in an ultrasonic nondestructive data acquisition system. The adaptive filter was designed using a Xilinx system generator, an FPGA design tool incorporated in the MATLAB/Simulink environment. An XtremeDSP development kit from Nallatch was used as a platform to implement the adaptive filter. The FPGA device used in the project was the Virtex 4 XC4SX35-10FF668. Two 14-bit DAC onboard channels (AD9772 DAC devices) were used to probe the input and output of the adaptive filtering system.

For audio signal processing, a SignalWave DSP/FPGA board from Lyrtech was used to test the adaptive filtering system. An onboard audio CODEC (sampling rate varies from 8 kSPS

to 48 kSPS) was used for processing signals. Real-time workshop and the Xilinx system generator in MATLAB/Simulink were used to compile the design.

2.3 PERFORMANCE SPECIFICATIONS

The system is designed to accommodate a sampling rate conversion of at least 44.1kSPS for audio signals and be able to increase the SNR by at least 20 decibels (dB).

2.4 TECHNICAL METHODS

Mathematical Approach

Adaptive filters operate by attempting to reduce a cost function. One of the most popular cost functions to use is known as the Least Square Error equation. It uses the mean square error as the cost function and attempts to reduce the cost function. Various adaptive algorithms can be obtained based on how to minimize the cost function. The cost function (J) can be represented as follows:

$$J = E\{e^2(n)\} \quad (1)$$

The error signal of the system can be expressed as:

$$e(n) = d(n) - y(n) = d(n) - f^T X(n) \quad , \quad (2)$$

where f is the filter coefficients and $X(n)$ which is a column vector of the filter input signal

The cost function becomes:

$$J = E\{e^2(n)\} = E\{(d(n) - f^T \cdot X(n))^2\} \quad (3)$$

$$J = E\{d(n)^2 + 2d(n) \cdot f^T \cdot X(n) + f^T \cdot X(n) \cdot X^T(n) \cdot f\} \quad (4)$$

By setting the gradient of J equal to zero and solving, for the filter coefficient f , we find that:

$$E\{d(n) \cdot X(n)\} = E\{X^T(n) \cdot X(n)\} f_{opt} \quad (5)$$

Solving for the optimum coefficients results in the following equation:

$$f_{opt} = R_{xx}^{-1} \cdot r_{dx} \quad (6)$$

Least Mean Square

The Least Mean Square (LMS) algorithm, introduced by Widrow and Hoff, is an adaptive algorithm. LMS algorithm uses the estimates of the gradient vector from the available data. The LMS incorporates an iterative procedure that makes corrections to the weight vector in the direction of the negative of the gradient vector which eventually leads to the minimum mean square error. Compared to other algorithms, the LMS algorithm is considered simpler because it does not require correlation function calculations nor does it require matrix inversions.

Mathematical Approach

The Widrow-Hoff LMS Algorithm attempts to approximate the Wiener-Hopf equation by updating the filter coefficients by a factor of the negative of the gradient of the cost function as follows:

$$f(n+1) = f(n) - \frac{\mu}{2} \nabla(n) \quad (7)$$

The gradient is then calculated using the partial derivative of the cost function with respect to the filter coefficients. It can be shown that the gradient is represented by the following:

$$\nabla(n) = -2e(n) \cdot X(n) \quad (8)$$

When the gradient (8) is plugged into the Wiener-Hopf equation (7), the result is the following equation for updating the filter coefficient:

$$f(n+1) = f(n) + \mu \cdot e(n)X(n) \quad , \quad (9)$$

where μ is the step size or learning factor for the filter. In order for the filter coefficients to converge to an optimum value, a value for μ must be carefully chosen. For this LMS algorithm, it can be shown that μ must satisfy the following constraint in order for the system to converge:

$$0 \leq \mu \leq \frac{2}{3 \cdot L \cdot r_{xx}(0)}, \quad (10)$$

Where r_{xx} is autocorrelation and L is the number of taps of the filter

Recursive Least Square

Recursive least square (RLS) is another algorithm for adaptive filters. This algorithm attempts to directly update the auto and cross-correlation matrices in order to approach the Wiener-Hopf equation.

Mathematical Approach

The RLS algorithm attempts to directly update its estimate of the optimum coefficients to approach the Wiener-Hopf equation.

$$R_{xx}(n+1) = R_{xx}(n) + X(n) \cdot X^T(n) \quad (11)$$

$$r_{dx}(n+1) = r_{dx}(n) + d(n) \cdot X(n) \quad (12)$$

Using these to update our values for each new input, we calculate the filter coefficients with the following:

$$f(n+1) = R_{xx}^{-1}(n+1) \cdot r_{dx}(n+1) \quad (13)$$

CHAPTER 3

PROJECT SIMLUATION

MATLAB simulations of both LMS and RLS were used to investigate the effectiveness of the adaptive filters for recovery a signal corrupted with noise. The theoretical results were later compared to the hardware results in order to ensure effectiveness. Simulation results were also used to investigate the differences between LMS and RLS, to determine which would be better suited to be implemented in hardware.

3.1 LMS & RLS SPECIFICATIONS

Two audio signals were used in the simulations. A speech sample artificially corrupted with car engine noise was used as the reference signal for the adaptive filter, and a similar version of the engine noise was used as the interference signal.

Input Signals

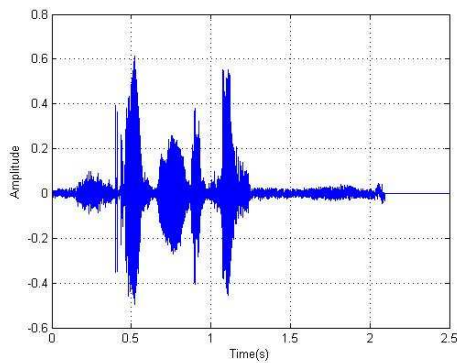


Figure 7. Target Signal

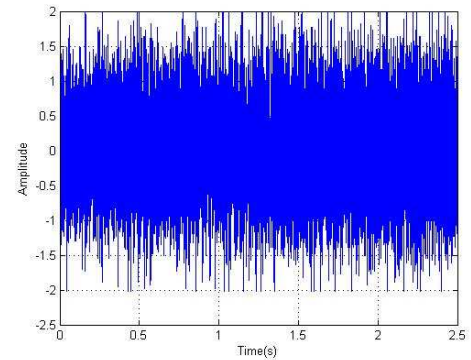


Figure 8. Interference Signal

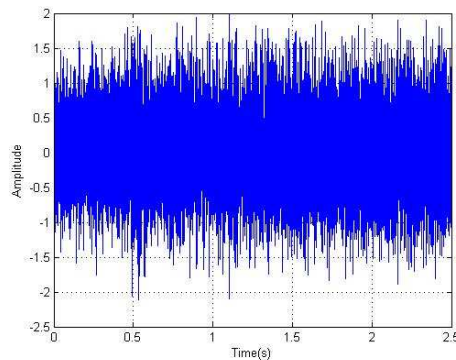


Figure 9. Reference Signal

Figure 7 shows the target signal, a speech sample of a woman saying “Give me the pen”. Figure 8 shows the engine noise. Finally, Figure 9 is the reference signal, which is the speech signal corrupted with the interference signal. A moving average process was used on the engine noise signal before being added to the speech signal to simulate an environment where the

interference signal was reflected several times. The process ensures that the filter input was not exactly the same noise that was corrupting the speech data. It can be seen that the signal in Figure 10 is smaller compared to signal in figure 11. This is expected due to the signal in Figure 11 being the interference signal.

Input Signals Spectral

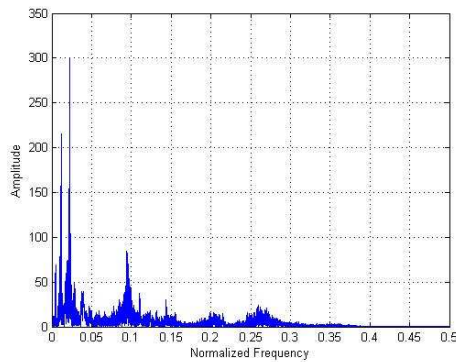


Figure 10. Target Signal Spectrum

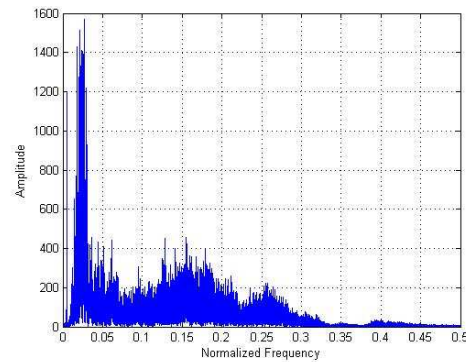


Figure 11. Interference Signal Spectrum

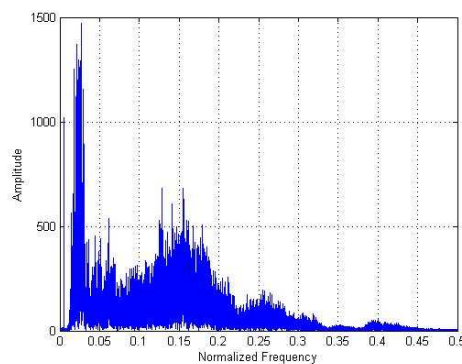


Figure 12. Reference Signal Spectrum

Figure 10 is the spectral content of the speech signal. Figure 11 is the spectral content of the engine noise. Finally, Figure 12 is the spectral content of the reference signal, which is the target signal with the interference signal.

3.2 LMS RESULTS

A comparison of different LMS filters was conducted to determine an appropriate filter to be used. The LMS adaptive filter was implemented first due to its simplicity. As explained before, LMS does not require correlation function calculation nor does it require matrix inversions. MATLAB simulations show that a reduction of 20 db can be achieved by tuning the step size of the LMS algorithm.

