Real-Time Noise Cancellation Using Adaptive Algorithms

Alaa Ali Hameed

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the Degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
September 2012
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Elvan Yılmaz
Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Muhammed Salamah
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Prof. Dr. Hasan Kömürcügil
Supervisor

Examining Committee

1. Prof. Dr. Hakan Altınçay
2. Prof. Dr. Hasan Kömürcügil
3. Assoc. Prof. Dr. Ekrem Varoğlu

ABSTRACT

The contamination of a signal of interest by other undesired signals (noise) is a problem encountered in many applications. The conventional linear digital filters with fixed coefficients exhibit a satisfactory performance in extracting the desired signal when the signal and noise occupy fixed and separate frequency bands. However, in most applications, the desired signal has changing characteristics which requires an update in the filter coefficients for a good performance in the signal extraction. Since the conventional digital filters with fixed coefficients do not have the ability to update their coefficients, adaptive digital filters are used to cancel the noise. The mean square error (MSE) technique is used as a measure of the noise reduction.

The adaptive filter generally uses finite impulse response (FIR) least-mean-square (LMS) and normalized LMS (NLMS) algorithms in signal processing or infinite impulse response (IIR) recursive-least-squares (RLS) algorithm in adaptive control for the noise cancellation applications.

The main aim of this thesis is to investigate the implementation of a real time noise cancellation application. The real time implementation is carried out by a Texas Instruments (TI) TMS320C6416T Digital Signal Processor (DSP). First, the LMS, NLMS and RLS algorithms are simulated using SIMULINK of MATLAB. Then, these algorithms have been transferred to the DSP board which let, them to work alone in real time independent of MATLAB. Furthermore, the performance of the aforementioned algorithms has been compared in different problem settings.

Keywords: Adaptive Filters, FIR Filters, IIR Filters, LMS Algorithm, NLMS Algorithm, RLS Algorithm.

ÖZ

Bir sinyalin, istenmeyen sinyal (gürültü) tarafından kirlenmesi birçok uygulamada karşı karşıya kalınan bir problemdir. Geleneksel sabit katsayılı doğrusal sayısal süzgeçler, sinyal ve gürültü sabit ve ayrı frekans bandlarını işgal ettiği zaman, istenen sinyalin elde edilmesi için yeterli bir performans sergilerler. Bununla birlikte, birçok uygulamada, istenen sinyalin değişen karakteristikerinden dolayı sinyal elde işleminde iyi bir performans elde etmek için süzgeç katsayılarında bir güncellemeye ihtiyaç duyulmaktadır. Geleneksel sabit katsayılı sayısal süzgeçlerin katsayılarını güncelleme yeteneği olmadığı için gürültüyü yok etmek için uyarlanabilir sayısal süzgeçler kullanılmaktadır. Ortalama–kare-hata tekniği gürültü azaltma ölçümü olarak kullanılır.

Uyarlanabilen sayısal süzgeç, genellikle sonlu-dürtü-cevabı (FIR) enaz-ortalama-kare (LMS) ve normalize olmuş LMS (NLMS) algoritmalarını sayısal sinyal işleme alanında veya sonsuz-dürtü-cevabı (IIR) tekrarlanan-enaz-kare (RLS) algortimasını gürültü yoketme uygulamalarında kullanır.

Bu tezin esas amacı gerçek zamanda bir gürültü yoketme uygulamasını araştırmaktır. Gerçek zaman uygulaması, Texas Instruments TMS320C6416T sayısal sinyal işlemcisi ile MATLAB'ın Simulink ortamında yapılmıştır. İlk olarak, LMS, NLMS ve RLS algoritmalarının benzetimi yapılmıştır. Daha sonra, bu algoritmalar sayısal sinyal işlemcisine transfer edilerek sayısal sinyal işlemcisinin MATLAB'dan bağımsız olarak gerçek zamanda kendi başına çalışması sağlanmıştır. Ayrıca, adı geçen algoritmaların performansı farklı problemler için karışlaştırılmıştır.

Anahtar kelimeler: Uyaranabilir süzgeçler, FIR süzgeçler, IIR süzgeçler, LMS algoritması, NLMS algoritması, RLS algoritması

Dedicated to my Family

ACKNOWLEDGMENT

I am sincerely thankful to my supervisor, Prof. Dr. Hasan Kömürcügil for his encouragement, help and support throughout this work. He devoted his time for helping me to explore knowledge in Digital Signal Processing and Adaptive Filtering with many motivation and careful review.

I would like to extend my gratitude to my monitoring jury members; Prof. Dr. Hakan Altinçay, Assoc. Prof. Dr. Ekrem Varoğlu. I have to thank them for the time they take to critically review my work and provide me useful and meaningful corrections.

I would like to extend my gratitude to my friends and colleagues from the Computer Engineering Department of Eastern Mediterranean University, for their help during my studies and publication of this thesis.

I owe a lot thankful message to my friends, Aliyu Kabir Musa, Ahmed Alyousif, Moin Naim, Hisham Ahmad, Husam Naufal, Bashar Mahmoud and Yousef Al-Shamma for their friendship, concern and moral support during my thesis.

I would like to express my deepest gratitude to my family, brothers (Mohammed Ali Hameed, Ahmed Ali Hameed, Hussein Ali Hameed and Hassan Ali Hameed) and sisters, who undoubtedly has given me the support and help to accomplish this work, they have a special place in my heart and indeed I will always be proud of them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Introduction

In the previous years, adaptive filters have been attracting the attention of many people due to their self-designing properties [1]-[4]. When some a priori knowledge about the statistics of the signal is available, an optimal filter for such application can be designed (i.e. Wiener Filter which minimizes the mean-square-error (MSE); the difference between the filter output and the desired response) [1]. If this a priori knowledge is not available, adaptive filtering algorithms possess the ability to adapt the filter coefficients to be compatible with the involved signal statistics. Hence, adaptive filtering algorithms have been used in many fields such as signal processing [1], communications systems [5], and control systems [3].

Adaptive filtering process consists of two major steps; filtering process which produces an output signal (response) from the input signal, and adaptation process; which adjusts the coefficients of the filter in a way in order to minimize a function called the cost function. Basically, there are many filter structures and filtering algorithms that are used in adaptive filtering applications.

Adaptive filters usually classified to two categories depending on their impulse response: finite-impulse-response (FIR) adaptive filter [6]; this filter's impulse response has a finite duration since it goes to zero after a finite time, and infinite impulse-response (IIR) adaptive filter [6]; which has an internal feedback mechanism and continues to respond indefinitely. IIR filters have diverse applications and are beyond the scope of this thesis.

FIR adaptive filters have different structures; adaptive transversal filters structure, the lattice predictor structure and the systolic array structure [1]. The adaptive structure of a transversal filter is shown in Fig. 1.1. The input vector tap at time *n* is denoted by u(n), the weight vector w(n) = $[w_0(n), w_1(n), \ldots, w_{N-1}(n)]^T$, and the desired response estimate (the filter output) is denoted by $\hat{d}(n|Un)$, where *Un* is the space spanned by the tap inputs *u(n), u(n−1), . . . , u(n−N+1).* By comparing the filter response *y(n)* with the real desired response *d(n)*, we produce an estimation error denoted by $e(n) = d(n) - \hat{d}(n|Un)$. The control mechanism adaptively adjusts the filter coefficients in order to obtain the desired response. Mathematically, this is interpreted as:

$$y(n) = \sum_{i=0}^{N-1} u(n-i) w_i \tag{1.1}$$

And

$$Y(z) = U(z) \sum_{i=0}^{N-1} z^{-i} w_i \tag{1.2}$$

Then the transfer function is:

$$H(z) = \frac{Y(z)}{U(z)} = \sum_{i=0}^{N-1} z^{-i} w_i \tag{1.3}$$

The impulse response of the transversal filter is h(n) = {w_0,w_1,………,w_{N-1}}.

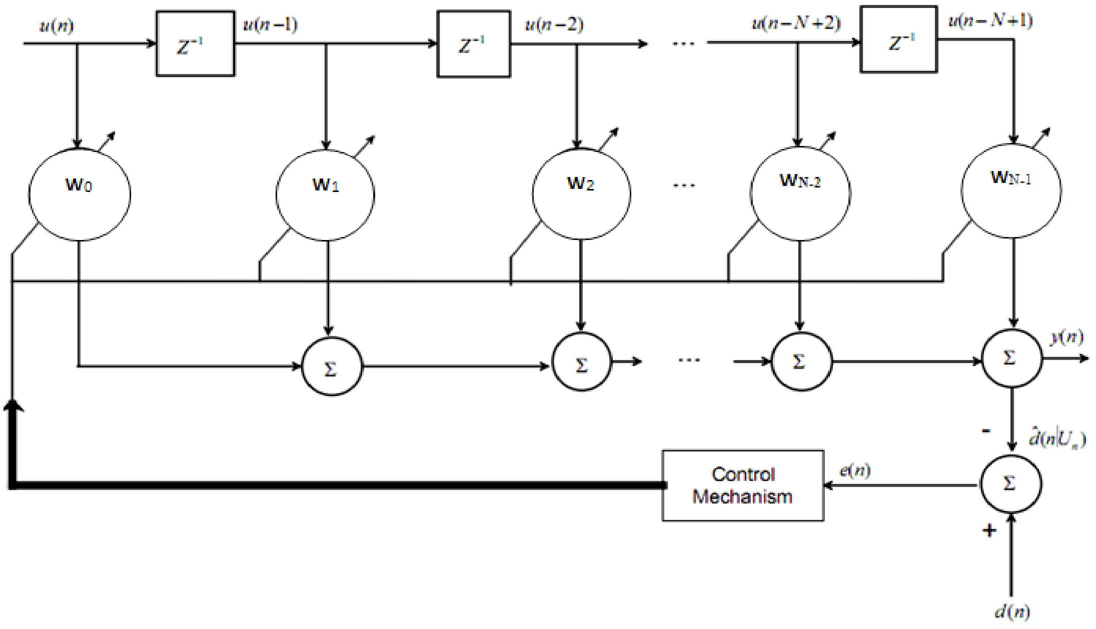


Figure 1.1 Structure of the adaptive transversal filter.

1.2 Objective of the Thesis

The main objective of this thesis is to investigate the implementation of a real time noise cancellation application. The real time implementation has been carried out by a Texas Instruments (TI) TMS320C6416T Digital Signal Processor (DSP).

First, the LMS, NLMS and RLS algorithms are simulated using SIMULINK of MATLAB. Then, these algorithms are transferred to the DSP board which let them to work alone in real time independent of MATLAB. Furthermore, the performances of the aforementioned algorithms are compared in different problem settings with mainly two input signals: (a) sinusoidal input signal with noise and (b) music input signal with noise.

Chapter 2

DIGITAL FILTERS

2.1 Filters

A filter is any device or system that takes a mixture of inputs and processes them to give corresponding required outputs. In communication systems, the term filter refers to a system that reshapes the frequency components of an input to give an output signal with desirable features. Filters are classified according to the linearity properties as linear and non-linear filters. In our research, we are going to discuss the linear adaptive filters.

2.2 Adaptive Filter Structures

Adaptive filtering process involves two basic steps:

1. A filtering process; which is designed to produce a desired output in response to an input data.
2. An adaptive process; aims to provide a mechanism for adjusting a set of the filter coefficients.

Generally, there are two types of digital filters, as mentioned in Chapter 1; FIR filters and IIR filters. IIR filters are beyond the scope of this thesis. FIR filters will be discussed in detail in the next sections.

2.2.1 Finite Impulse Response Filters

There are three types of filter structures that distinguish themselves in the context of an adaptive filter with finite impulse response. The three filter structures are as follows [1]:

1. Transversal Filter: consists of three basic elements, as in Fig. 2.1:

   (a) unit-delay element (z^{-1})

   (b) multiplier

   (c) Adder

The number of delay elements, shown as $N - 1$ in Fig. 2.1, is commonly referred to as the order of the filter. Each multiplier in the filter is used to multiply each tap input (to which it is connected) by a filter coefficient or a tap weight. Thus, a multiplier connected to the k^{th} tap input $u(n-k)$ produces the inner product $w_k(n- k)$, where w_k is the corresponding tap weight and k = 0, 1, . . . ,N-1. The role of each adder in the filter is to sum the multiplier outputs and to produce a total filter output as in (2.1).

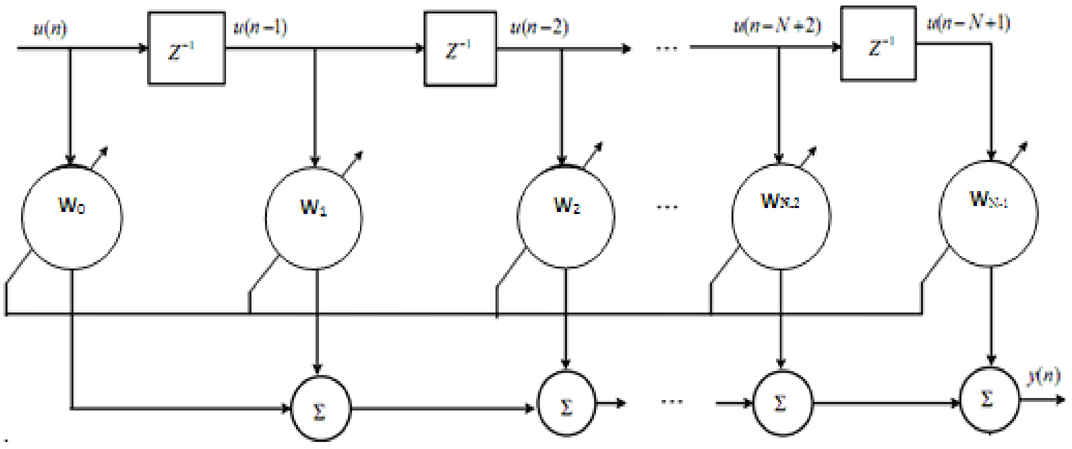


Figure 2.1 Transversal Filters.

$$y(n) = \sum_{k=0}^{N-1} w_k u(n-k), \qquad (2.1)$$

2. Lattice Predictor: is modular in structure in that it consists of a number of separate stages, each looks as a lattice. Fig. 2.2 shows an N −1 stages lattice predictor; the number N − 1 refers to the predictor order. The *m*th stage of a lattice predictor is described by the pair of input-output relations:

$$f_m(n) = f_{m-1}(n) + \Gamma^*_m \; m b_{m-1}(n-1), \qquad (2.2)$$

$$b_m(n) = b_{m-1}(n-1) + \Gamma_m f_{m-1}(n), \qquad (2.3)$$

where $m = 1, 2, \ldots, N-1$, where $N-1$ is the final predictor order. The variable $f_m(n)$ is the *m*th forward prediction error, and $b_m(n)$ is the *m*th backward prediction error. The coefficient Γ_m is called the mth reflection coefficient. The forward prediction error $f_m(n)$ is defined as the difference between the input $u(n)$ and its one-step predicted value. Correspondingly, the backward prediction error $b_m(n)$ is defined as the difference between the input $u(n-N)$ and its "backward" prediction based on the set of m "future" inputs $u(n), \ldots, u(n-N+1)$.