Least Mean Square Taps Evaluation

Figure 13 shows that as the number of taps in the filter increased, the noise reduction increased as well. This became less noticeable as the number of taps exceeded 10. As seen in Table 1, there is no significant difference when the numbers of taps varied from 10 to 20. Because of this observation, and in order to simplify process of hardware design, it was decided to implement ten taps. The rest of the results were achieved using a 10 tap adaptive filter.

Table 1. Number of taps vs. LMS Mean Square Error Evaluation

Taps (L)	Mean Square Error (J)	Reduction [dB]
4	0.011600	-9.96
6	0.003400	-15.29
8	0.001500	-18.84
10	0.001100	-20.19
12	0.000936	-20.89
14	0.000896	-21.08
16	0.000876	-21.18
18	0.000864	-21.24
20	0.000856	-21.28

Note: The original mean square error (J) is 0.114844.

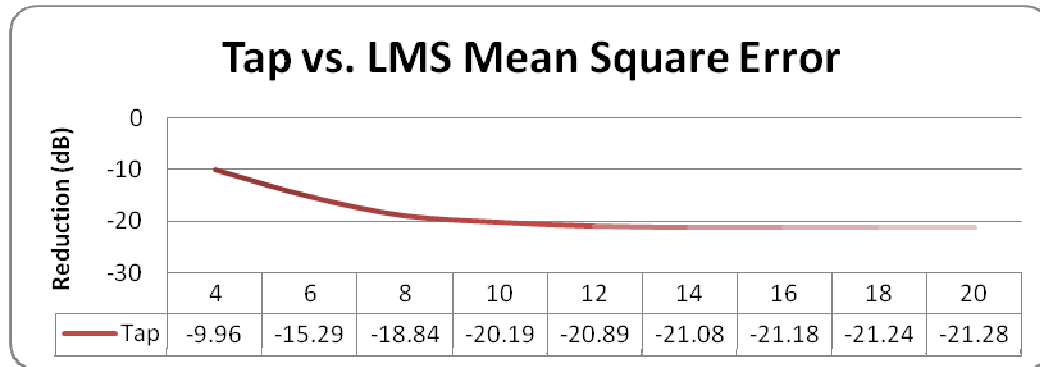


Figure 13. Number of taps vs. LMS Mean Square Error

LMS MATLAB Code

The code in Appendix A was used to perform the LMS algorithm. The first several samples from each signal were not processed to give the algorithm a starting point from which it could accurately recover the target signal. The number of samples skipped was one fewer than the number of taps used. This is done to account for the fact that there was insufficient data for these samples to be filtered. For each sample after that, the current input vector was used to calculate the filter output by using matrix multiplication. The recovered signal was then calculated and used to update the filter coefficients for the next sample. This process was continued until all the samples had been filtered.

LMS MATLAB Results

It can be seen in Figure 14 that the coefficients of the LMS required 0.5 seconds to converge. These coefficients were used to filter the signal to reduce the error. The LMS coefficients took 1.3 seconds to become stable.

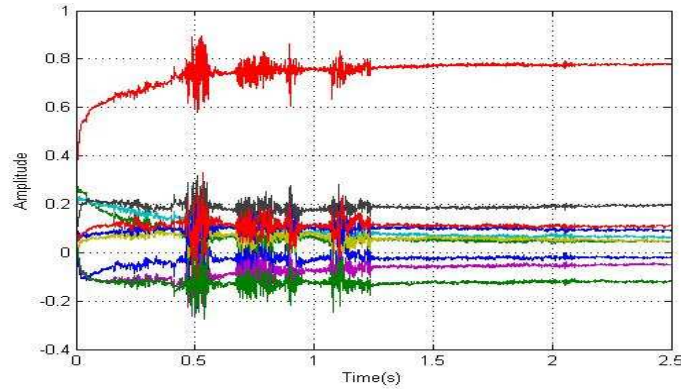


Figure 14. Convergence of LMS Coefficients

With ten taps, the LMS algorithm was able to reduce the mean square error from 0.114844 to 0.0011. This is a reduction of more than 20.0 dB, which allowed the target signal to be recovered, as seen in Figure 15 and Figure 16. The MATLAB audio results can be heard clearly. However, the results of the implementation allowed a small part of the noise of the engine motor to be heard in the background of the output signal.

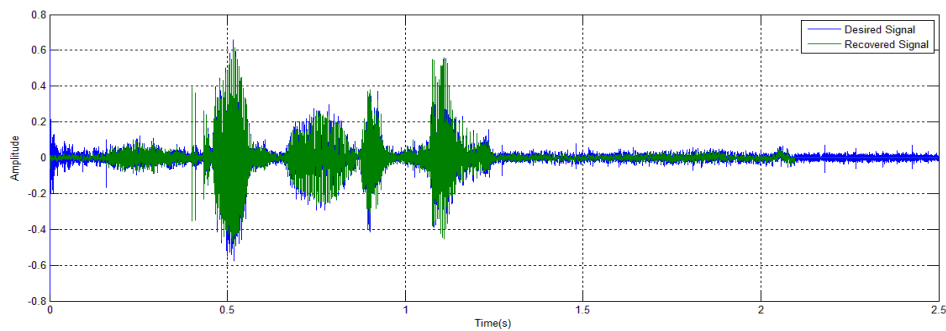


Figure 15. LMS Desired and Recovered Signals

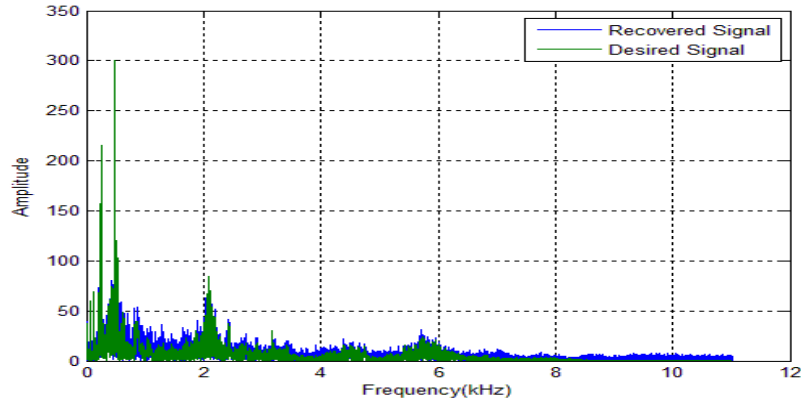


Figure 16. LMS Desired and Recovered Signal Spectra

3.3 RLS RESULTS

Recursive least square is another algorithm for the adaptive filter. This algorithm attempts to directly update the auto- and cross-correlation matrices in order to approach the Wiener-Hopf equation. A recursive least square (RLS) adaptive filter was implemented second. MATLAB simulation was conducted to compare results with LMS.

Recursive Least Square Taps Evaluation

Table 2 shows that as the number of taps in the filter increased, the dB reduction increased as well. This became less severe as the number of taps exceeded 12. As seen in Table 2, the reduction from 12 to 20 taps was minimal. It was decided from the MATLAB result to use 10 Taps, even though the difference between taps 10 and 12 was 5 dB. The main reason was to have accurate comparisons between LMS and RLS. The rest of the results were achieved by a ten tap adaptive filter.

Table 2. Number of taps vs. RLS Mean Square Error Evaluation

Taps (L)	Mean Square Error (J)	Reduction [dB]
4	0.008618	-11.25
6	0.001755	-18.16
8	0.000430	-24.26
10	0.000082	-31.44
12	0.000024	-36.81
14	0.000034	-35.27
16	0.000013	-39.58
18	0.000013	-39.31
20	0.000016	-38.68

Note: The original means Square Error (J) is 0.114844

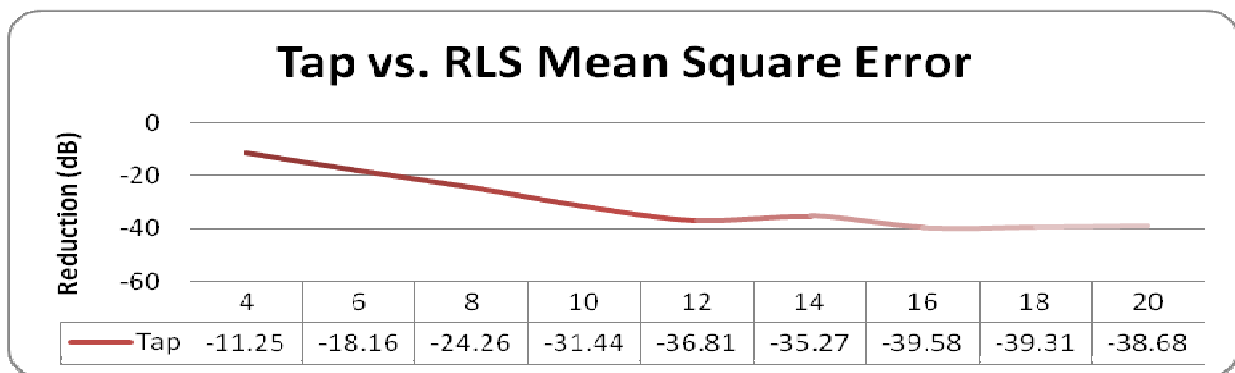


Figure 17. Number of taps vs. RLS Mean Square Error

RLS MATLAB Code

The code in Appendix B was used to perform the RLS algorithm. The first several samples from each signal were not processed to give the algorithm a starting point from which it could accurately recover the target signal. The number of samples skipped was one fewer than the number of taps used. This was done to account for the fact there was insufficient data for these samples to be filtered. For each sample, the current input vector was determined and used to calculate the filter output. It used matrix multiplication to determine the output and the recovered signal was then calculated. Two matrices were needed to perform the RLS algorithm. The autocorrelation matrix was set to a small, non-zero value initially to prevent any possible

singularity condition of the matrix. Both the autocorrelation and cross correlation matrices were updated and used to determine the filter coefficients for the next sample.

RLS MATLAB Results

The RLS coefficients settled in less than half a second and remained steady throughout the simulation. The spectral content of the recovered signal was much closer to that of the desired signal than that obtained from the LMS algorithm.

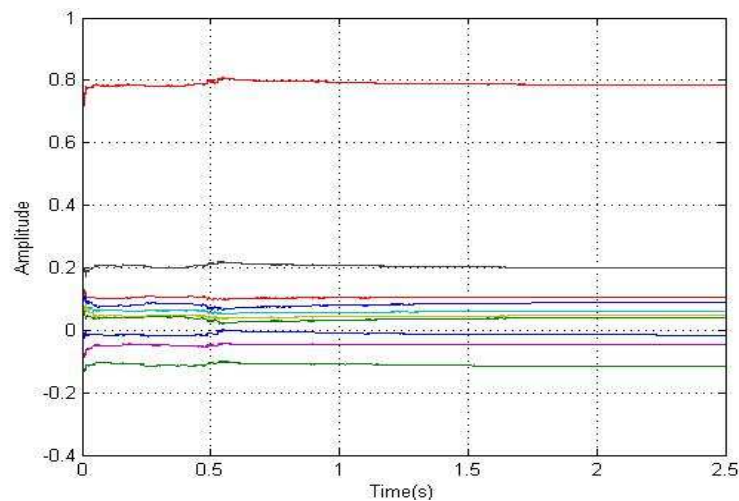


Figure 18. Convergence of RLS Coefficients

With ten taps, the RLS algorithm was able to reduce the mean square error from 0.114844 to 0.000082. This is a reduction of more than 31 dB. This reduction was enough to be able to recover the target signal, as seen in Figure 19 and Figure 20. The MATLAB audio results can be heard clearly without any engine noise in the background.

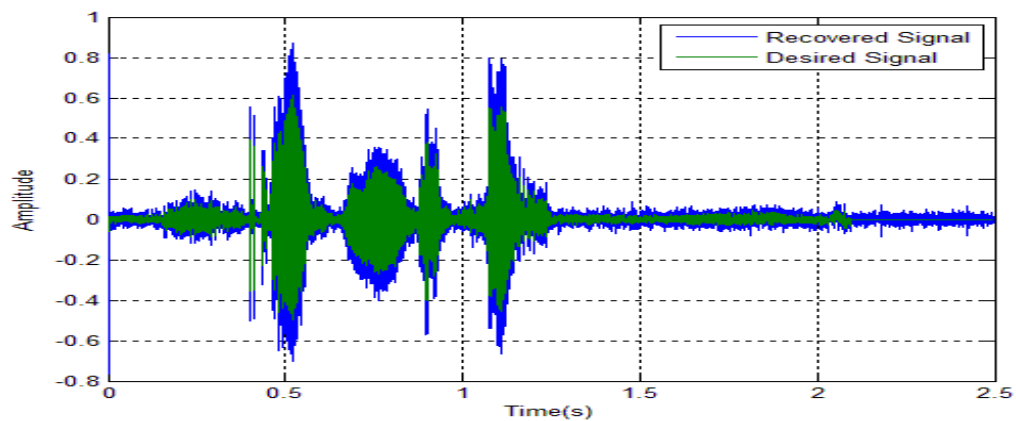


Figure 19. RLS Desired and Recovered Signals

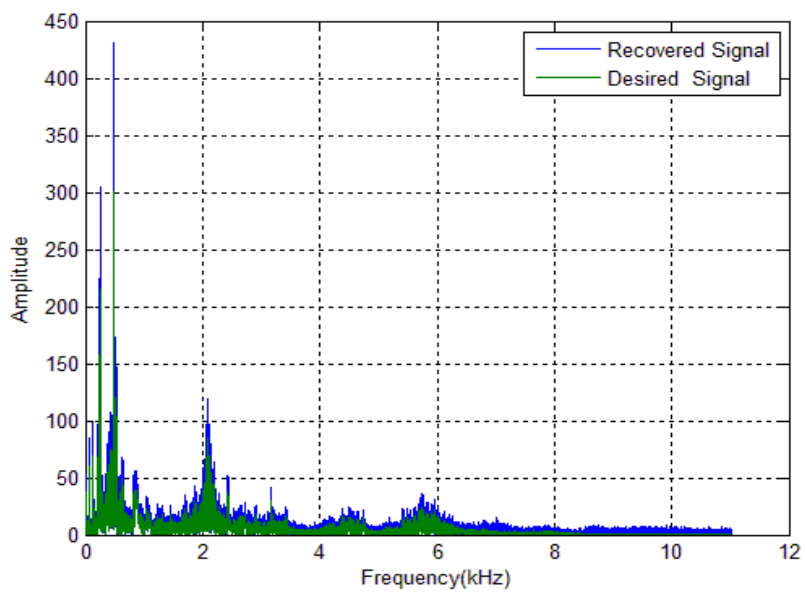


Figure 20. RLS Desired and Recovered Signals Spectra

3.4 THE DIFFERENCE BETWEEN RLS AND LMS

Table 3. Difference between RLS and LMS

Algorithm	Original Mean Square Error (J)	Mean Square Error (J)	Reduction [dB]
RLS	0.114844	0.000082	-31.44
LMS	0.114844	0.001100	-20.19

The RLS algorithm was able to achieve an additional 11 dB reduction in the mean square error over the LMS algorithm as shown in Table 3. In spite of this fact, the LMS is more widely used due to the complexity inherent in the RLS algorithm. The RLS algorithm requires a matrix inverse calculation at every time step. Although QR decomposition can be used for the matrix inversion, it was not added to the project because of its implementation complexity in a fixed point system. Inverse matrix calculations are difficult to perform on embedded system. For this reason, the LMS algorithm was chosen to be implemented in hardware.

CHAPTER 4

PROJECT HARDWARE DESIGN

After simulating the LMS and RLS design in MATLAB, the active noise cancellation system was implemented in hardware. There were three main steps taken: design and verify the active noise cancellation using an LMS adaptive filter, compare different FIR structures for efficiency in hardware implementation, and implement the design on an embedded system. The two FPGA boards were used: the SignalWave board was used for all the speech data, while the XtremeDSP Development Kit – Virtex-4 Edition, was used for the ultrasound data.

4.1 ACTIVE NOISE CANCELLATION SYSTEM DESIGN

Active Noise Cancellation System Design Specification for XtremeDSP


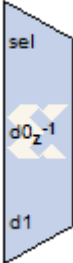
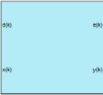
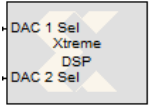
The active noise cancellation system design for the XtremeDSP had three specifications which needed to be fulfilled:

- The active noise cancellation system would resemble Figure 6. It contained an input signal, output signal, reference signal, error signal, and adaptive filter to behave as an adaptive noise cancellation system.
- The system would contain an LMS adaptive filter even though it was shown previously that RLS was more effective. The reason for choosing LMS over RLS was due to the simplicity of LMS versus the complexity of RLS.
- Six Taps would be used. This was done because the data for this part was ultrasound data instead of audio data. The ultrasound data was obtained from a MATLAB file provided by an advisor, Dr. Yufeng Lu. His results proved having a six tap FIR filter for the adaptive filter was sufficient for this data.

Active Noise Cancellation System Design for Xtreme DSP

Using the Xilinx blocks, the LMS with Adaptive Filter was designed. It consists of:

Table 4. System Components for the XtremeDSP Board

System Components	Quantity	Xilinx Blocks	Description
ROM Block	2		Each block contains the information extracted from the ultrasound MATLAB simulation
Multiplexer	2		The oscilloscope can only receive two inputs. This block is used to have control over the signals. Of which two signals out of the four signals ($x(n)$, $y(n)$, $d(n)$, and $e(n)$) will be outputted to the oscilloscope.
Adaptive Filter Block	1		It's a sub-block that contains all the adaptive filter design
Xtreme Dsp Block	1		This block converts the entire design into a bit stream for the purpose of communication with the FPGA Board.

Active Noise Cancellation with an LMS Adaptive Filter for the XtremeDSP Board

The system in Figure 21 was designed to resemble the system in Figure 6. It contained an input signal, output signal, reference signal, error signal, and adaptive filter to behave as an active noise cancellation system. The interference and reference signals were ultrasound data that had been evaluated in MATLAB and input into the system by using two ROM blocks. Ultrasound data was chosen to test the hardware design by comparing MATLAB simulation results to the output of the design in Figure 21.