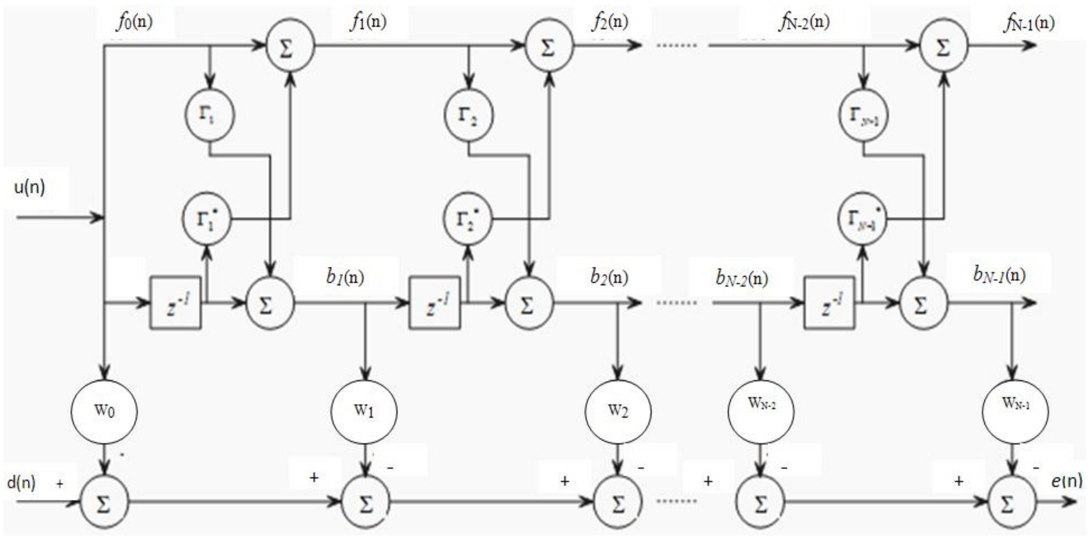


Figure 2.2 Multistage Lattice Predictor.

3. Systolic Array: consists of a parallel computing network which is used to map a number of linear algebra operations, such as matrix multiplication, triangularization, and back substitution. Basically, two types of processing elements may be distinguished in a systolic array: boundary cells and internal cells. In each case, the parameter *r* represents a value stored within the cell. The function of the boundary cell is to produce a response equal to the input *u* divided by *r* which is a number stored in the cell. The function of the internal cell is: (a) to multiply the input *z* by the number *r* stored in the cell, subtract product *rz* from the second input, and thereby produce the difference *u* – *rz* as an output from the right side of the cell, and (b) to transmit the first input *z* downward without alteration.

Chapter 3

ADAPTIVE FILTERS AND NOISE CANCILLATION

3.1 Introduction

Digital Signal Processing (DSP) is the major technology that can be applied to noise filtering, system identification, and voice prediction. Standard DSP techniques are not enough to solve these problems quickly and obtain acceptable results. Adaptive filtering techniques must be implemented to obtain accurate solutions with timely convergence.

3.2 Adaptive Filtering System Configurations

Adaptive filter had first established its engineering use in 1960s. It was applied as an equalizer to combat the effect of Inter-Symbol Interference (ISI) of data transmission in telephone channels [1]. Since then, adaptive filter was modified into different forms and applied in many different areas such as; signal processing and communication systems.

There are four major types of adaptive filtering configurations; adaptive system identification [8], adaptive noise cancellation, adaptive linear prediction [9], and adaptive inverse system [7]. All of the above systems are similar in the implementation of the algorithm, but different in system configuration. All four systems have the same general components; an input signal x(n), a desired result *d*(*n*), an output *y(n)*, an adaptive transfer function w(n), and an error signal *e(n)* which is the difference between the desired output *d(n)* and the actual output y(n). In addition to these, the system identification and the inverse system configurations have an unknown linear system u(n) that can produce a linear output to the given input [1].

3.2.1 Adaptive System Identification Configuration

The adaptive system identification is primarily responsible for determining a discrete estimation of the transfer function for an unknown digital or analog system. The same input x(n) is applied to both the adaptive filter and the unknown system from which the outputs are compared, as shown in Fig. 3.1. The output of the adaptive filter *y(n)* is subtracted from the output of the unknown system (which results in the desired response signal *d*(*n*)). The resulting difference is an error signal e(*n*) which is used to manipulate the filter coefficients of the adaptive system. After convergence, the error signal tends toward zero.

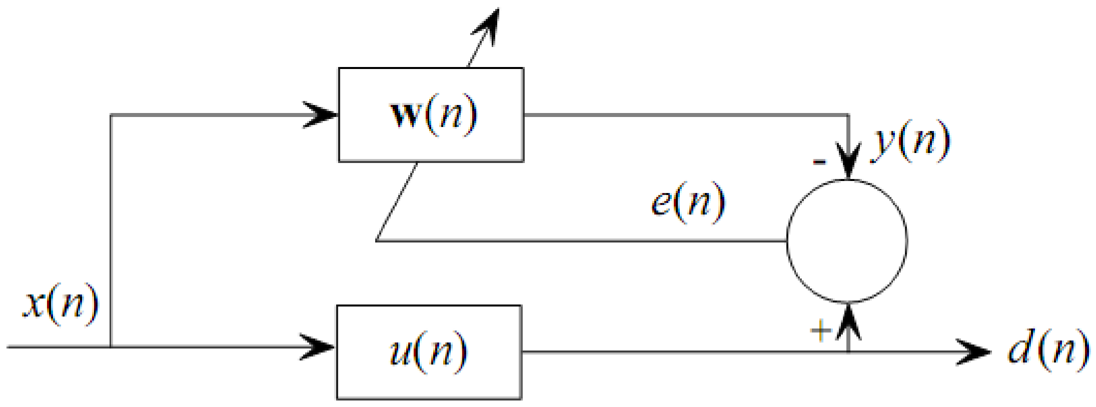


Figure 3.1 Adaptive System Identification Configuration.

After a number of iterations of this process, the adaptive filter's transfer function will converge to, or near to, the unknown system's transfer function. For this configuration, the error signal does not have to go to zero (although convergence to zero is the ideal situation) to closely approximate the given system. There will, however, be a difference between the adaptive filter transfer function and the unknown system transfer function if the error is nonzero and the magnitude of that difference will be directly related to the magnitude of the error signal.

3.2.2 Adaptive Noise Cancellation Configuration

The second configuration is the adaptive noise cancellation configuration as shown in Fig. 3.2. In this configuration, the input x(n) (a noise source $N_1(n)$), is compared with a desired signal $d(n)$, which consists of a signal $s(n)$ corrupted by another noise signal $(N_0(n))$. The adaptive filter coefficients adapt to cause the error signal to be a noiseless version of the signal $s(n)$.

Both of the noise signals for this configuration need to be uncorrelated to the signal $s(n)$. In addition, the noise sources must be correlated to each other in some way, preferably equal, to get the best results [2]. Assuming that $y(n) \approx N_1(n)$, $d(n) = s(n) + N_0(n)$ and the error can be written as $e(n) = s(n) + N_0(n) - y(n)$ since, noise sources are correlated to each other the error reduces to $e(n) = s(n)$.

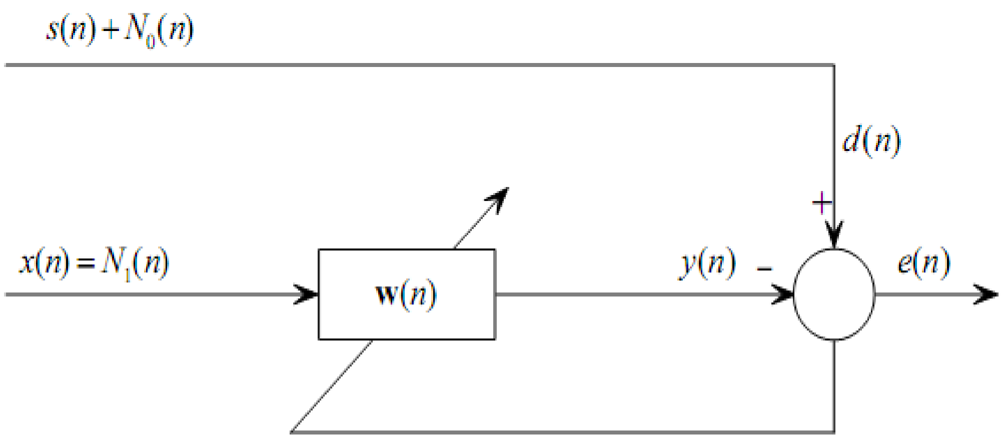


Figure 3.2 Adaptive Noise Cancellation Configuration.

3.2.3 Adaptive Linear Prediction Configuration

Adaptive linear prediction is the third type of adaptive configuration as shown in Fig. 3.3. This configuration essentially performs two operations: The first operation, is linear prediction; if the output is taken from the error signal *e(n)*. The adaptive filter coefficients are being trained to predict, from the statistics of the input signal x(n), what the next input signal will be. The second operation, is a noise filter similar to the adaptive noise cancellation outlined in the previous section; if the output is taken from *y(n).*

In the case of noise filtering, as outlined in the previous section, *y*(n) will converge to the noiseless version of the input signal.

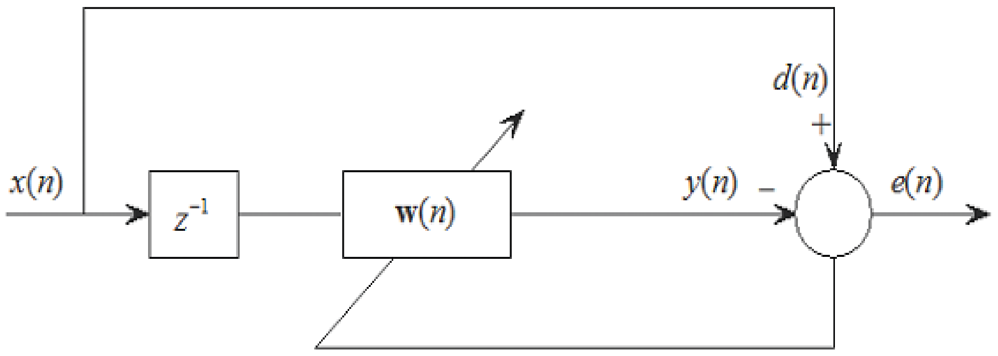


Figure 3.3 Adaptive Linear Prediction Configuration.

3.2.4 Adaptive Inverse System Configuration

The final filter configuration is the adaptive inverse system configuration as shown in Fig. 3.4. The goal of the adaptive filter here is to model the inverse of the unknown system *u*(n). This is particularly useful in adaptive equalization where the goal of the filter is to eliminate any spectral changes that are caused by a prior system or transmission line.

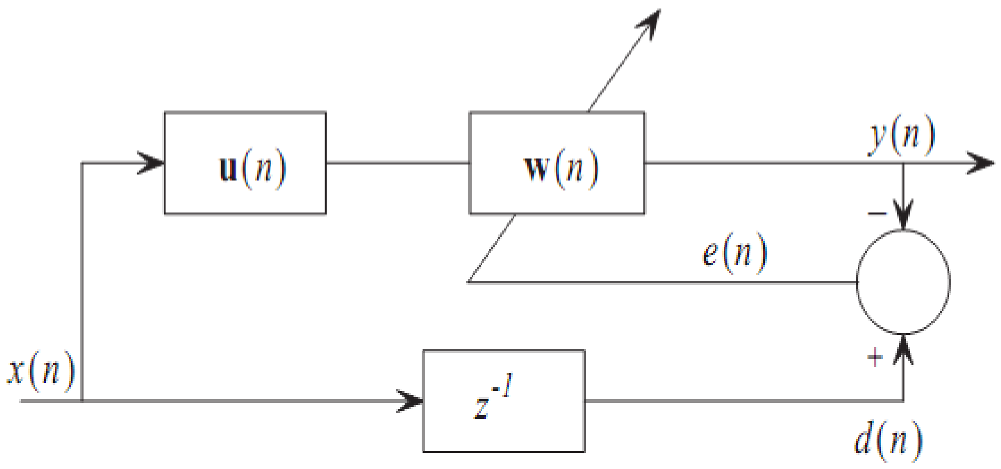


Figure 3.4 Adaptive Inverse System Configuration.

The way this filter works is as follows; the input x(*n*) is sent through the unknown system *u(n)* and then through the adaptive filter resulting in an output *y(n)*. The input is also sent through a delay to attain *d(n)*. As the error signal is converging to zero, the adaptive filter coefficients w(*n*) are converging to the inverse of the unknown system *u(n)*.

For this configuration, the error can theoretically go to zero. This is only true if the unknown system consists only of a finite number of poles or the adaptive filter is an Infinite Impulse Response (IIR) filter. If neither of these conditions is true, the system will converge only to a constant due to the limited number of zeroes available in a Finite Impulse Response FIR system [1].

3.3 Performance Measures in Adaptive Systems

Some important measures will be discussed in the following sections; convergence rate, minimum mean square error, computational complexity, stability, and filter length [2].

3.3.1 Convergence Rate

The Convergence rate determines the rate at which the filter converges to its resultant state. Usually a faster convergence rate is a desired characteristic of an adaptive system. Convergence rate is not independent of all the other performance characteristics. There is usually a tradeoff, with convergence rate and other performance criteria [2].

3.3.2 Mean Square Error

The MSE is a metric indicating how much a system can adapt to a given solution. A small MSE is an indication that the adaptive system has accurately modeled, predicted, adapted and/or converged to a solution for the system. There are a number of factors which will help to determine the MSE including, but not limited to; quantization noise, order of the adaptive system, measurement noise, and error of the gradient due to the finite step size [2].

3.3.3 Computational Complexity

Computational complexity is particularly important in real time adaptive filter applications. When a real time system is being implemented, there are hardware limitations that may affect the performance of the system. A highly complex algorithm will require much greater hardware resources than a simplistic algorithm [2].

3.3.4 Stability

Stability is probably the most important performance measure for the adaptive system. By the nature of the adaptive system, there are very few completely asymptotically stable systems that can be realized. In most cases, the systems that are implemented are marginally stable, with the stability determined by the initial conditions, transfer function of the system and the step size of the input [2].

3.3.5 Filter Length

The filter length of the adaptive system is inherently tied to many of the other performance measures. The length of the filter specifies how accurately a given system can be modeled by the adaptive filter. In addition, the filter length affects the convergence rate, by increasing or decreasing computation time, it can affect the stability of the system, at certain step sizes, and it affects the MSE. If the filter length of the system is increased, the number of computations will increase, decreasing the maximum convergence rate [2].

Chapter 4

ADAPTIVE FILTERING ALGORITHMS

4.1 Introduction

Adaptive filtering methods are generally used to cope with the changes in the system parameters [4]. In FIR adaptive filters, the filter coefficients are iteratively updated by minimizing the difference between the desired response and the output of the adaptive filter. Before starting the discussion of the adaptive algorithms that we will see in this thesis, an optimization technique called the *steepest descent* will be presented.

4.2 Steepest-Descent Method

This is a recursive method since it starts from some initial (arbitrary) values of the weights vector and it improves as the number of iterations increases. The important thing to note is that the steepest descent method is descriptive of multiparameter closed-loop deterministic control system which finds the minimum point of the ensemble-averaged error-performance surface without the knowledge of the surface itself [1].

Considering a transversal filter having the tap inputs $u(n)$, $u(n-1)$, . . . , $u(n-N+1)$ and a set of tap weights $w_0(n)$,$w_1(n)$, . . . ,$w_{N-1}(n)$. The vector of the tap inputs represents samples drawn from a wide-sense stationary process of zero mean and correlation matrix $\mathbf{R} = \mathbf{u}(n)\mathbf{u}^H(n)$, where H represents Hermitian transpose. Also the filter has a desired response $d(n)$ that provides a frame of reference for the optimum filtering action; this is illustrated clearly in Fig. 4.1.