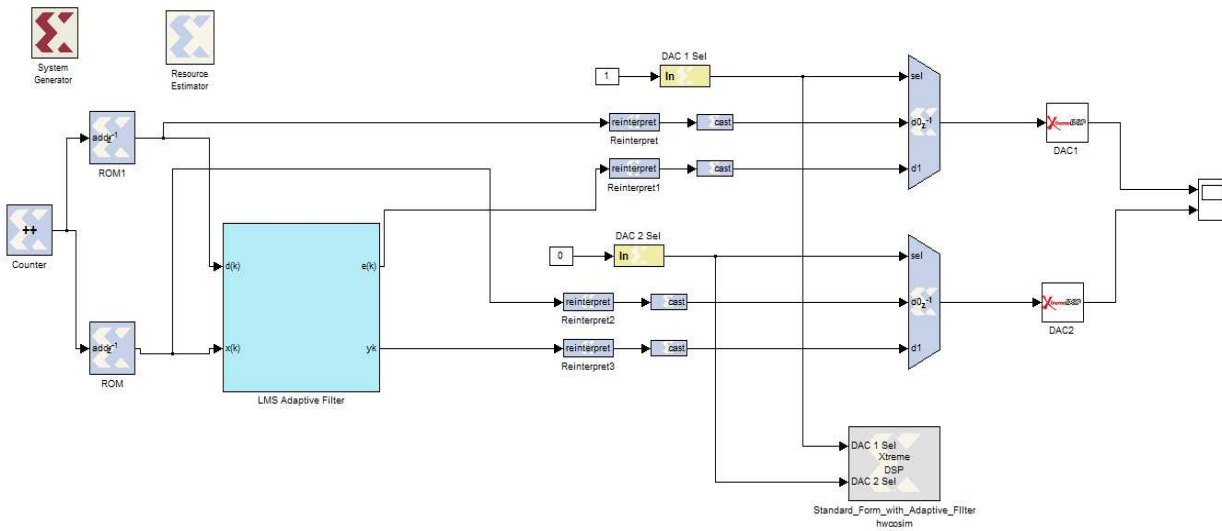


Figure 21. Active Noise Cancellation System for the XtremeDSP Board

LMS Adaptive Filter for the XtremeDSP Board

The adaptive filter shown in Figure 22 was designed to separate the adaptive coefficients and FIR filter so it would be easier to change between the different structures of FIR filters. These structures will be discussed later in the report.

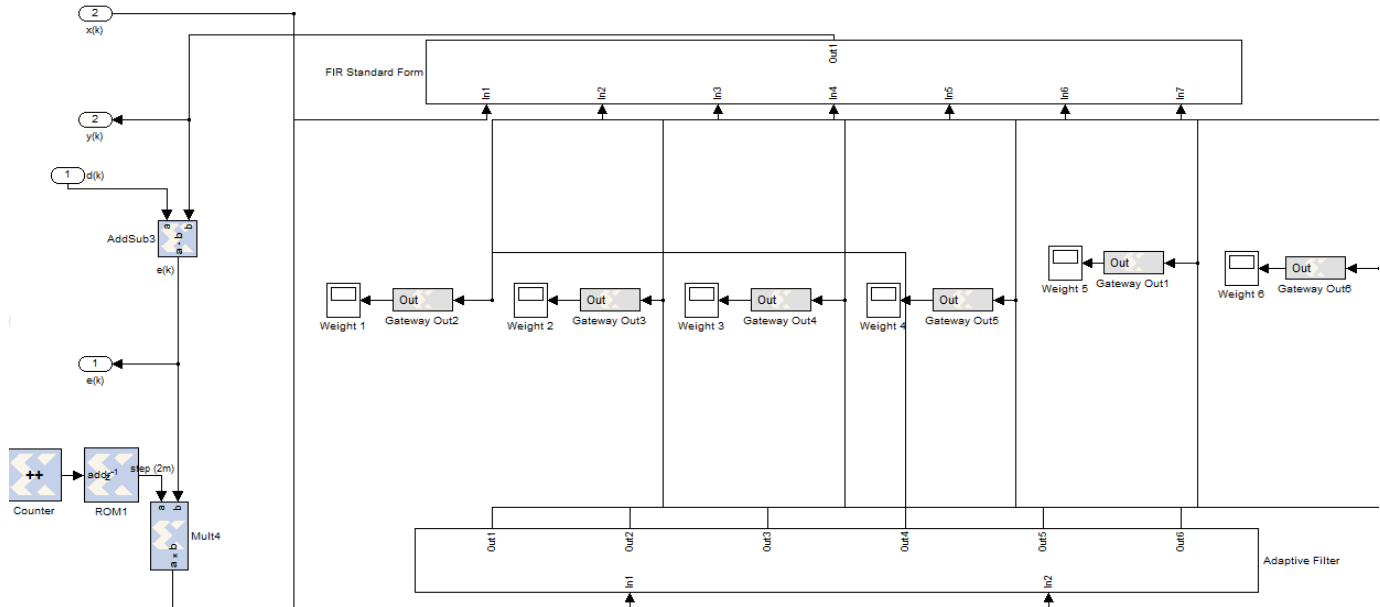


Figure 22. LMS Adaptive for the XtremeDSP Board

Adaptive Coefficients for the XtremeDSP Board

The design to calculate the adaptive coefficients for XtremeDSP board shown in Figure 23 used the standard LMS algorithm to update the coefficients with the error signal and the filter input signal.

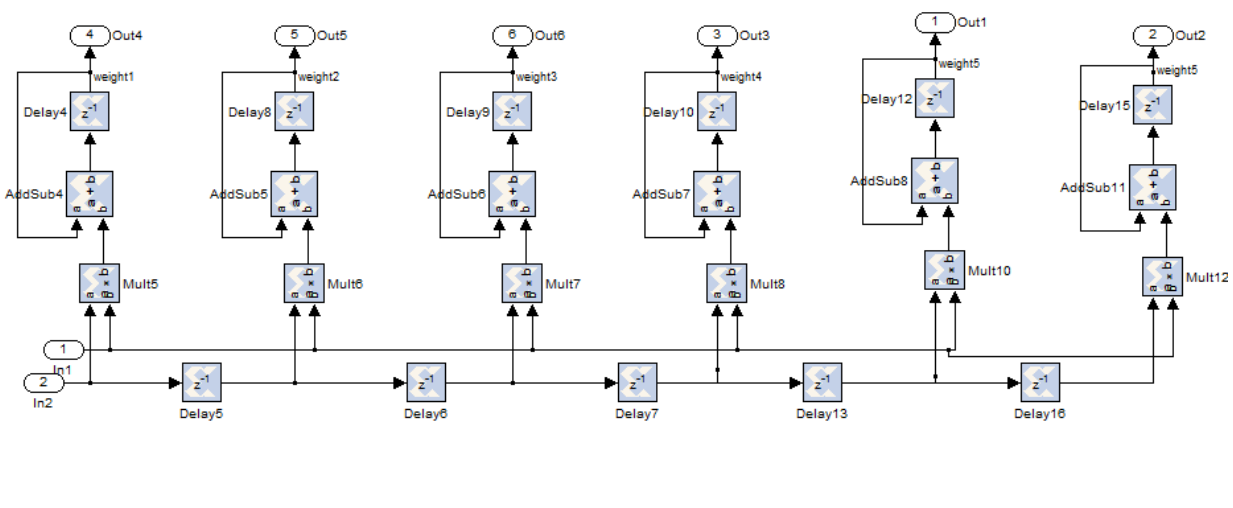


Figure 23. Adaptive Coefficients for the XtremeDSP Board

FIR Filter for the XtremeDSP Board

The design in Figure 24 is a standard form FIR filter. The input signal is delayed and multiplied by the filter coefficients. All the products are then added together to form the output.

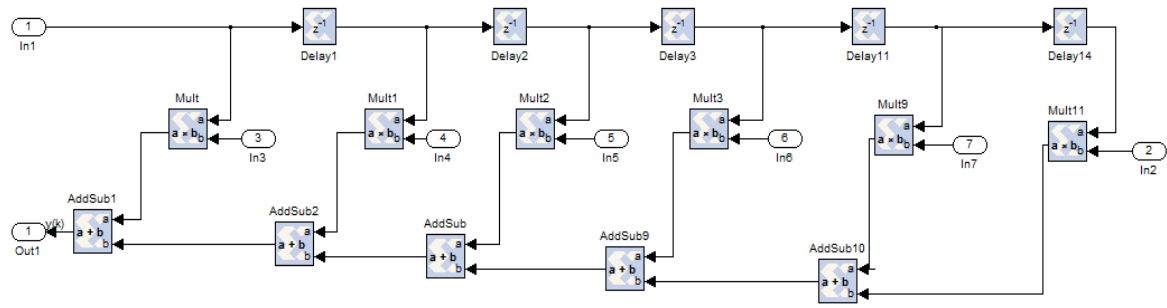


Figure 24. FIR Filter for the XtremeDSP Board

Result of the Active Noise Cancellation System Design for the XtremeDSP Board

Figure 25 displays the reference signal (orange) and the output signal (blue), which were plotted using MATLAB to verify the results.

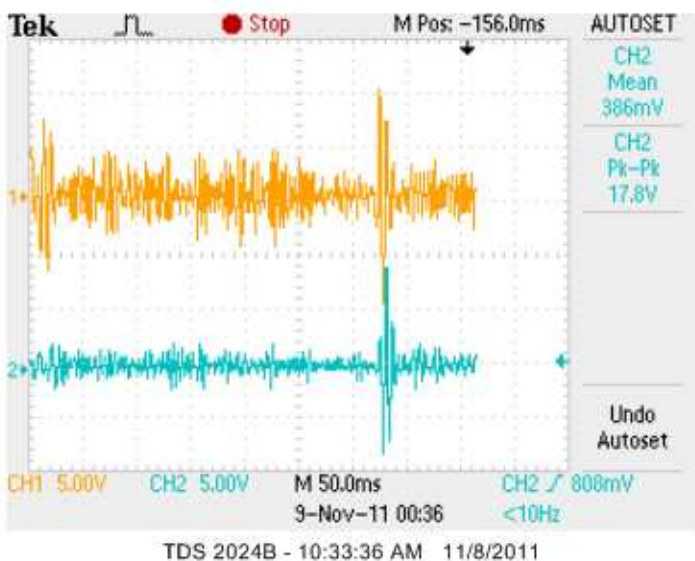


Figure 25. Oscilloscope Results

In Figure 26, Simulation Output Signal (y) (bottom graph) is the output of the hardware design for the active noise cancellation system design. The signal was compared to the top graph, Hardware Output Signal (y), to ensure that the design was working.