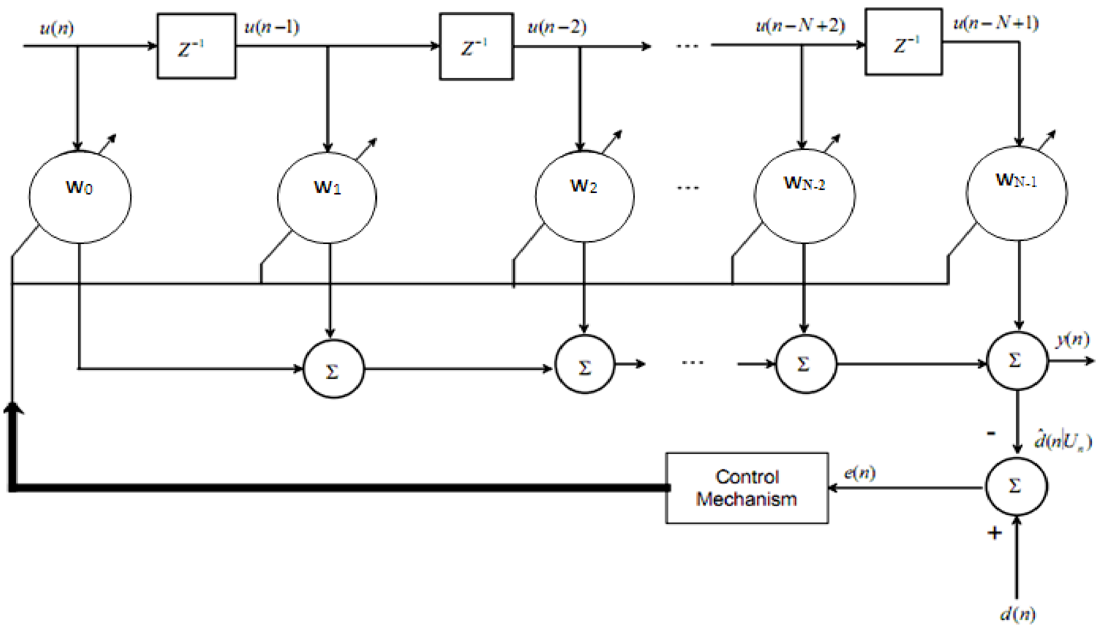


Figure 4.1 Adaptive transversal filter's structure.

The tap inputs vector at time n is denoted by $\mathbf{u}(n)$, and the estimate of the filter output, which is called the desired response, is denoted by $\hat{d}(n|Un)$, where Un is the space spanned by the tap inputs $u(n)$, $u(n-1)$, . . . , $u(n-N+1)$. By comparing this with the actual desired response $d(n)$, an estimation error denoted by $e(n)$ is produced.

$$e(n) = d(n) - \hat{d}(n|U_n) = d(n) - \mathbf{w}^H(n)\mathbf{u}(n), \qquad (4.1)$$

where the inner product of the coefficients vector **w**(n) and the tap input vector **u**(n) is given by the term $\mathbf{w}^H(n)\mathbf{u}(n)$. The coefficients vector, the tap-input vector and the cost function are, respectively, denoted by:

$$w(n) = [w_0(n)\ w_1(n) \ldots w_{N-1}(n)]^T, \tag{4.2}$$

$$u(n) = [u(n)\ u(n-1) \ldots u(n-N+1)]^T, \tag{4.3}$$

$$J(n) = E\{|e(n)|^2\}, \tag{4.4}$$

where E[.] denotes the expectation operator. If the tap-input vector **u**(n) and the desired $d(n)$ are jointly stationary (i. e., If x and y are jointly stationary then $ax+by$ is stationary for any constants a and b), then the mean-squared error or cost function $J(n)$ at time n could be written as:

$$J(n) = \sigma_d^2 - \mathbf{w}^H(n)\mathbf{p} - \mathbf{p}^H\mathbf{w}(n) + \mathbf{w}^H(n)\mathbf{R}\mathbf{w}(n), \tag{4.5}$$

where;

σ_d^2= variance of the desired response $d(n)$.

p = the vector representing the cross-correlation between the tap-input vector **u**(n) and the desired response $d(n)$.

R = the correlation matrix of the tap-input vector **u**(n).

Equation (4.5) represents the mean-squared error. This error would result if the coefficients vector of the filter is kept fixed at the value **w**(n). Since **w**(n) varies with time n, the mean-squared error naturally varies with time n in a corresponding manner. A result of this, the cost function ($J(n)$) for the mean-squared error is used in that equation. The change in the mean-square error $J(n)$ with time n means that the estimation error process $e(n)$ is non-stationary [1].

The dependence of the mean-squared error $J(n)$ on the entries of the filter coefficients vector $\mathbf{w}(n)$ as a bowl-shaped surface with a unique minimum is visualized. This is called as the surface error of the adaptive filter. This occurs when the tap-weight vector takes on the optimum value $\mathbf{w}_0$ [1]. We define:

$$\mathbf{R}\mathbf{w}_0 = \mathbf{p}, \tag{4.6}$$

and the minimum mean-squared error is:

$$J_{min} = \sigma_d^2 - \mathbf{p}^H\mathbf{w}_0, \tag{4.7}$$

The *Steepest-Descent Algorithm* [2] is relatively straightforward; nevertheless, it has serious difficulties in the computations, especially, when the filter contains a large number of coefficients and when the input vector has relatively large values. This implies that we can use the steepest-descent method to find the minimum value of the function of the mean-squared error J_{min} as follows:

1. Start with an initial value $\mathbf{w}(0)$ for the filter coefficients vector, which is chosen arbitrarily. The value $\mathbf{w}(0)$ gives us an initial guess as to where the minimum point of the error-performance surface may be located. Usually, $\mathbf{w}(0)$ is set equal to the null vector.
2. Using this assumption, we compute the gradient vector, the real and imaginary parts of which are defined as the derivative of the mean-squared error $J(n)$,

evaluated with respect to the real and imaginary parts of the tap-weight vector **w**(n) at time n.

3. Compute the next guess of the tap-weight vector by changing the present guess in a direction opposite to that of the gradient vector.
4. Go back to step 2 and repeat the process.

Let $\nabla(J(n))$ denote the value of the gradient vector at time n. Let **w**(n) denote the value of the filter coefficients vector at time $n + 1$, computed using the recursive relation given by:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + 1/2\, \mu[-\nabla(J(n))], \tag{4.8}$$

where μ is a positive real-valued constant, and

$$\nabla(J(n)) = \begin{pmatrix} \frac{\partial J(n)}{\partial a_0(n)} & j\frac{\partial J(n)}{\partial b_0(n)} \\ \frac{\partial J(n)}{\partial a_1(n)} & j\frac{\partial J(n)}{\partial b_1(n)} \\ \frac{\partial J(n)}{\partial a_2(n)} & j\frac{\partial J(n)}{\partial b_2(n)} \\ \vdots & \vdots \\ \frac{\partial J(n)}{\partial a_{N-1}(n)} & j\frac{\partial J(n)}{\partial b_{N-1}(n)} \end{pmatrix} = -2\mathbf{p} + 2\mathbf{R}\mathbf{w}(n), \tag{4.9}$$

where $\frac{\partial J(n)}{\partial a_k(n)}$ and $\frac{\partial J(n)}{\partial b_k(n)}$ are partial derivatives of the cost function $J(n)$ with respect to real part $a_k(n)$ and the imaginary part $b_k(n)$ of the kth tap weight $w_k(n)$, respectively. For the application of the steepest-descent algorithm, we assume that in (4.9), the correlation matrix **R** and the cross-correlation vector **p** are known, so we may compute the gradient vector $\nabla(n)$ for a given value of the tap-weight vector $\mathbf{w}(n)$. Substituting (4.9) in (4.8) we will get the updated value of the tap-weight vector by using the simple recursive relation:

$$\mathbf{w}(n + 1) = \mathbf{w}(n) + \mu[\mathbf{p} - \mathbf{R}\mathbf{w}(n)] \quad n = 1, 2, 3, \ldots \tag{4.10}$$

It is known that the parameter μ controls the size of the incremental correction applied to the tap-weight vector as we proceed from one iteration cycle to the other. We call μ the *step-size parameter or weighting constant*. Equation (4.10) provides the mathematical description of the steepest-descent algorithm.

According to (4.10), the correction $\delta\mathbf{w}(n) = \mathbf{w}(n + 1) - \mathbf{w}(n)$ applied to the tap-weight vector at time $n + 1$ is equal to $\mu[\mathbf{p} - \mathbf{R}\mathbf{w}(n)]$. This correction may be expressed as μ times the expectation of the inner product of the input vector $\mathbf{u}(n)$ and the estimation error $e(n)$. This suggests using a bank of cross-correlators to compute the correction $\delta\mathbf{w}(n)$ applied to the tap-weight vector $\mathbf{w}(n)$ as in Fig. 4.2. Another point is that we may view the steepest-descent algorithm of (4.10) as a feedback model.

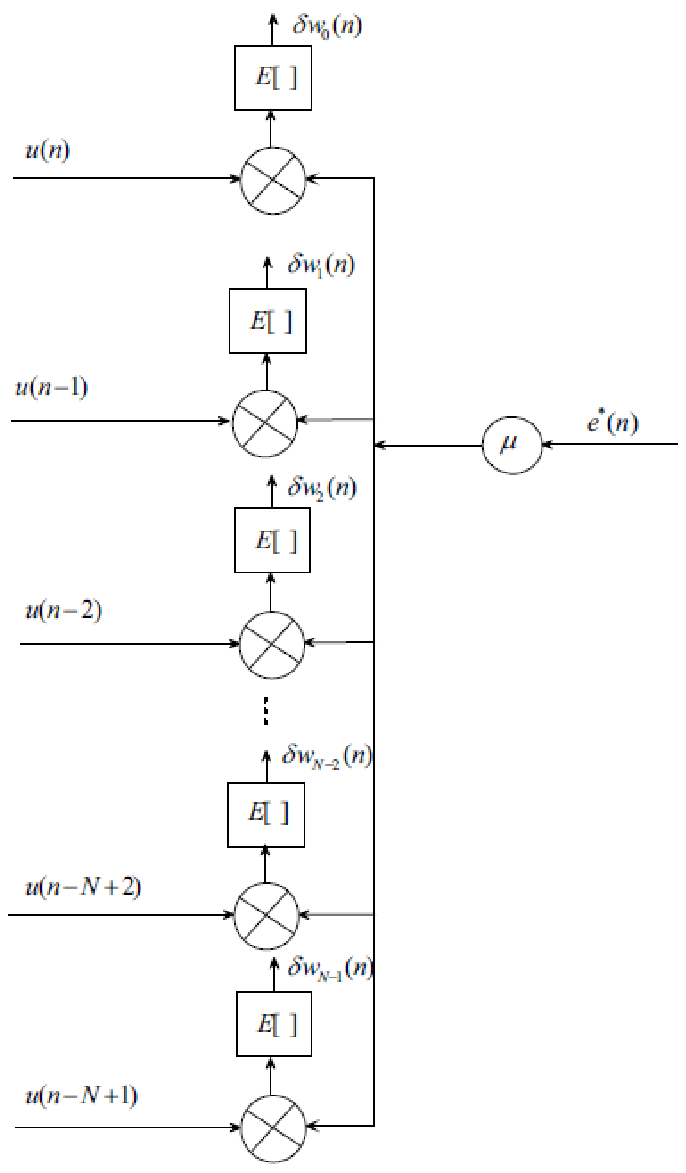


Figure 4.2. Bank of cross-correlators for computing the corrections of the elements of the tap-weight vector at n + 1.

The operation of the *least-mean-square* (LMS) *algorithm* is descriptive of *a feedback control system*. Basically, it can be subdivided into two basic processes:

1. An adaptive process, which cares about the automatic adjustment of the filter coefficients.

2. A filtering process, which cares about implementing the inner product of the filter coefficients that emerge from the adaptive process in order to provide a good estimate of the desired response, and generate an estimation error by comparing this estimate with the actual value of the desired response; which in turn (the estimation error) is used to actuate the adaptive process, thereby closing the feedback loop.

We are going to identify the two basic components in the structural constitution of the LMS algorithm as in Fig. 4.3, which has a transversal filter with LMS algorithm (for filtering process), and a mechanism for adaptive control process on the tap weights of the transversal filter.

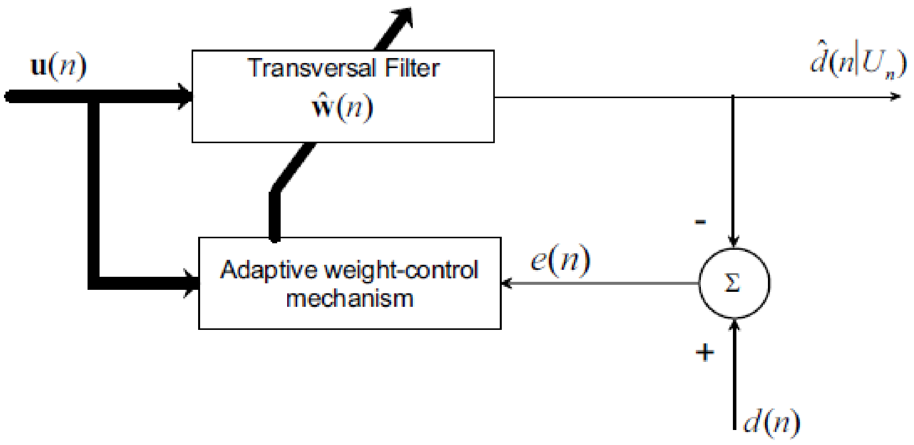


Figure 4.3 Block Diagram of Adaptive Transversal Filter.

While the filtering process is taking place, the desired response $d(n)$ is supplied for processing alongside the tap-input vector $\mathbf{u}(n)$. With this input the transversal filter produces an output $\hat{d}(n|U_n)$ used as an estimate of the desired response $d(n)$. Also we may set up an estimation error $e(n)$ as the difference between the desired response and the filter output, as in Fig. 4.4. Both $e(n)$ and $\mathbf{u}(n)$ are applied to the control mechanism, and the feedback loop around the tap weights is thereby closed.

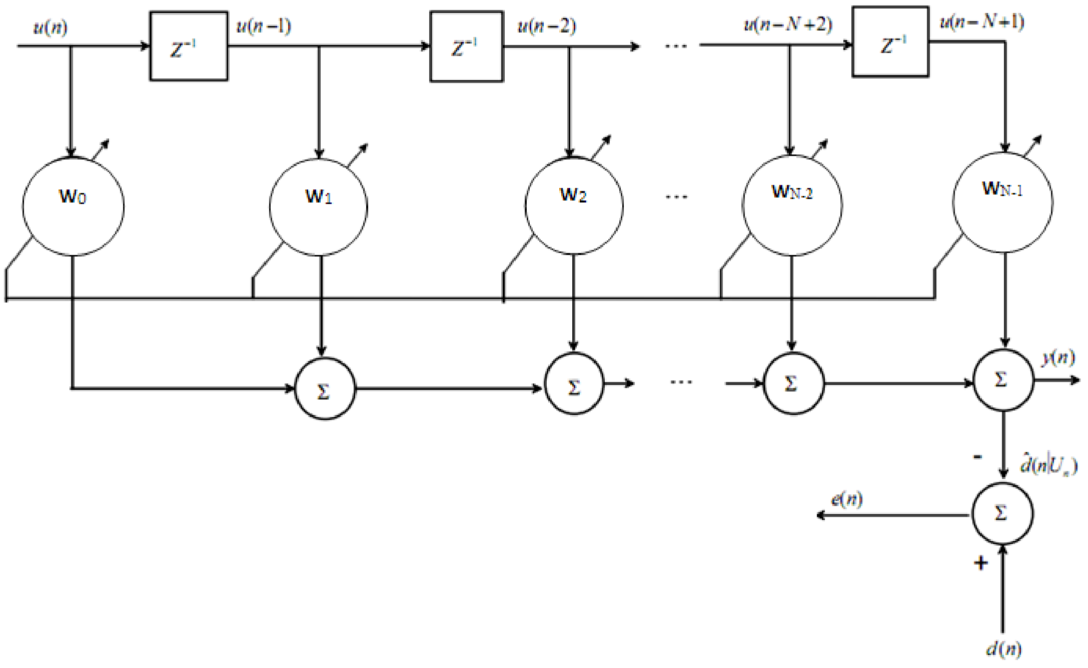


Figure 4.4 Detailed Structure of the Transversal Filter Component.