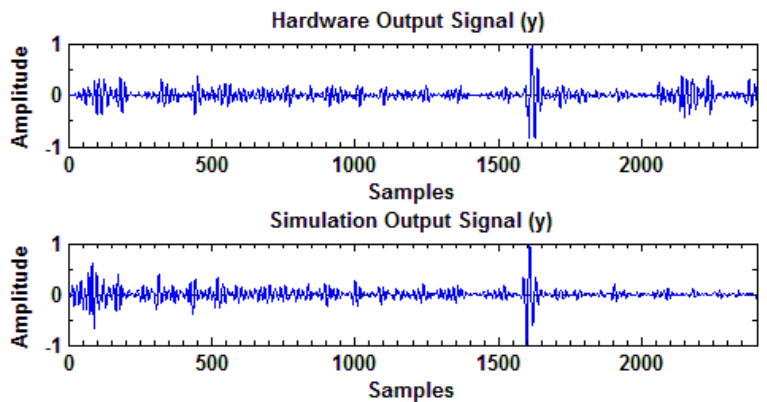


Figure 26. Comparison of the Output Signal Plots

4.2 DIFFERENT STRUCTURE OF FIR

The different representations of FIR filter structures (Standard, Transpose, Systolic, and Systolic Pipeline) were analyzed. The first step was to verify that changing the structure of the FIR portion of the adaptive filter would not change the output of the entire system. The structure would affect memory, timing, and cost in the hardware aspect. The reason to test these four different types of structure was to determine which was the most efficient with our active noise cancellation system when applied to the board. This was done by verifying that the structure worked with the LMS algorithm, observing the critical path latency, and calculating the maximum clock frequency for which the system was still operable.

Standard Form

The FIR designs of the project were mapped in a parallel architecture. As seen in Figure 27, the delay blocks were on the top while the addition blocks were parallel to them. That was what gave them a parallel form. The reason to place the blocks in this form was to have the maximum clock rate determined by the critical path latency. The critical path is considered the longest combinational delay path in a circuit. As seen in Figure 27, the arrow indicates the critical path latency. The red block shows where it starts, and the blue blocks represent the blocks that have an effect on the critical path. The latency for standard form is calculated by counting one multiplication and five additions. ^[4]

$$\text{Critical Path Latency} = \tau_{mult} + 5 * \tau_{add}$$

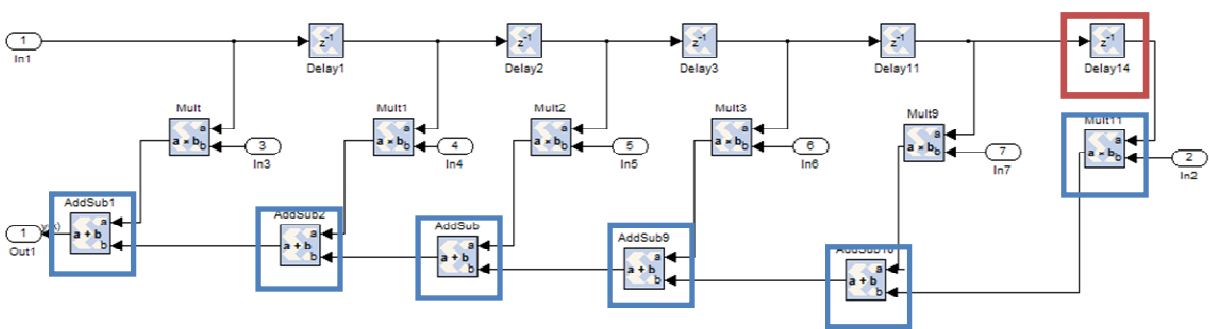


Figure 27. Standard Form

Standard Form Results

After finding the latency, the next step was to verify the validity of the FIR structure with the active noise cancellation system. To do so, the system shown in Figure 27, Standard Form, was placed into the overall system design shown in Figure 21, Active Noise Cancellation System Filter for the XtremeDSP Board. The system was run and the results were recorded in MATLAB. The second graph, Hardware Output Signal (y) in Figure 28, represents the output of the system with Figure 27, Standard Form FIR structure. It can be seen, by comparing the hardware results with those obtained with MATLAB, that the structure works.

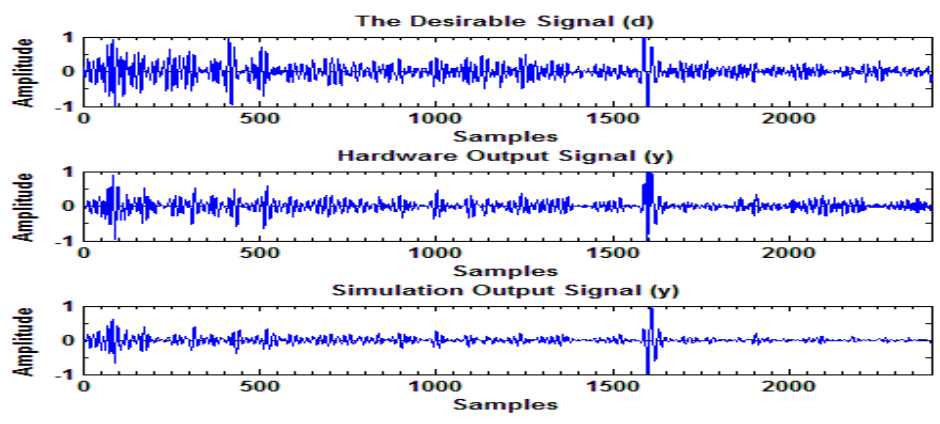


Figure 28. Standard Form Results

Transpose Form

As seen in Figure 29, the delay and addition blocks at the bottom were parallel with the multiplication blocks; this was what gave them a parallel form. The reason the blocks were placed in such a form was to have the maximum clock rate determined by the critical path latency. As seen in Figure 29 the arrow marks the critical path latency. The red block shows where it starts, and the blue blocks represent the blocks that are used to determine the critical path. The latency for transpose form is caused by one addition and one multiplication. [4]

$$\text{Critical Path Latency} = \tau_{mult} + \tau_{add}$$

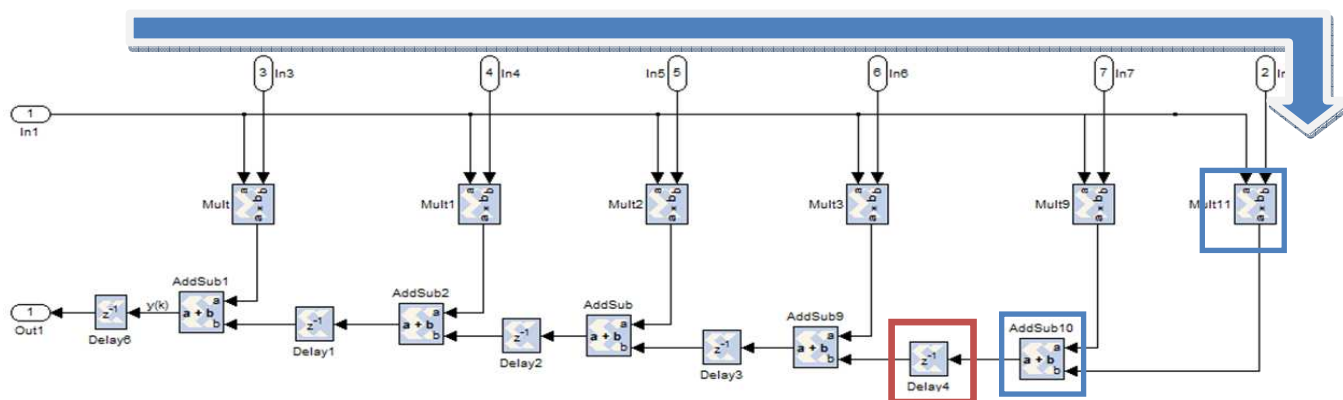


Figure 29. Transpose Form

Transpose Form Results

After finding the latency, the next step was to verify that this FIR structure performed correctly with the system. To do so, the system in Figure 29, Transpose Form, was placed into the overall system design shown in Figure 21, Active Noise Cancellation System Filter for the XtremeDSP Board. The system was run and the results were recorded in MATLAB. The second graph, Hardware Output Signal (y) in Figure 30 represented the output of system with Figure 29, Transpose Form, FIR structure. By comparing these results with those obtained with MATLAB,

it can be seen that this structure did not work. The reason it failed to perform properly was that the adaptive filter algorithm updated each filter coefficient after each new input. Transpose form used the coefficient before storing it in the delay block. This resulted in the filter coefficients being delayed as well. Therefore the adaptive filter algorithm failed to function.^[3]

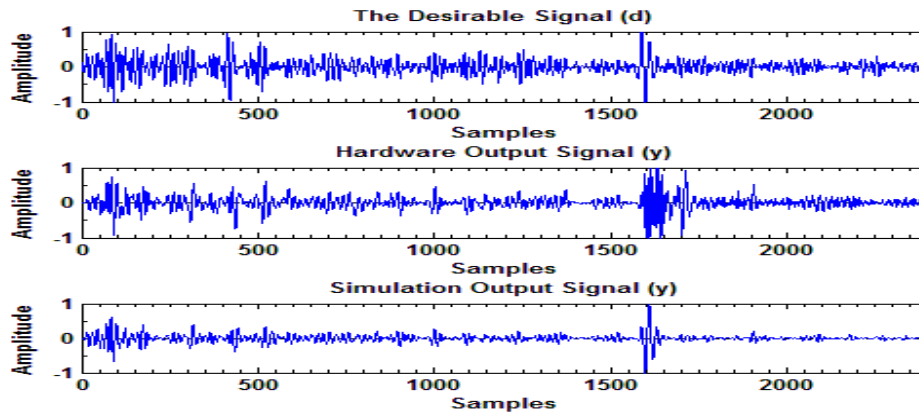


Figure 30. Transpose Form Results

Systolic Form

The FIR designs of the project were mapped in a parallel architecture. As seen in figure 31, the delays in the top were parallel with the addition blocks; this was what gave them a parallel form. The reason to place the blocks in such form was to have the maximum clock rate determined by the critical path latency. As seen in Figure 31, the arrow marks the critical path latency. The red block shows where it starts, and the blue blocks represents the blocks that determine the critical path. The latency is due to one addition and one multiplication.^[4]

$$\text{Critical Path Latency} = \tau_{mult} + \tau_{add}$$

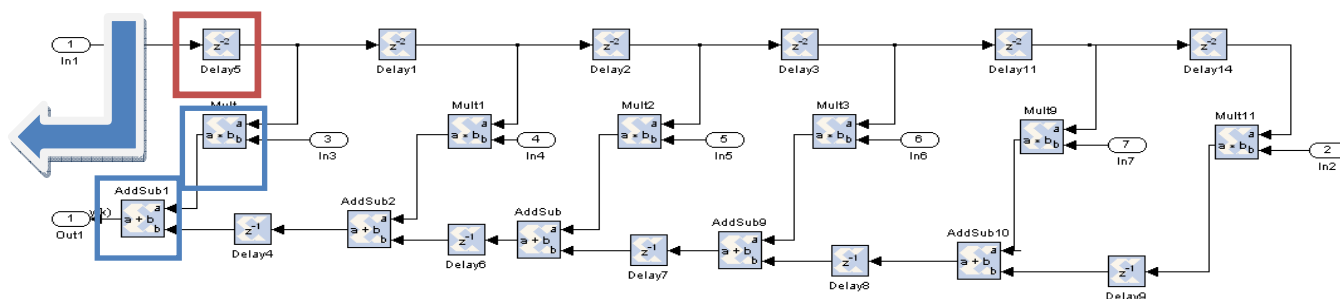


Figure 31. Systolic Form

Systolic Form Results

After finding the latency, the next step was to verify that the FIR structure actually worked with the system. To do so, the system in Figure 31, Systolic Form, was placed into the system design shown in Figure 21, Active Noise Cancellation System Filter for the XtremeDSP Board. The system was run and the results were recorded in MATLAB. The second graph, Hardware Output Signal (y), in Figure 32, represents the output of system Systolic Form, FIR structure. As seen by comparing the results obtained from hardware with the MATLAB result that this structure worked.

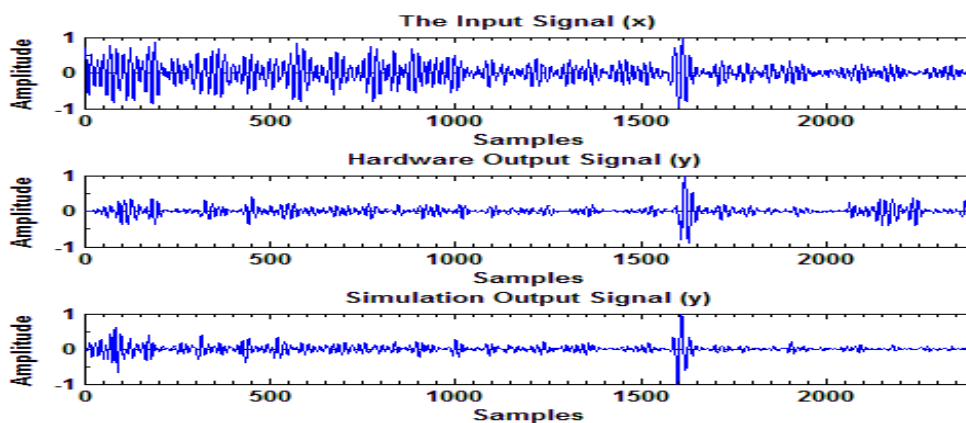


Figure 32. Systolic Form Results

Systolic Pipeline Form

The FIR designs of the project were mapped in a parallel architecture. As seen in Figure 33, the delays in the top are parallel with the addition blocks; this was what gave them parallel form. The reason to place the blocks in such form was to have the maximum clock rate determined by the critical path latency. As seen in Figure 33, the arrow marks the critical path latency. The red block shows where it starts, and the blue blocks represents the blocks that determine the critical path. The latency is seen to be the greater of the following: one addition or one multiplication. [4]

$$\text{Critical Path Latency} = \max(\tau_{mult}, \tau_{add})$$

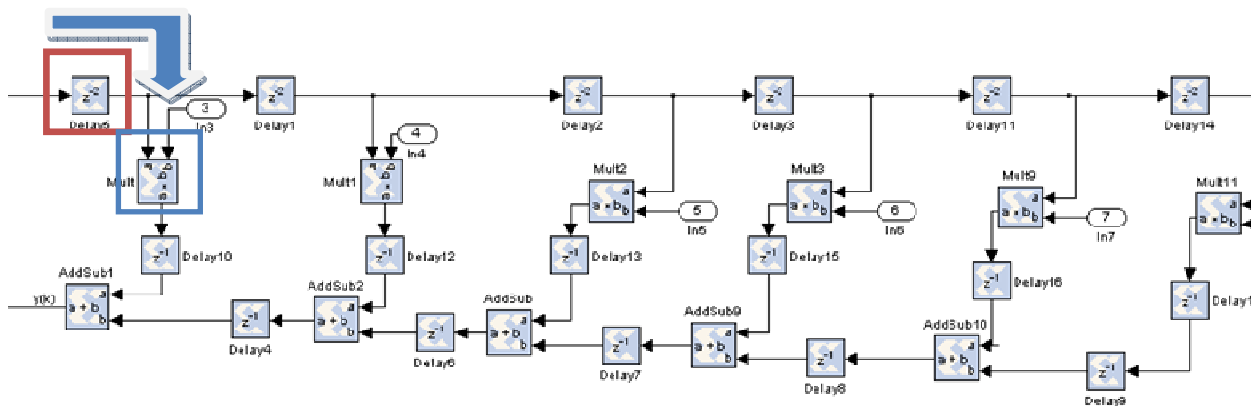


Figure 33. Systolic Pipeline Form

Systolic Pipeline Results

After finding the latency, the next step was to verify that the FIR structure worked with the system. To do so, the system in Figure 33, Systolic Pipeline Form, was placed into the system design shown in Figure 21, Active Noise Cancellation System Filter for the XtremeDSP Board. The system was run and the results were recorded in MATLAB. The second graph, Hardware Output Signal (y), in Figure 34 represents the output of system with Figure 33,

Systolic Pipeline Form, FIR structure. It can be seen by comparing the hardware results with those obtained with MATLAB, that this structure works.

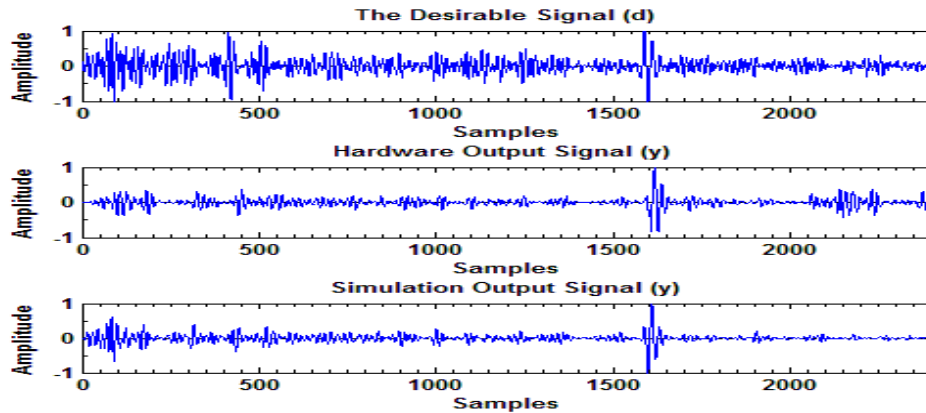


Figure 34. Systolic Pipeline Form Results

FIR Structure Results

The standard form was chosen to be implemented in the final designs. It was chosen because it was the simplest form and used the fewest hardware elements among the four forms examined. While the systolic forms had improved latency, it was determined that this was not a significant enough improvement to justify the added hardware elements.

4.3 DSP/FPGA IMPLEMENTATION

Two algorithms for adaptive filters, LMS and RLS, were successfully simulated in MATLAB. An active noise cancellation system with an LMS adaptive filter was successfully designed. An FIR structure for the system was determined. After all of these were accomplished, a hardware implementation was designed to perform real-time noise cancellation.

Active Noise Cancellation Design Specifications for the SignalWave Board


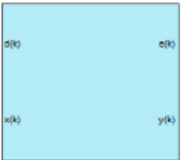
The active noise cancellation system design for the Signal Waveboard had three specifications:

- The active noise cancellation system would resemble Figure 6. It contained an input signal, output signal, reference signal, error signal, and adaptive filter to behave as an adaptive noise cancellation system.
- The system would contain an LMS adaptive filter. The reason for choosing LMS over RLS was due to the simplicity of LMS compared to RLS.
- The FIR filter would have 10 taps. It followed the results of the simulation LMS MATLAB results that using ten taps was sufficient for the requirements. The audio data used was the same data used in the MATLAB simulation.

Active Noise Cancellation Design for the SignalWave Board

Using the Xilinx block set, the LMS adaptive filter was designed. It consisted of:

Table 5. System Components for the SignalWave Board

System Components	Quantity	Xilinx Blocks	Description
ROM Block	2		Each block contains the information extracted from the ultrasound MATLAB simulation
Adaptive Filter Block	1		A sub-block that contains all the adaptive filter design

Active Noise Cancellation Design with an LMS Adaptive Filter for the SignalWave Board

The design in Figure 35 resembled that of Figure 6. It contained an input signal, output signal, reference signal, error signal, and adaptive filter to behave as an active noise cancellation system. For the input and reference signals, a microphone was used to record the two audio data that were used previously: a woman saying “give me a pen” and a car motor. These two audio samples were then mixed to simulate the target signal being corrupted by background noise. A simple moving average process was used to corrupt the target audio signal. It was chosen to simulate the noise echoing and being recorded several times at different intensities. The original noise file was then used as the filter input signal. These signals were then output through the audio jack on a PC into the SignalWave Board. The SignalWave board then used the real-time active noise cancellation to recover the target audio signal and output it to a speaker.