Figure 4.5 presents details of the *adaptive weight-control mechanism*. Specifically, a scaled version of the *inner product of the estimation error* $e(n)$ and tap-input $u(n-k)$ is computed for $k = 0, 1, . . . ,N - 1$. The obtained result defines the *correction* $\delta\hat{w}_k(n)$ applied to the tap weight $\hat{w}_k(n)$ at time n+1. The scaling factor μ is called the *step-size parameter or adaptation constant* (as mentioned previously).

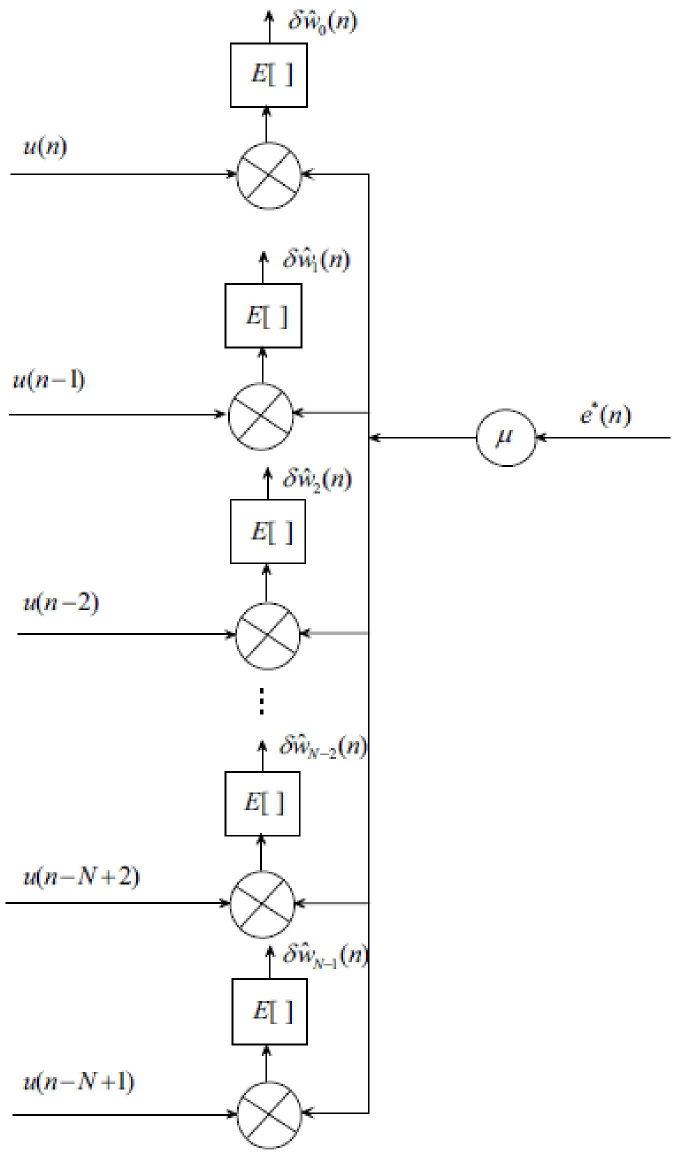


Figure 4.5. Detailed Structure of the Adaptive Weight-Control Mechanism.

Comparing Fig. 4.5 and Fig. 4.2 we see that the LMS algorithm uses the inner product $u(n-k)e^*(k)$ as an estimator of element k in the gradient vector $\nabla(J(n))$ that characterizes the steepest-descent method. The recursive computation of each tap weight in the LMS algorithm suffers from *gradient noise*.

The filter coefficients vector $\hat{\mathbf{w}}(n)$ which is computed by the LMS algorithm executes a *random motion* around the optimum point of the error surface. This motion motivates us to investigate two convergence behaviors of the LMS algorithm.

1. Convergence behavior in the mean sense.
2. Convergence behavior in the mean square sense.

4.3 Least-Mean-Square Adaptation Algorithm

If it was possible to make an exact measurement of the gradient vector $\nabla(J(n))$ at each time iteration, and if the step-size μ is chosen suitably, then the filter coefficients vector computed by using the steepest-descent method would indeed converge.

Exact measurements of the gradient vector are, in reality, impossible because this would require prior knowledge of the autocorrelation matrix **R** of the tap input and the cross-correlation vector **p** between the tap input vector and the desired response. As a result of this, the gradient vector must be estimated using the available data. That means the tap-weight vector according to an algorithm *adapts to the incoming data* (Least-Mean-Square (LMS) Algorithm). A significant feature of the LMS algorithm is its simplicity; it does not require measurements of the pertinent correlation functions, and it does not require matrix inversion [1].

To develop a good estimate of the gradient vector $\nabla(J(n))$, we substituted estimates of the autocorrelation matrix **R** and the cross-correlation vector **p** in (4.9).

$$\nabla(J(n)) = -2\mathbf{p} + 2\mathbf{R}\mathbf{w}(n), \tag{4.11}$$

The simplest choice of estimator for **R** and **p** is to use instantaneous estimates that are based on sample values of the tap input vector and desired response.

$$\hat{\mathbf{R}}(n) = \mathbf{u}(n)\mathbf{u}^H(n), \tag{4.12}$$

$$\hat{\mathbf{p}} = \mathbf{u}(n)d^*(n), \tag{4.13}$$

Corresponding, the instantaneous estimate equation of the gradient vector is:

$$\hat{\nabla}(J(n)) = -2\mathbf{u}(n)d^*(n) + 2\mathbf{u}(n)\mathbf{u}^H(n)\hat{\mathbf{w}}(n), \tag{4.14}$$

This estimate is generally biased because the filter coefficients estimate vector $\hat{\mathrm{w}}(n)$ is a random vector that depends on the input vector $\mathbf{u}(n)$. Noting that the estimate $\hat{\nabla}(J(n))$ can also be viewed as the gradient operator ∇ applied to the absolute instantaneous squared error $|e(n)|^2$.

Substituting the estimate of (4.14) for the gradient vector $\nabla(J(n))$ in the steepest descent algorithm as described in (4.8), we get a new recursive relation for updating the tap-weight vector:

$$\hat{\mathbf{w}}(n + 1) = \hat{\mathbf{w}}(n) + \mu\mathbf{u}(n)[d^*(n) - \mathbf{u}^H(n)\hat{\mathbf{w}}(n)] \tag{4.15}$$

Here we have used the "cap" over the symbol of the tap-weight vector to distinguish it from the value obtained by the steepest-descent algorithm. A summary of the LMS algorithm is shown in Table 4.1.

Table 4.1 Summary of the LMS algorithm.

1. Filter Output.

$$y(k) = \hat{\mathbf{w}}^H(k)\mathbf{x}(k)$$

2. Estimation Error.

$$e(k) = d(k) - y(k)$$

3. Tap – Weight Adaptation.

$$\hat{\mathbf{w}}(k+1) = \hat{\mathbf{w}}(k) + \mu e^*(k)\mathbf{x}(k)$$

where $\mathbf{x}(k)$ is the tap input vector and $d(k)$ is the desired filter output.

4.4 Normalized Least Mean Square Algorithm

In the LMS algorithm, the selection of the step-size causes a problem in many of applications where the LMS algorithm is used and when the input x(k) is large. To overcome this problem, the normalized least-mean-square (NLMS) is proposed [10]-[13]. In NLMS algorithm, the step-size μ is normalized by the energy of the data vector. A summary of the NLMS algorithm is given in Table 4.2.

Table 4.2 Summary of the NLMS algorithm

1. Filter Output.

$$y(k) = \hat{\mathbf{w}}^H(k)\mathbf{x}(k)$$

2. Estimation Error.

$$e(k) = d(k) - y(k)$$

3. Tap – Weight Adaptation.

$$\hat{\mathbf{w}}(k+1) = \hat{\mathbf{w}}(k) + \frac{\mu e^*(k)\mathbf{x}(k)}{\epsilon + \mathbf{x}^H(k)\mathbf{x}(k)}$$

where ϵ is a the regularization parameter to avoid dividing by zero.

The NLMS algorithm converges much faster than LMS algorithm with very little extra computational complexity; NLMS is very commonly used in some applications such as echo cancellation problems [14].

However, the NLMS algorithm has a problem; when the input vector x(k) is small, then a rise of numerical difficulties may occur because by then we have to divide by a small value for tap-input power $\|x(k)\|^2$.

4.5 Recursive-Least-Squares Algorithm

The recursive-least-squares (RLS) algorithm [1], [15]-[16] was proposed in order to provide superior performance compared to those of the LMS algorithm and its variants [17]-[22], with few parameters to be predefined, especially in highly correlated environments. In the RLS algorithm, an estimate of the autocorrelation matrix is used to decorrelate the current input data. Also, the quality of the steady-state solution keeps on improving over time, eventually leading to an optimal solution. A summary of the algorithm is shown in Table 4.3.

Even though the RLS algorithm has very good performance in such environments, it actually suffers from its high computational complexity $O(N^2)$. Also, In RLS algorithm, the forgetting factor (β) has to be chosen carefully such that its value should be very close to one in order to ensure stability and convergence of the RLS algorithm. However, this in turn poses a limitation for the use of the algorithm because small values of β may be required for signal tracking if the environment is non-stationary [23].

Table 4.3 Summary of the RLS algorithm

Initialize the algorithm by setting,

$$\hat{\mathbf{w}}(0) = \mathbf{0}$$

$$\mathbf{P}(0) = \delta^{-1}\mathbf{I}$$

and

$$\delta = \begin{cases} \text{small positive constant for high SNR} \\ \text{large positive constant for low SNR.} \end{cases}$$

for each instant of time, $k = 1, 2, \ldots$ compute

$$\mathbf{D}(k) = \mathbf{P}(k-1)\mathbf{x}(k)$$

$$\mathbf{k}(k) = \frac{\mathbf{D}(k)}{\beta + \mathbf{x}^H(k)\mathbf{D}(k)}$$

$$\xi(k) = d(k) - \hat{\mathbf{w}}^H(k-1)\mathbf{x}(k)$$

$$\hat{\mathbf{w}}(k) = \hat{\mathbf{w}}(k-1) + \mathbf{k}(k)\xi^*(k)$$

and

$$\mathbf{P}(k) = \beta^{-1}\mathbf{P}(k-1) - \beta^{-1}\mathbf{k}(k)\mathbf{x}^H(k)\mathbf{P}(k-1).$$

In the following chapter we will explain the adaptive noise cancellation problem in detail and implementing it using LMS, NLMS and RLS algorithms using the SIMULINK package.

Chapter 5

SIMULATIONS AND EXPERIMENTAL RESULTS

5.1 Introduction

In this thesis, the SIMULINK of *MATLAB* Software Package is used for the simulation of the standard LMS, NLMS and RLS algorithms in noise cancellation (see Fig. 3.2) configurations. Simulations discuss the performances of these algorithms with additive white Gaussian noise (AWGN) with different parameters and different input signals.

A real time implementation is carried out by a TI TMS320C6416T DSP (full details in Appendix) by transferring the SIMULINK schemes (a sample; i.e. the LMS SIMULINK schematic is shown in Fig. 5.1) to the DSP board which let it work alone in real time independent of MATLAB.

5.2 Sinusoidal Input Signal

In this experiment, a sinusoidal signal ($s(n) = A\sin(2\pi ft)$) is created with the following parameters: frequency f = 5Hz and amplitude A = 0.4 as shown in Fig. 5.2. Then, an AWGN (shown in Fig. 5.3) with zero mean and variance σ^2 = 0.03 is added to the input signal. The resulting signal is assumed to be the received signal ($d(n)=s(n)+N(n)$) and is shown in Fig. 5.4.The performances of the three algorithms are compared with different parameters and different filter lengths.

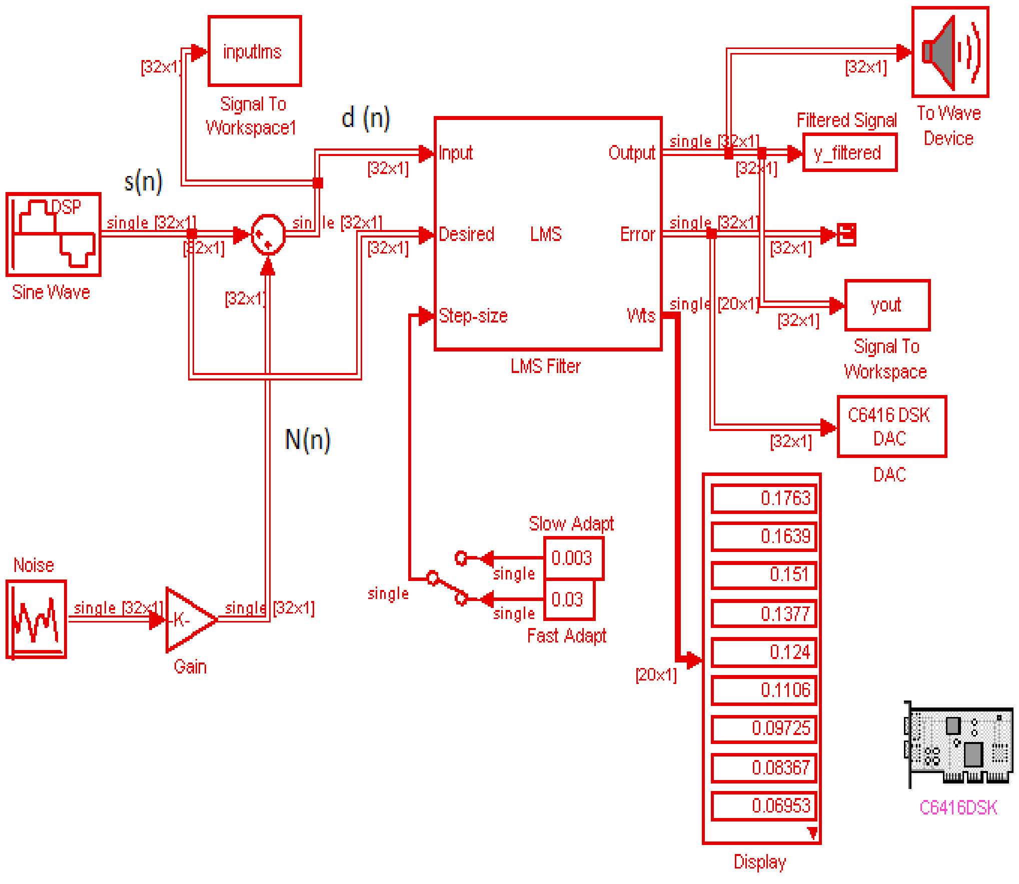


Figure 5.1 SIMULINK schematic for LMS algorithm.

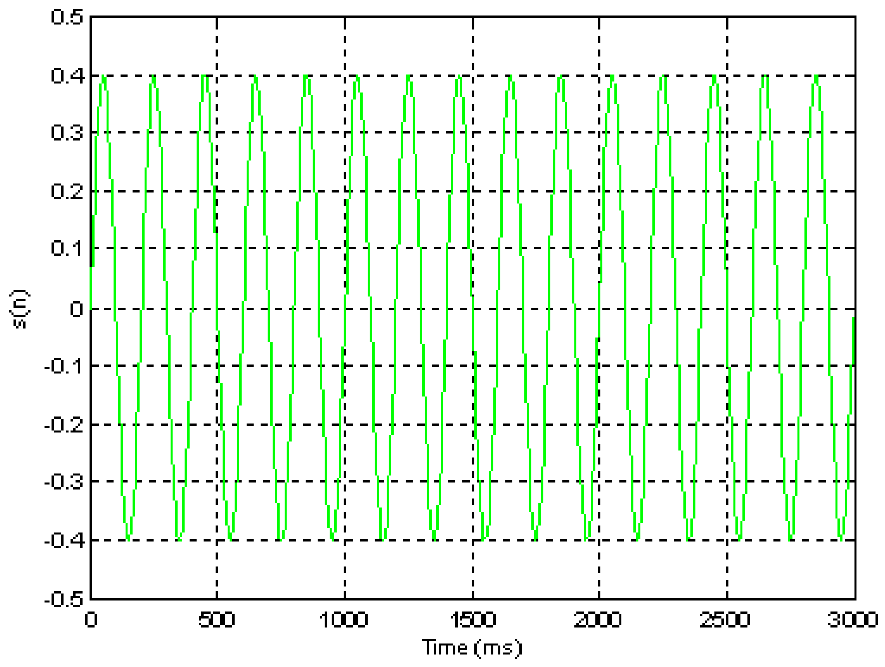


Figure 5.2 Desired sinusoidal signal (s(*n*)).

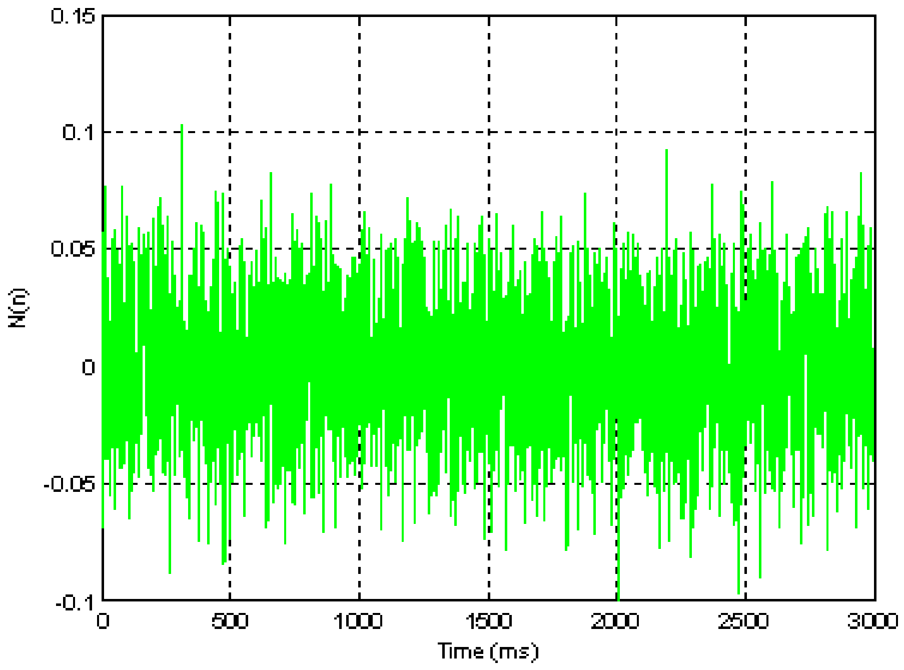


Figure 5.3 Additive White Gaussian Noise (*N*(*n*)).

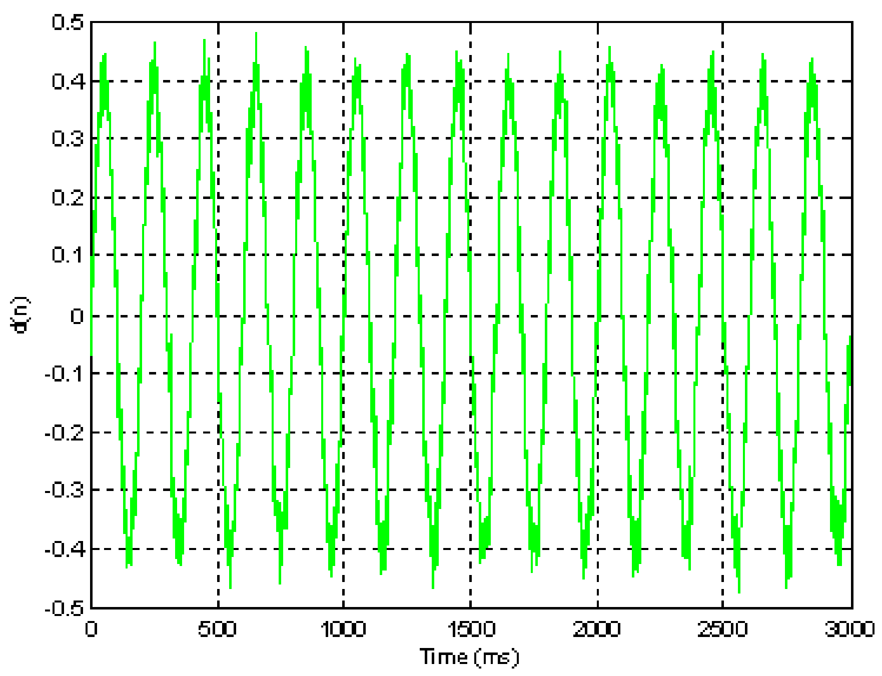


Figure 5.4 Input sinusoid with additive white gaussian noise.

In the first part of this experiment, the filter length is assumed to be N = 20 taps for all algorithms, μ = 0.03 for LMS and NLMS algorithms and for the RLS algorithm β = 1. Fig. 5.5 shows the MSE of all algorithms. From the figure we see that the RLS algorithm provides the fastest convergence rate and lowest MSE compared to the other algorithms. NLMS algorithm converges to the same MSE as that of the RLS algorithm with lower convergence rate. Even though the LMS algorithm converges to the same MSE of the other algorithms, but it has the lowest convergence rate. Fig. 5.6 shows the recovered sinusoid by the aforementioned algorithms.

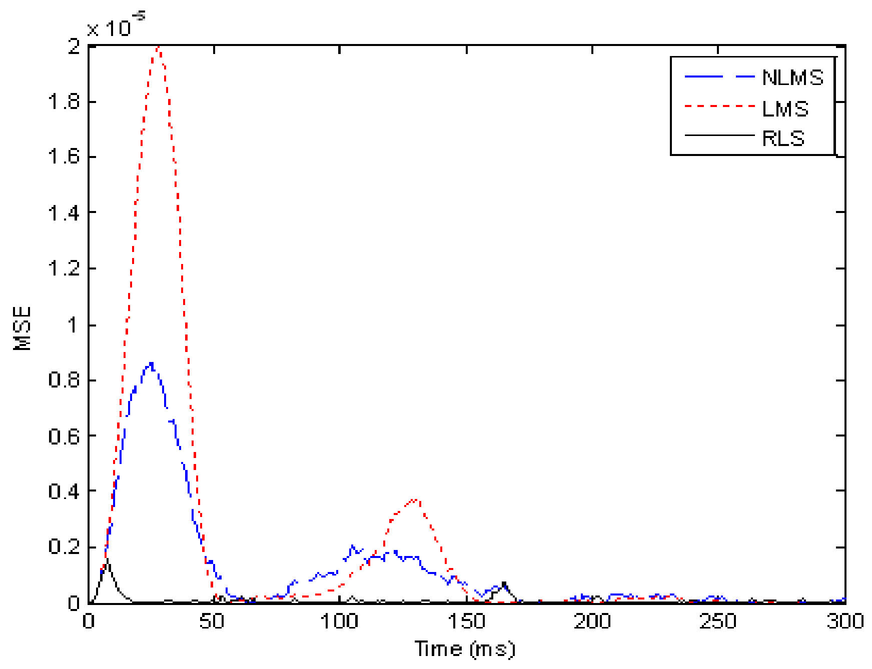


Figure 5.5 MSE of LMS, NLMS and RLS algorithms: N = 20 taps, (μ = 0.03) for LMS and NLMS, (β = 1) for RLS.

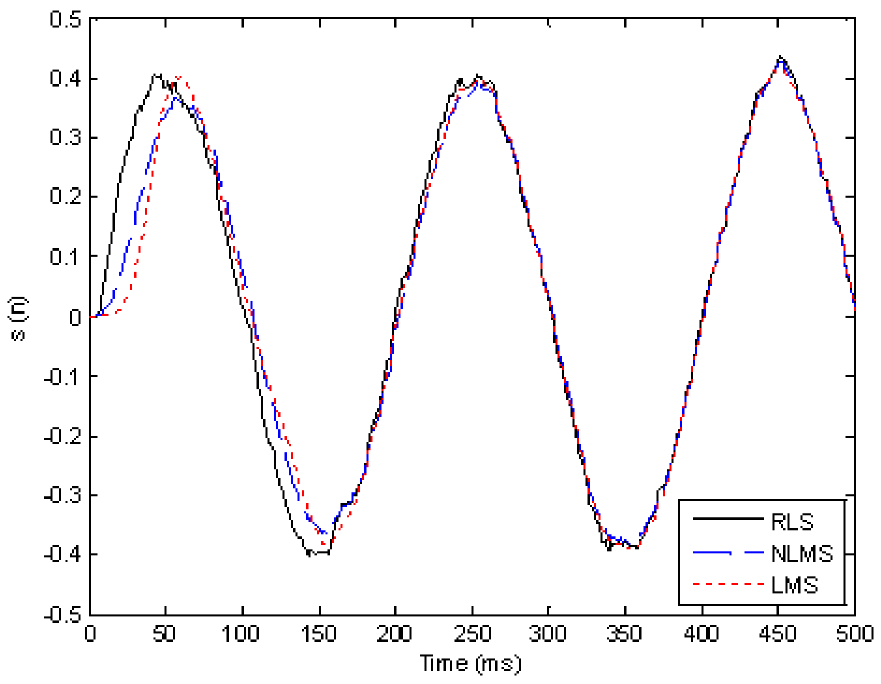


Figure 5.6 Recovered sinusoid by LMS, NLMS and RLS algorithms: N = 20 taps, (μ = 0.03) for LMS and NLMS, (β = 1) for RLS.

In the second part, the filter length is assumed to be $N = 40$ taps for all algorithms, $\mu = 0.03$ for LMS and NLMS algorithms and for the RLS algorithm $\beta = 1$. Fig. 5.7 shows the MSE of all algorithms. From the figure we again notice that the RLS algorithm provides the fastest convergence rate and lowest MSE compared to the other algorithms. The NLMS algorithm converges to the same MSE as that of the RLS algorithm with slightly lower convergence rate. The LMS algorithm converges to the same MSE of the other algorithms. However, it has the lowest convergence rate. Fig. 5.8 confirms what has been shown in Fig. 5.7 by showing the recovered sinusoid by all algorithms.

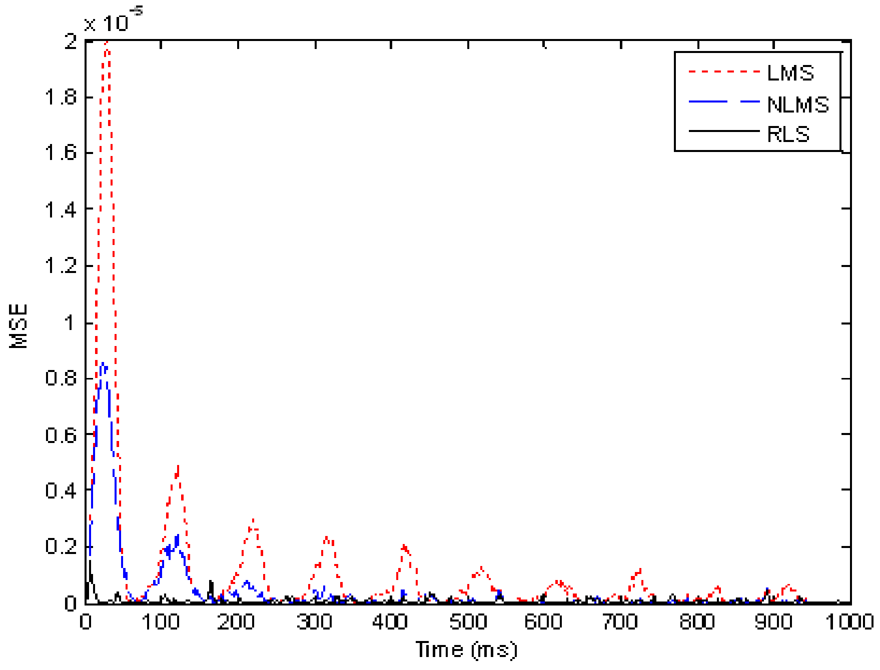


Figure 5.7 MSE of LMS, NLMS and RLS algorithms: $N = 40$ taps, ($\mu = 0.03$) for LMS and NLMS, ($\beta = 1$) for RLS.

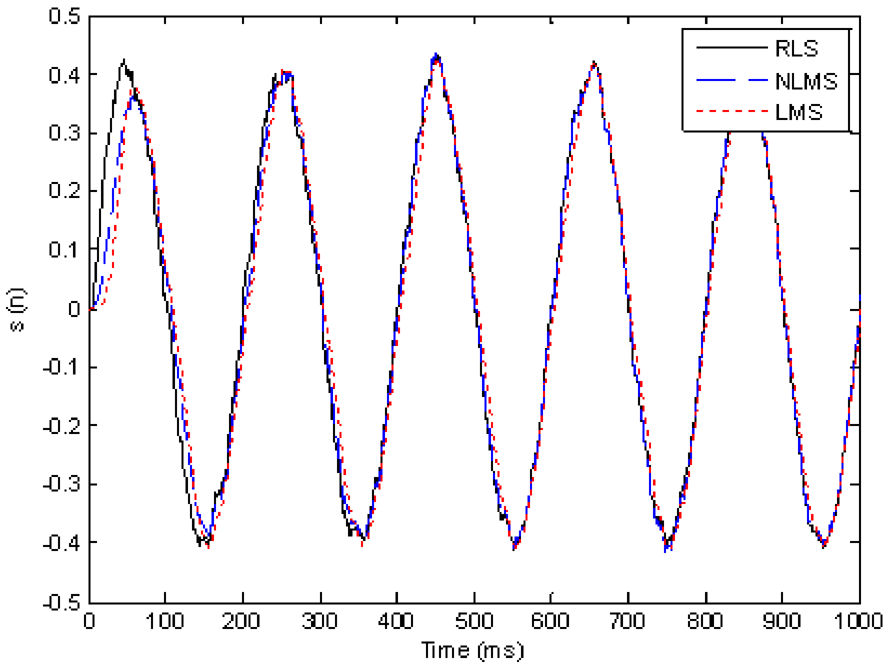


Figure 5.8 Recovered sinusoid by LMS, NLMS and RLS algorithms: N = 40 taps, (μ = 0.03) for LMS and NLMS, (β = 1) for RLS.

In the third part, the filter length is assumed to be N = 20 taps for all algorithms, μ = 0.003 for LMS and NLMS algorithms and for the RLS algorithm β = 1. Fig. 5.9 shows the MSE of all algorithms. From the figure we again notice that the RLS algorithm provides the fastest convergence rate and lowest MSE compared to the other algorithms. The LMS algorithm converges to the same MSE as that of the RLS algorithm with slightly lower convergence rate. Now, even though the NLMS algorithm converges to the same MSE of the other algorithms, it has the lowest convergence rate. This is because of the very low step-size when it is divided by the power of the input vector. Fig. 5.10 confirms what has been shown in Fig. 5.9 by showing the recovered sinusoid by all algorithms.