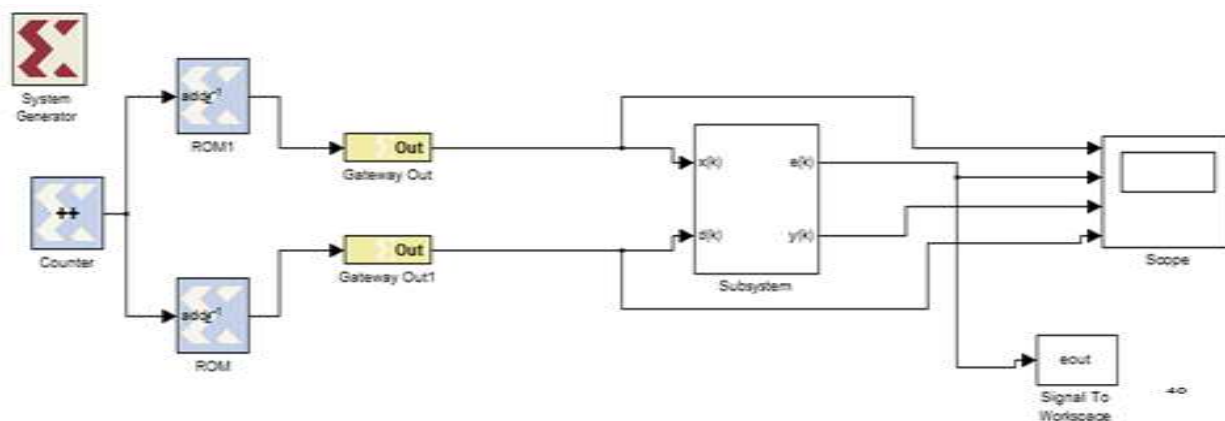


Figure 35. Active Noise Cancellation System for the XtremeDSP Board

LMS Adaptive Filter Design for the SignalWave Board

The adaptive filter in Figure 36 resembled the adaptive filter design used on the FPGA board in Figure 22. It contained the adaptive coefficient design and FIR standard form design.

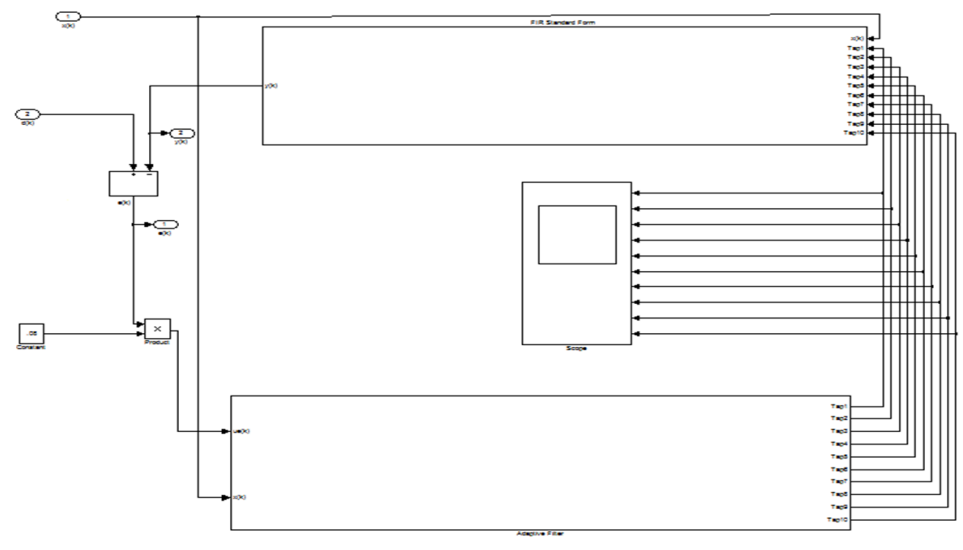


Figure 36. LMS Adaptive Filter for the Signal Wave Board

Adaptive Coefficients Design for the Signal Wave Board

The model in Figure 37 was designed similarly to that in Figure 23. The main difference was the number of taps was increased to 10. This was done because, as it was explained in the Software LMS Results, ten taps provided the desired results for an active noise cancellation system.

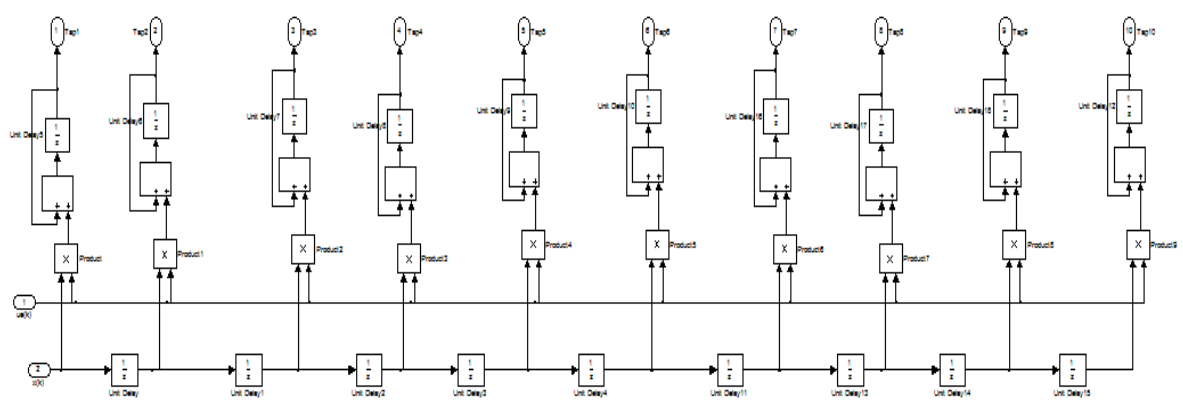


Figure 37. Adaptive Coefficients for the Signal Wave Board

FIR Filter Design for the Signal Wave Board

The design shown in Figure 38 is a standard form FIR filter. It was chosen due to the findings in the section on FIR filter forms. It closely resembles the design shown in Figure 27. The main difference is the number of taps is increased to 10.

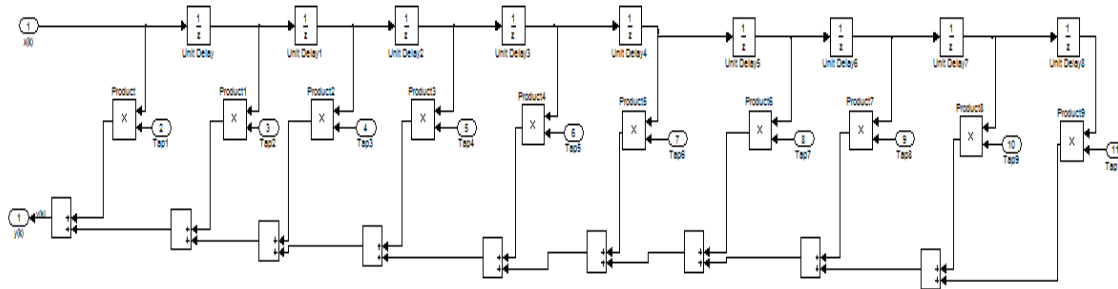


Figure 38. FIR Filter for the Signal Wave Board

Result of the Active Noise Cancellation Design for the Signal Wave Board

The results obtained for the hardware implementation, Figure 39 and 40, show slightly more engine noise remaining in the recovered signal. The engine noise was still significantly reduced. The audio signal can be clearly heard, but only a 14.6 dB reduction in noise was obtained.

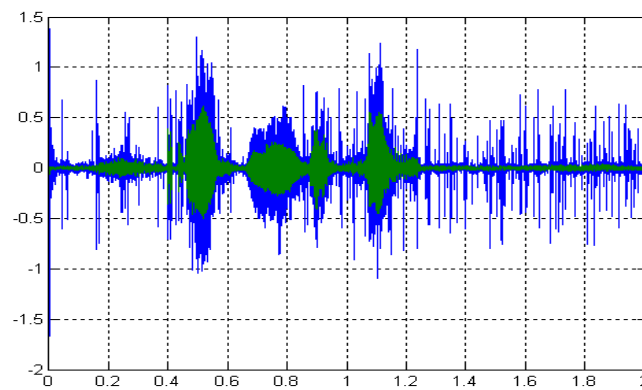


Figure 39. LMS Desired and Recovered Signals

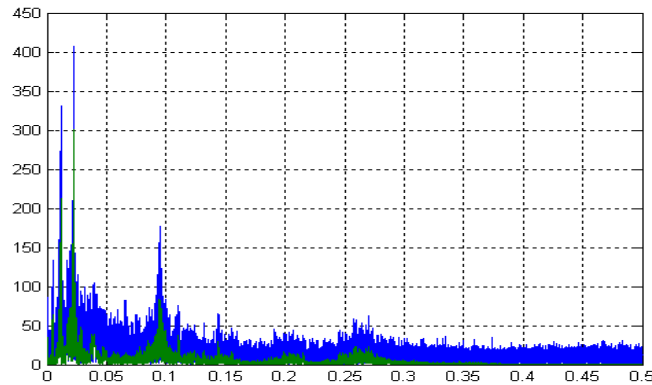


Figure 40. LMS Desired and Recovered Signals Spectra

CHAPTER 5

SUMMARY

A Least Mean Square (LMS) adaptive filter was implemented first. A MATLAB simulation was conducted to determine a step size for an acceptable performance. Simulink models were designed to generate *.bit* files to program the FPGA devices. Various pre-defined step sizes were chosen for data sets from different applications. Different structures for FIR filters were designed and compared in terms of maximum frequency (minimal delay) and usage of logic resources. Hardware implementation of on-line step size calculation was implemented as a comparison. A recursive least square (RLS) adaptive filter was then implemented.