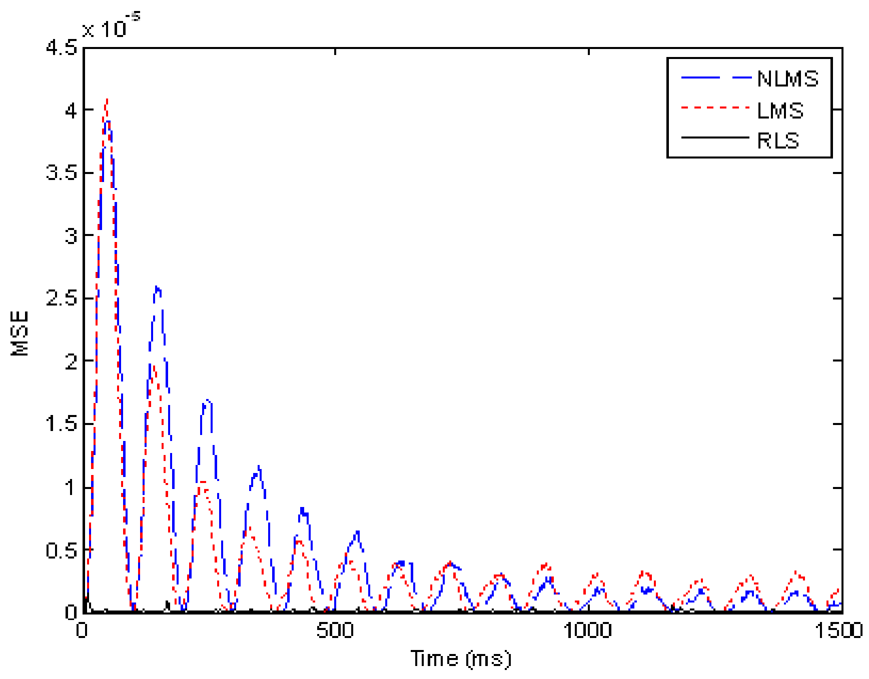


Figure 5.9 MSE of LMS, NLMS and RLS algorithms: N = 20 taps, (μ = 0.003) for LMS and NLMS, (β = 1) for RLS.

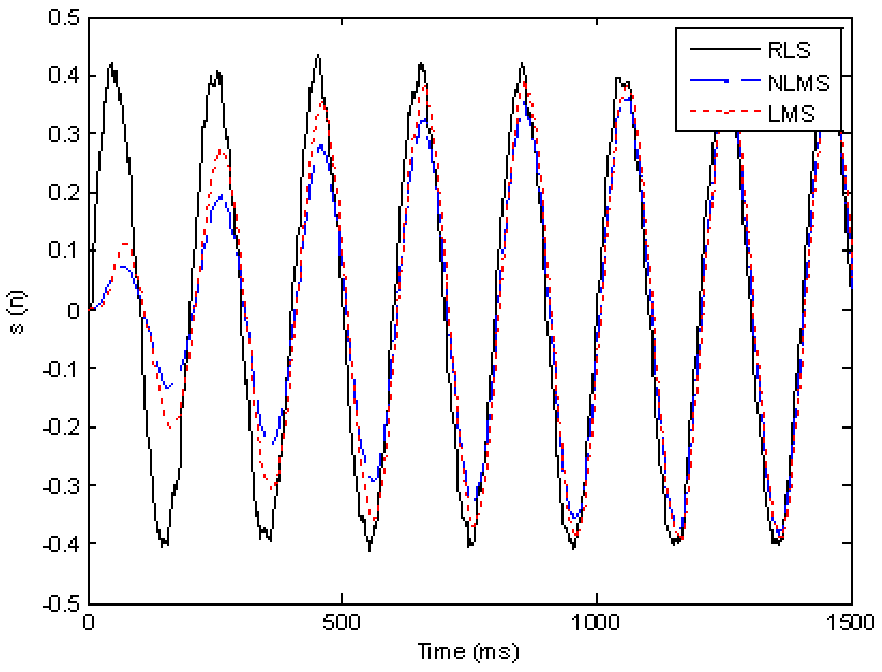


Figure 5.10 Recovered sinusoid by LMS, NLMS and RLS algorithms: N = 20 taps, (μ = 0.003) for LMS and NLMS, (β = 1) for RLS.

In the last part, the filter length is assumed to be N = 40 taps for all algorithms, μ = 0.003 for LMS and NLMS algorithms and for the RLS algorithm β = 1. Fig. 5.11 shows the MSE of all algorithms. From the figure we again notice that the RLS algorithm provides the fastest convergence rate and lowest MSE compared to the other algorithms. The LMS algorithm converges to the same MSE as that of the RLS algorithm with slightly lower convergence rate. Now, even though the NLMS algorithm converges to the same MSE of the other algorithms, it has the lowest convergence rate. This is because of the very low step-size when it is divided by the power of the input vector. Fig. 5.12 confirms what has been shown in Fig. 5.11 by showing the recovered sinusoid by all algorithms. Also, it is noted that increasing the filter length for the aforementioned experiments provides no gain; hence 20 taps filter length could be enough for recovering such signal.

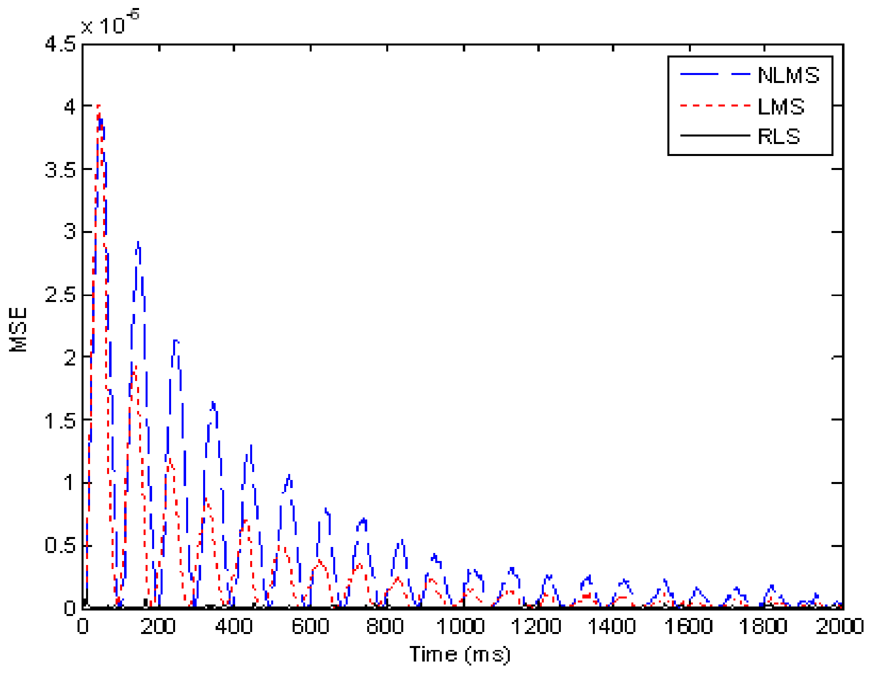


Figure 5.11 MSE of LMS, NLMS and RLS algorithms: N = 40 taps, (μ = 0.003) for LMS and NLMS, (β = 1) for RLS.

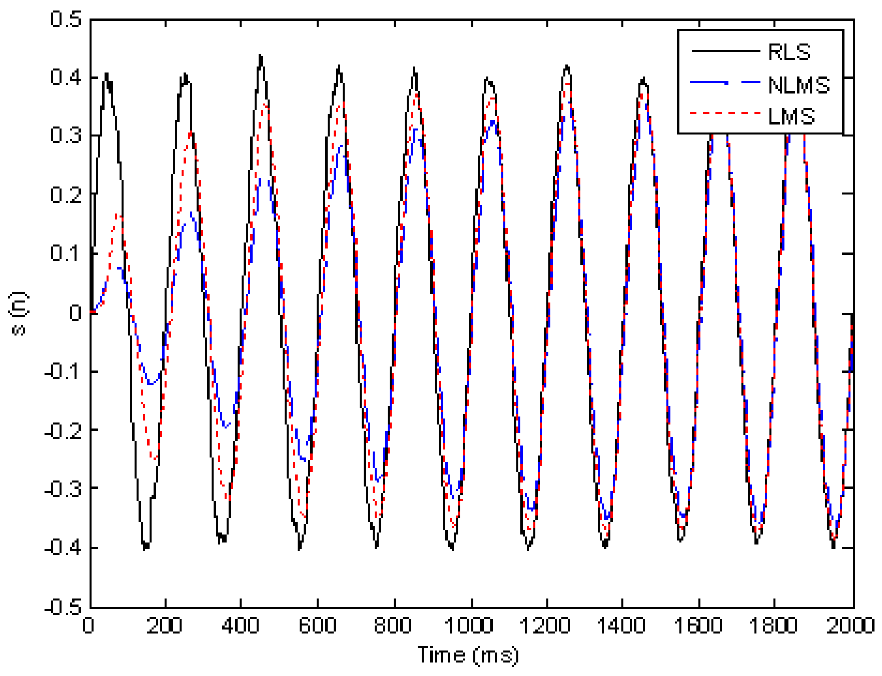


Figure 5.12 Recovered sinusoid by LMS, NLMS and RLS algorithms: N = 40 taps, (μ = 0.003) for LMS and NLMS, (β = 1) for RLS.

5.3 Music Input Signal

5.3.1 Simulation Results

In this experiment, a music signal is used. To be comparable with the results shown by the oscilloscope, a portion of the signal is shown in Fig. 5.13. Then, an AWGN (shown in Fig. 5.15) with zero mean and variance $\sigma^2 = 0.03$ is added to the input signal. The resulting signal is assumed to be the received signal and is shown in Fig. 5.17.

The filter length is assumed to be N = 10 taps for all algorithms, μ = 0.05 for LMS and NLMS algorithms and for the RLS algorithm β = 1. Figs. 5.19, 5.21 and 5.23 show the output errors of LMS NLMS and RLS algorithms, respectively. From the figure we see that the RLS algorithm provides smallest estimated output error compared to the other algorithms. Figs. 5.25, 5.27 and 5.29 show the recovered music signal by LMS NLMS and RLS algorithms, respectively.

5.3.2 Experimental Results

In this part, the experiments of section 5.3.1 are uploaded to the DSP card and the results are shown on the oscilloscope. Fig. 5.14 shows the input music signal. Then, an AWGN (shown in Fig. 5.16) with zero mean and variance $\sigma^2 = 0.03$ is added to the input signal. The resulting signal is assumed to be the received signal and is shown in Fig. 5.18.

The filter length of the filter specifies how accurately a given system can be modeled by the adaptive filter. In addition, the filter length affects the convergence rate, by increasing or decreasing computation time, it can affect the stability of the system, at certain step sizes, and it affects the MSE.

In this experiment the filter length is assumed to be $N = 10$ taps for all algorithms, $\mu = 0.05$ for LMS and NLMS algorithms and for the RLS algorithm $\beta = 1$. Figs. 5.20, 5.22 and 5.24 show the output errors of the LMS, NLMS and RLS algorithms, respectively. From the figures we see that the RLS algorithm provides smallest estimated output error compared to the other algorithms. And Figs. 5.26, 5.28 and 5.30 show the recovered music signal by the LMS, NLMS and RLS algorithms, respectively. The results seen by the oscilloscope are in compatible with the simulation results.

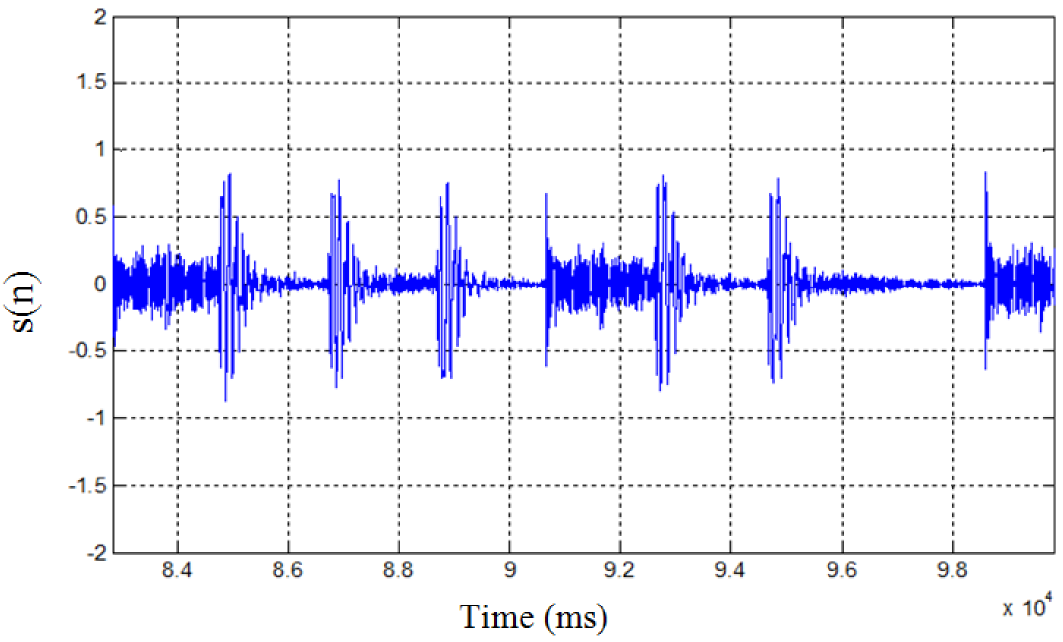


Figure 5.13 Input music signal ($s(n)$) for simulation results.

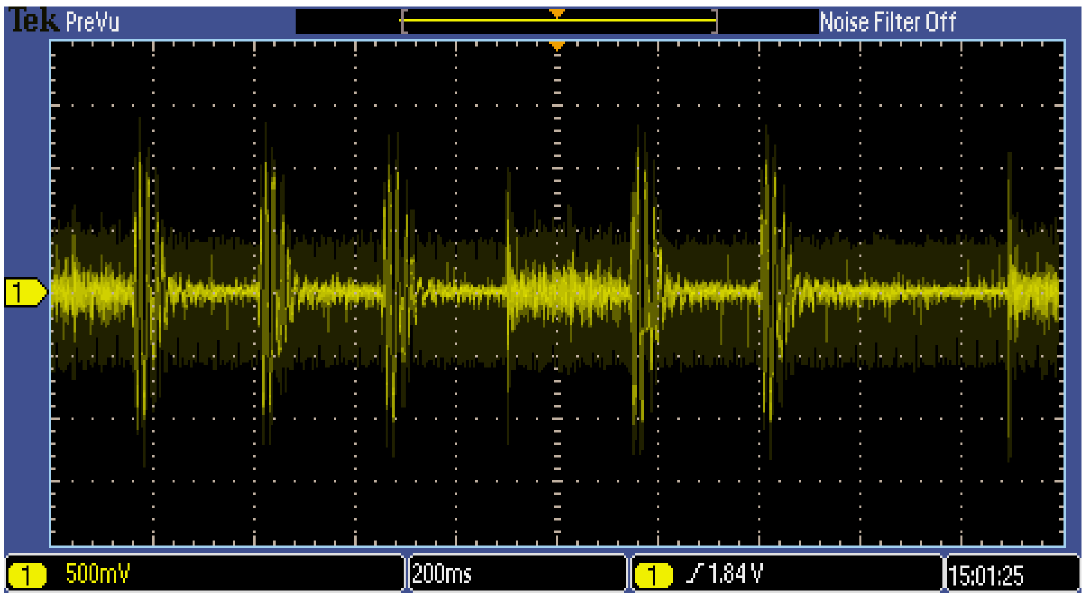


Figure 5.14 Input music signal ($s(n)$), as seen on oscilloscope.

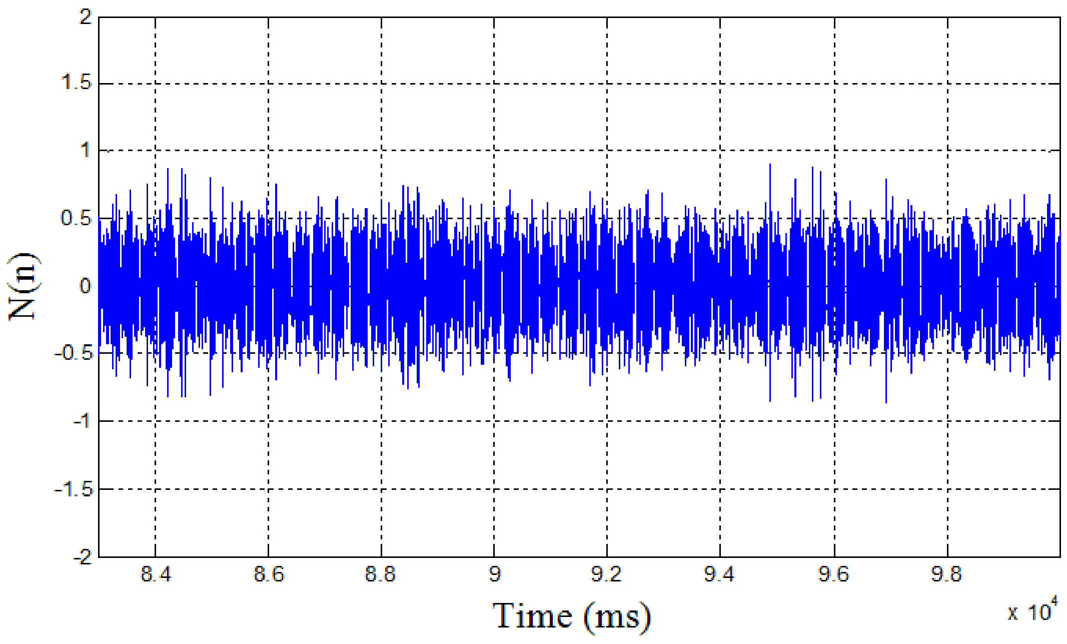


Figure 5.15 Added noise ($N(n)$) for simulation results.

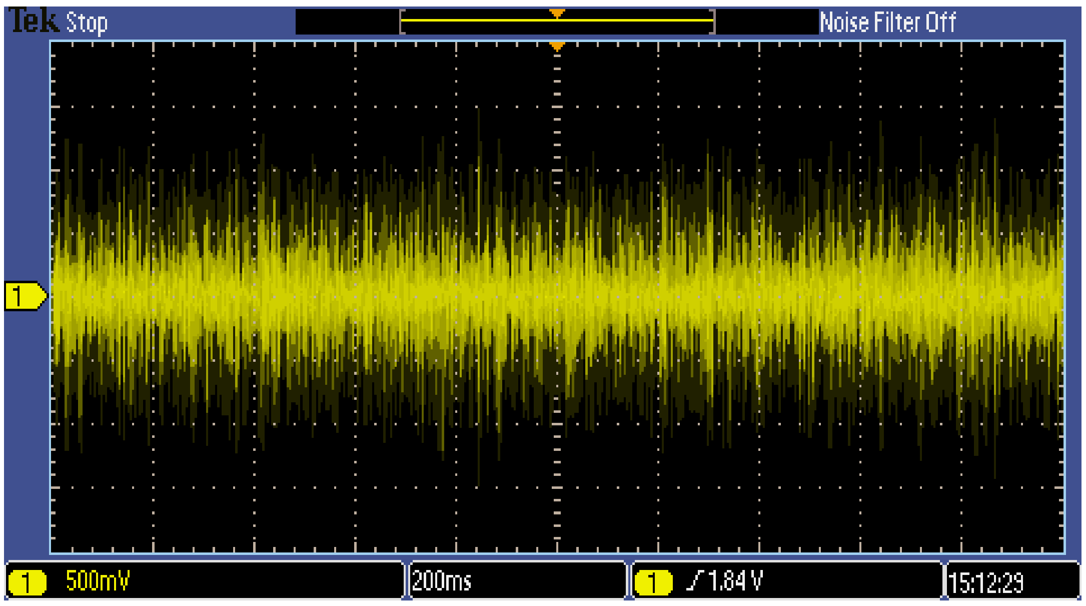


Figure 5.16 Added noise ($N(n)$), as seen on oscilloscope.

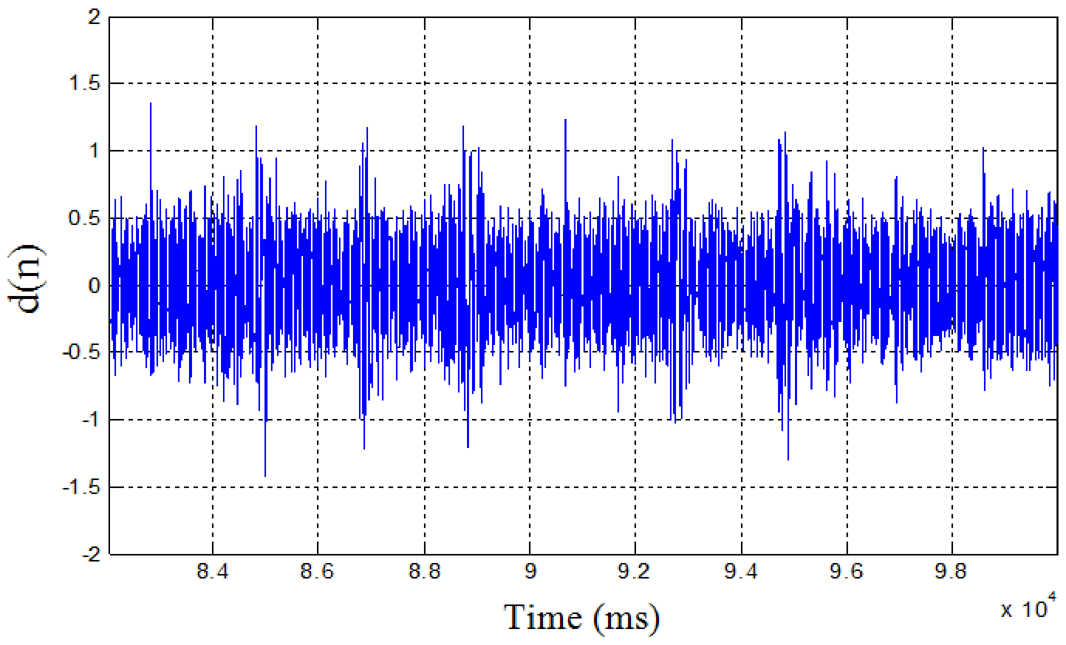


Figure 5.17 Input signal with noise ($d(n)$) for simulation results.

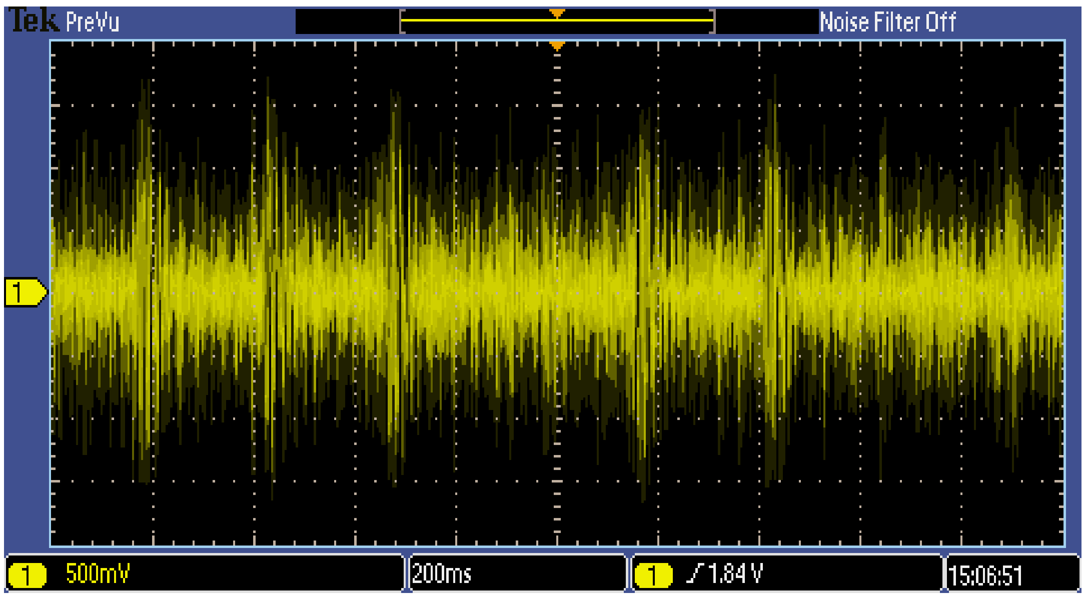


Figure 5.18 Input music signal with noise ($d(n)$), as seen on oscilloscope.

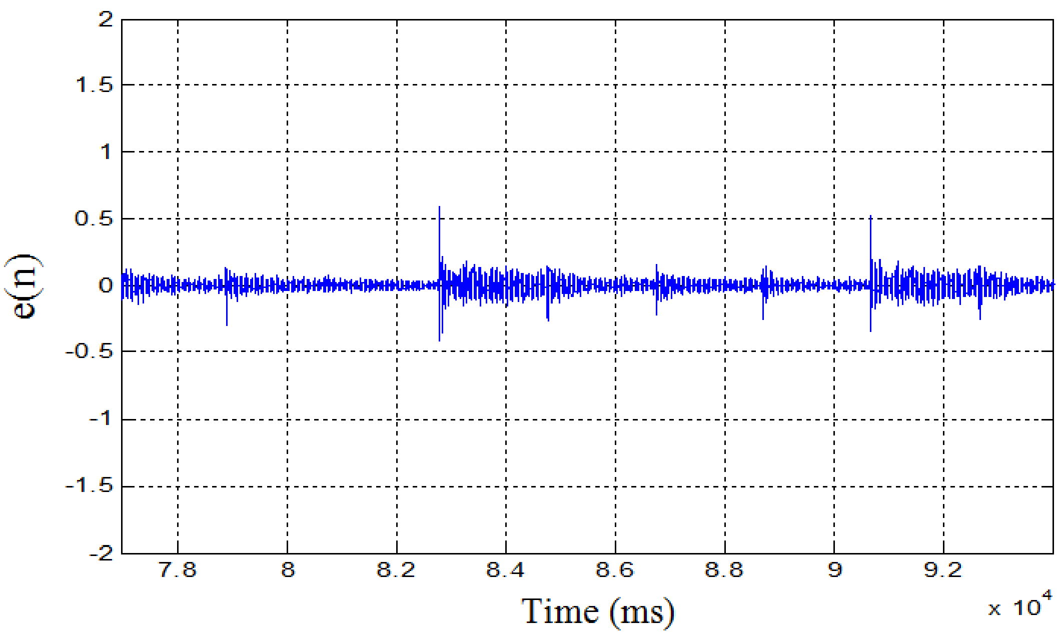


Figure 5.19 Output error (*e*(*n*)) of LMS algorithm: *N* = 10 taps, (*μ* = 0.05).

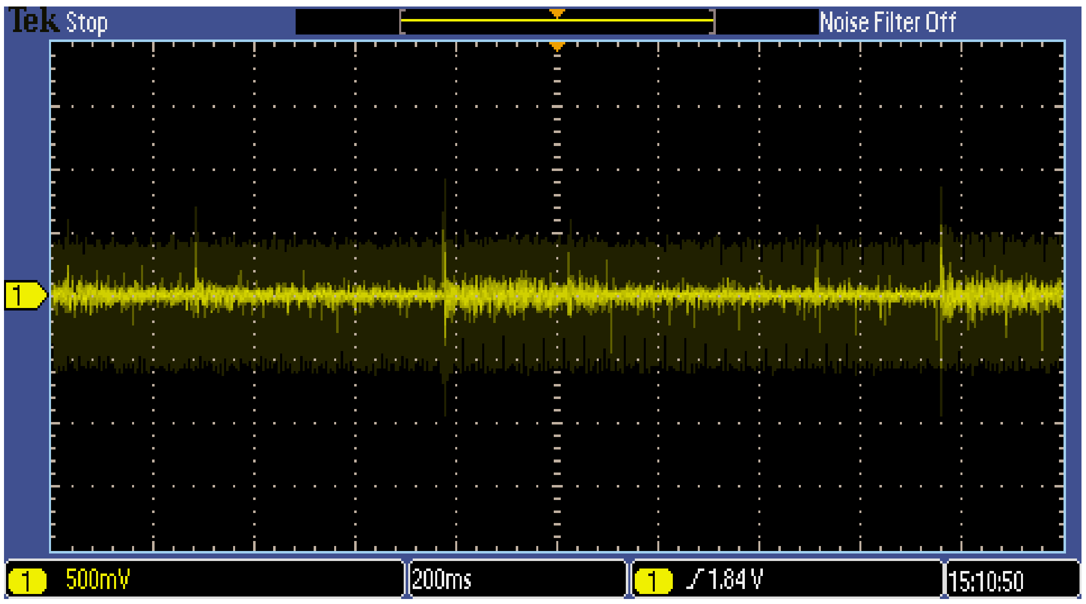


Figure 5.20 Output error (*e*(*n*)) of LMS algorithm on oscilloscope: *N* = 10 taps, (*μ* = 0.05).

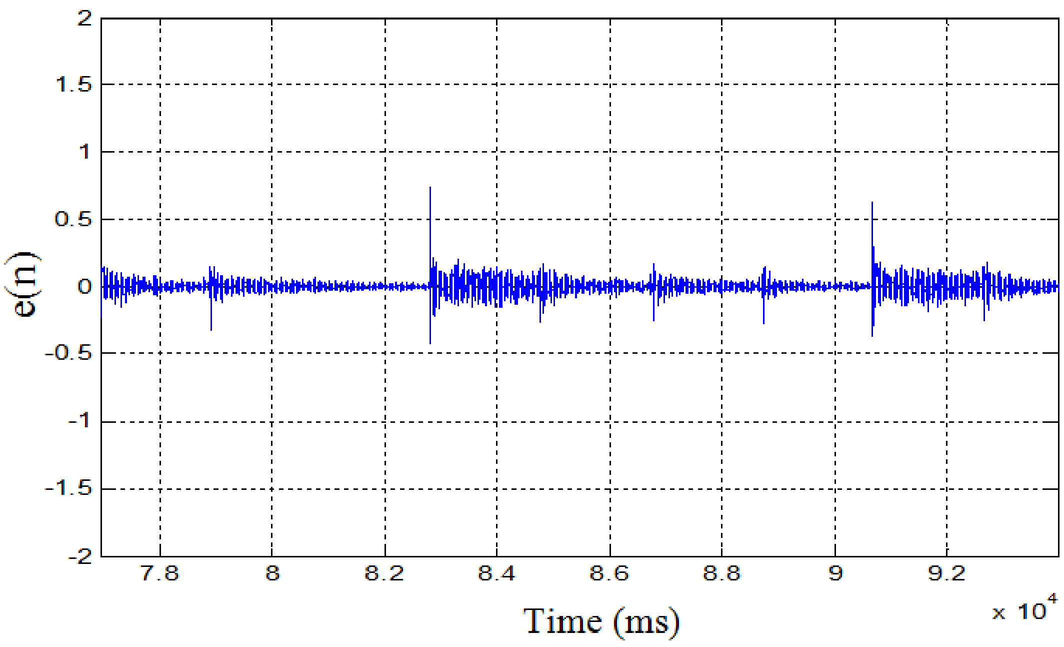


Figure 5.21 Output error ($e(n)$) of NLMS algorithm: N = 10 taps, (μ = 0.05).