CHAPTER 6

CONCLUSION

An active noise cancellation system was successfully simulated and implemented. The system met all requirements. The ultra-sound data was properly filtered, and the audio data was adequately filtered to recover the target signal. These requirements were accomplished in both simulation and hardware.

Some difficulty was encountered with the SignalWave board. There was some difficulty in programming of the FPGA side of the board to communicate with the DSP side. This led to the final hardware implementation to be done entirely on the DSP side of the board. In the future, students may wish to determine the cause for this problem and find a solution.

Also, the hardware could be designed into a real-time platform using microphones and speakers to detect the audio signals and perform the filtering on them. The speakers could then be used to generate a sound wave to negate the noise. This approach would require extensive study into acoustics, which was unavailable to us at the time of the project.

APPENDIX A

CODE FOR LMS ALGORITHM

```
function [recov f_out f_coef] = LMS_forgetting(ref,f_in,Tap,alpha,delta)

N = length(ref);
f_coef = zeros(Tap,N);
f_out = zeros(N,1);
recov = zeros(N,1);
P = zeros(N,1);
f_in_init = [ f_in(Tap-1:-1:1)' 0 ]';
P(Tap-1) = f_in_init' * f_in_init;
for i = Tap : N-1
    f_in_vec = f_in(i:-1:i-(Tap-1));
    f_out(i) = f_coef(:,i)*f_in_vec;
    recov(i) = ref(i) - f_out(i);
    P(i) = alpha*P(i-1) + f_in(i) * f_in(i);
    mu = delta;
    f_coef(:,i+1) = f_coef(:,i) + mu * recov(i) * f_in_vec;
end
```

APPENDIX B

CODE FOR RLS ALGORITHM

```

function [recov f_out f_coef] = RLS_brute_forgetting(ref,f_in,Tap,delta,Auto_corr_lamda, Cross_corr_lamda)
N = length(ref);
f_coef = zeros(Tap,N);
f_out = zeros(N,1);
recov = zeros(N,1);
Auto_corr = delta * eye(Tap);
Cross_corr = zeros(Tap,1);
for i = Tap : N-1
    f_in_vec = [ f_in(i:-1:i-(Tap-1)) ];
    f_out(i) = f_coef(:,i)*f_in_vec;
    recov(i) = ref(i) - f_out(i);
    Auto_corr = Auto_corr_lamda* Auto_corr + f_in_vec * f_in_vec';
    Cross_corr = Cross_corr_lamda* Cross_corr + ref(i).* f_in_vec;

    f_coef(:,i+1) = Auto_corr \ Cross_corr; %%%inv(Auto_corr) * Cross_corr;
end

```

APPENDIX C

CODE USED TO FILTER USING THE LMS ALGORITHM

```
close all
clc

Fs = 22050;
T = 1/Fs;

x = wavread('Race Car Idle.wav');
y = wavread('Jesvoiceref.wav');

[target ref f_in t P] = c2signalsetup(x, y, Fs);

Tap = 100;

delta = 12*2/(3*Tap^2*P);
alpha = 1;
[recov f_out f_coef] = LMS_forgetting(ref,f_in,Tap,alpha,delta);

orig_error = y(1:length(recov)) - ref;
error = y(1:length(recov)) - recov;

plot_adaptive_sim(target,ref,f_in,f_out, recov, error, orig_error, f_coef, t, Fs);

J = sum((error .* error))/length(error);
orig_J = sum((orig_error .* orig_error)) / length(orig_error);

soundsc(ref, Fs)
soundsc(recov, Fs)
```


APPENDIX D

CODE USED TO FILTER USING THE RLS ALGORITHM

```
clc
clear all
close all

Fs = 22050;
T = 1/Fs;

x = wavread('Race Car Idle.wav');
y = wavread('Jesvoiceref.wav');

[target ref f_in t P] = c2signalsetup(x, y, Fs);
Tap = 100;
Cross_corr_lamda = 1;
Auto_corr_lamda = 1;
delta = 0.05;
[recov f_out f_coef] = RLS_brute_forgetting(ref,f_in,Tap,delta,Auto_corr_lamda, Cross_corr_lamda);

orig_error = y(1:length(recov)) - ref;
error = y(1:length(recov)) - recov;

plot_adaptive_sim(target,ref,f_in,f_out, recov, error, orig_error, f_coef, t, Fs);

J = sum((error .* error))/length(error);

soundsc(ref, Fs)
soundsc(recov, Fs)
```

APPENDIX E

CODE USED TO SIMULATE INTERFERENCE FOR REFERENCE SIGNAL

```

function [target ref f_in t P] = c2signalsetup(noise_file, target_file, Fs)

T = 1/Fs;
NSamples = Fs * 2.5;

target = target_file(1:NSamples);
f_in = noise_file(NSamples+11:2*NSamples+10);
den = [1 -.5 .44 .33 -.19];
num = [1 .7 .5 .24 .01];
Add = filter(num, den, noise_file(NSamples+1:2*NSamples));
ref = target + Add;

t = 0:T:(NSamples-1)*T;
w_t = 0:NSamples-1;
w_t = w_t/NSamples/1000*Fs;

f_in_FFT = fft(f_in);
target_FFT = fft(target);
ref_FFT = fft(ref);

figure;
plot(w_t(1:length(w_t)/2), abs(ref_FFT(1:length(w_t)/2)));grid on;
xlabel('Frequency(kHz)'); ylabel('Amplitude')
figure;
plot(w_t(1:length(w_t)/2), abs(target_FFT(1:length(w_t)/2)));grid on;
xlabel('Frequency(kHz)'); ylabel('Amplitude')
figure;
plot(w_t(1:length(w_t)/2), abs(f_in_FFT(1:length(w_t)/2)));grid on;
xlabel('Frequency(kHz)'); ylabel('Amplitude')
figure;
subplot(3,1,1);
plot(t, target_FFT);
subplot(3,1,2);
plot(t, f_in_FFT);
subplot(3,1,3);
plot(t, ref_FFT);
%soundsc(ref, Fs)
correlate = xcorr(f_in, ref);
figure;
plot(correlate);

pow = xcorr(f_in, f_in);
P = max(pow)/length(pow);

```

APPENDIX F

CODE USED TO DISPLAY RESULTS

```

function plot_adaptive_sim(target,ref,f_in,f_out,recov, error, orig_error,f,t, Fs)

N = length(ref);

display_offset = 1;
f_in_norm = f_in;
ref_norm = ref;
recov_norm = recov;
f_out_norm = f_out;

fft_f_in = fft(f_in_norm,N);
fft_ref = fft(ref_norm,N);
fft_recov= fft(recov_norm,N);
fft_f_out = fft(f_out_norm,N);

f_vector = 0:1:N-1;
f_vector = 100*f_vector./N;

figure;
subplot(3,2,1);
plot(abs(f_in_norm));grid on;

title('f_i_n-norm');
subplot(3,2,2);
plot(f_vector,abs(fft_f_in));grid on;
title('f_i_n-norm FFT');
xlim([0 20]); subplot(3,2,3);

plot(abs(ref_norm));grid on;

title('ref-norm');
subplot(3,2,4);
plot(f_vector,abs(fft_ref));grid on;
title('ref-norm FFT');
xlim([0 20]);

subplot(3,2,5);
plot(abs(recov_norm));grid on;

title('recov-norm');
subplot(3,2,6);
plot(f_vector,abs(fft_recov));grid on;
title('recov-norm FFT');
xlim([0 20]);

[f_row f_col] =size(f);
%
figure;
for i = 1 : f_row

```

```

subplot(f_row,1,i);
plot(f(i,:));grid on;
title(['Adaptive filter coefficients; f(' num2str(i) ') ']);
xlim([f_row+5 f_col] );
end

figure;
plot(t, f);grid on; xlabel('Time(s)'); ylabel('Amplitude');

recov_fft = fft(recov);
target_fft = fft(target);
ref_fft = fft(ref);
f_in_fft = fft(f_in);
recov_fft = recov_fft(1:length(recov_fft)/2);
target_fft = target_fft(1:length(target_fft)/2);
ref_fft = ref_fft(1:length(ref_fft)/2);
f_in_fft = f_in_fft(1:length(f_in_fft)/2);
f2_vector = f_vector(1:length(f_vector)/2)/100;

figure;
plot(t, recov, t, target);grid on; xlabel('Time(s)');ylabel('Amplitude'); axis([0 2.5 -1 1])

figure;
plot(f2_vector*Fs/1000, abs(recov_fft), f2_vector*Fs/1000, abs(target_fft));grid on;
xlabel('Frequency(kHz)'); ylabel('Amplitude')

figure;
plot(t,orig_error, t, error); grid on; xlabel('Time(s)');ylabel('Amplitude');
figure;
plot(t, recov, t, target, t, error);grid on; xlabel('Time(s)');ylabel('Amplitude');

```

BIBLIOGRAPHY

- [1] Benesty, Jacob, and Yiteng Huang. *Adaptive Signal Processing: Applications to Real-world Problems*. Berlin: Springer, 2003. Print.
- [2] Cowan, C. F. N., Peter M. Grant, and P. F. Adams. *Adaptive Filters*. Englewood Cliffs, NJ: Prentice-Hall, 1985. Print.
- [3] "DSP for FPGAs Retiming Signal Flow Graphs." XILINX. Lecture.
- [4] Honig, Michael L., and David G. Messerschmitt. *Adaptive Filters: Structures, Algorithms, and Applications*. Boston: Kluwer, 1984. Print.