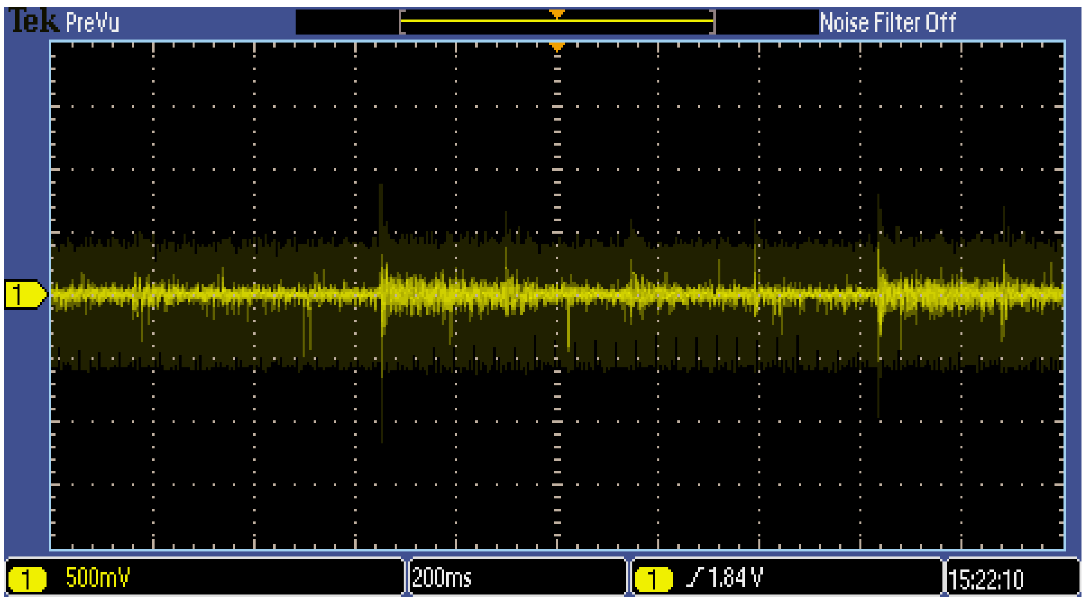


Figure 5.22 Output error ($e(n)$) of NLMS algorithm on oscilloscope: N = 10 taps, (μ = 0.05).

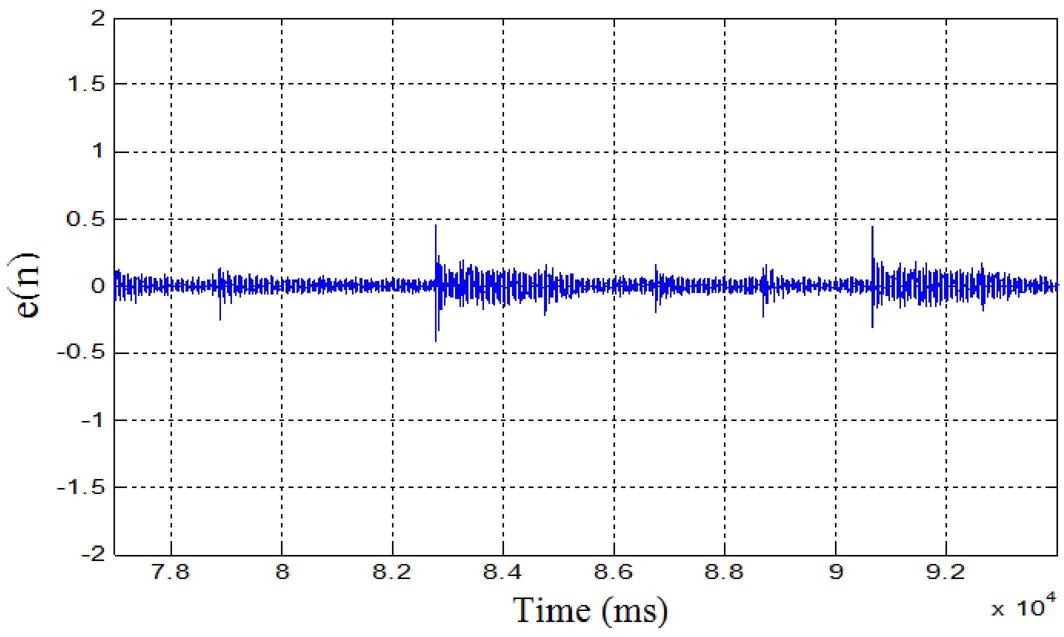


Figure 5.23 Output error (e(n)) of RLS algorithm: N = 10 taps, (β = 1).

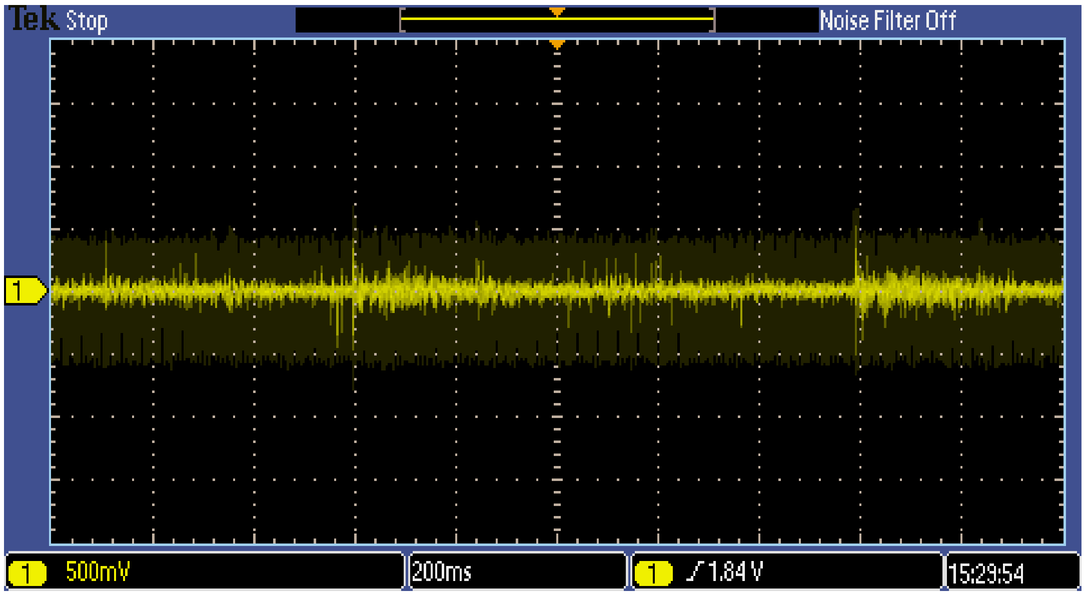


Figure 5.24 Output error (e(n)) of RLS algorithm on oscilloscope: N = 10 taps, (β = 1).

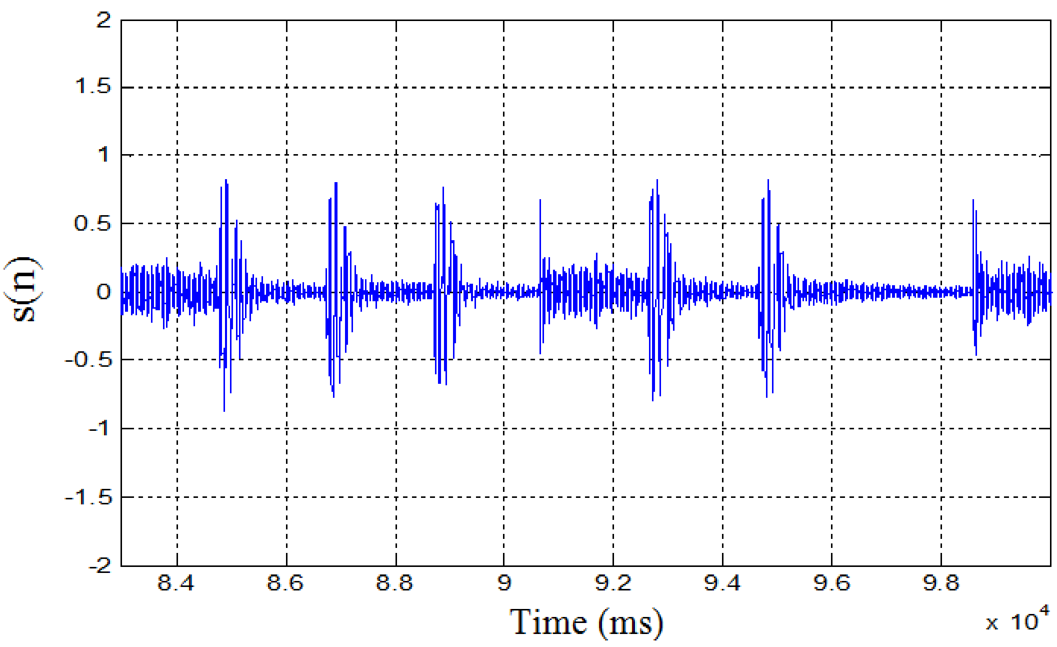


Figure 5.25 Recovered signal by LMS algorithm: N = 10 taps, (μ = 0.05).

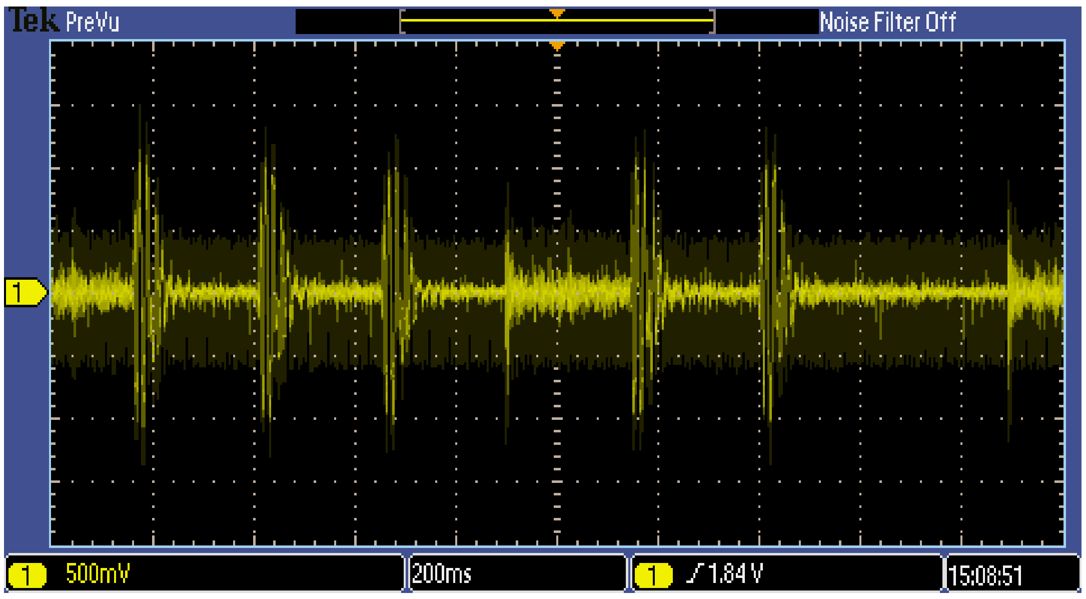


Figure 5.26 Recovered signal by LMS algorithm on oscilloscope: N = 10 taps, (μ = 0.05).

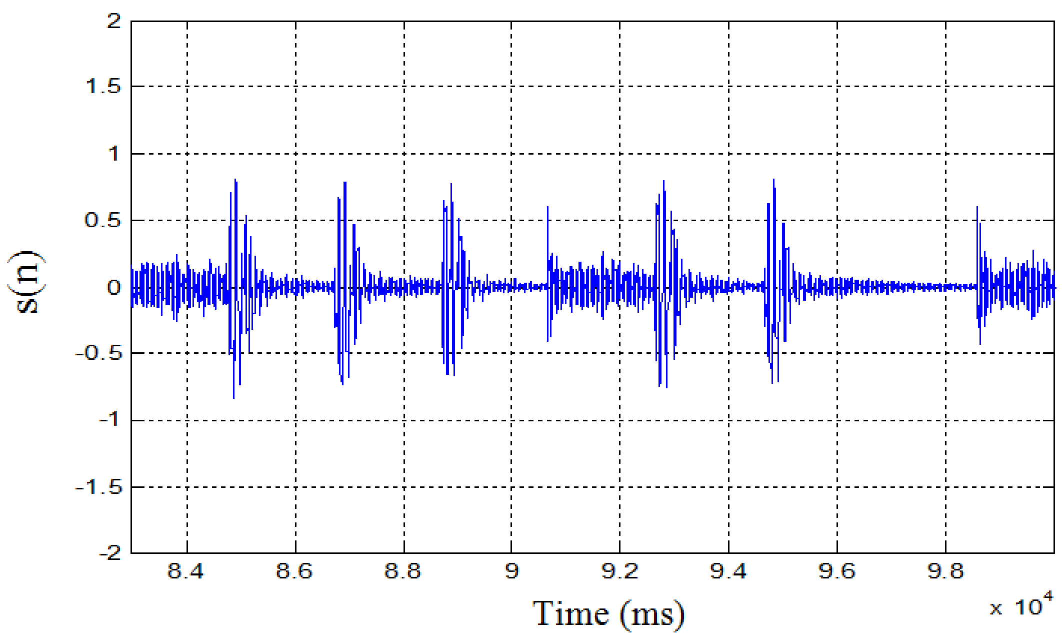


Figure 5.27 Recovered signal by NLMS algorithm: *N* = 10 taps, (μ = 0.05).

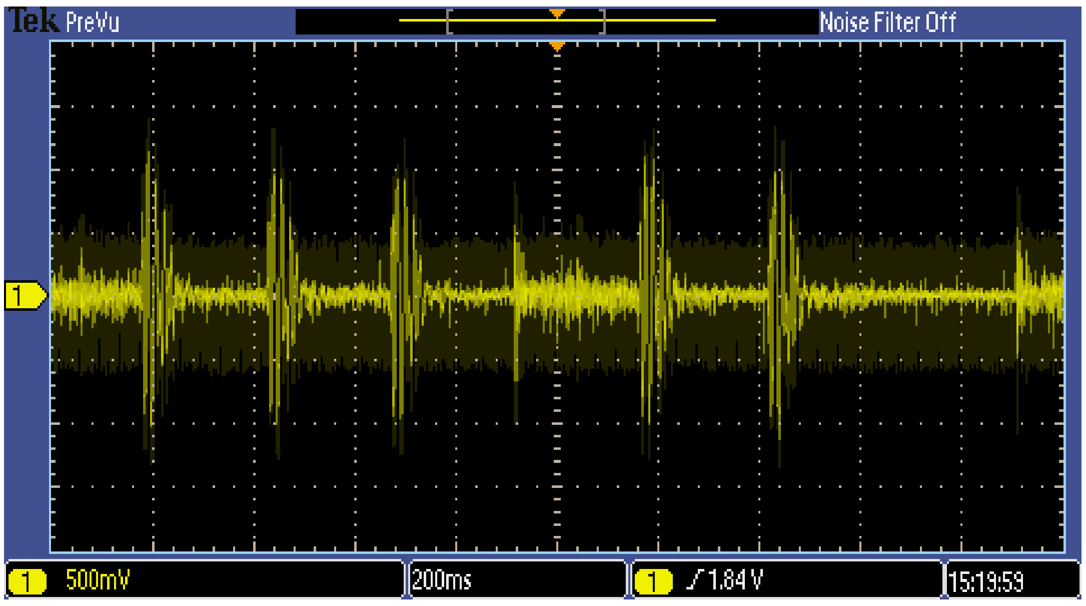


Figure 5.28 Recovered signal by NLMS algorithm on oscilloscope: *N* = 10 taps, (μ = 0.05).

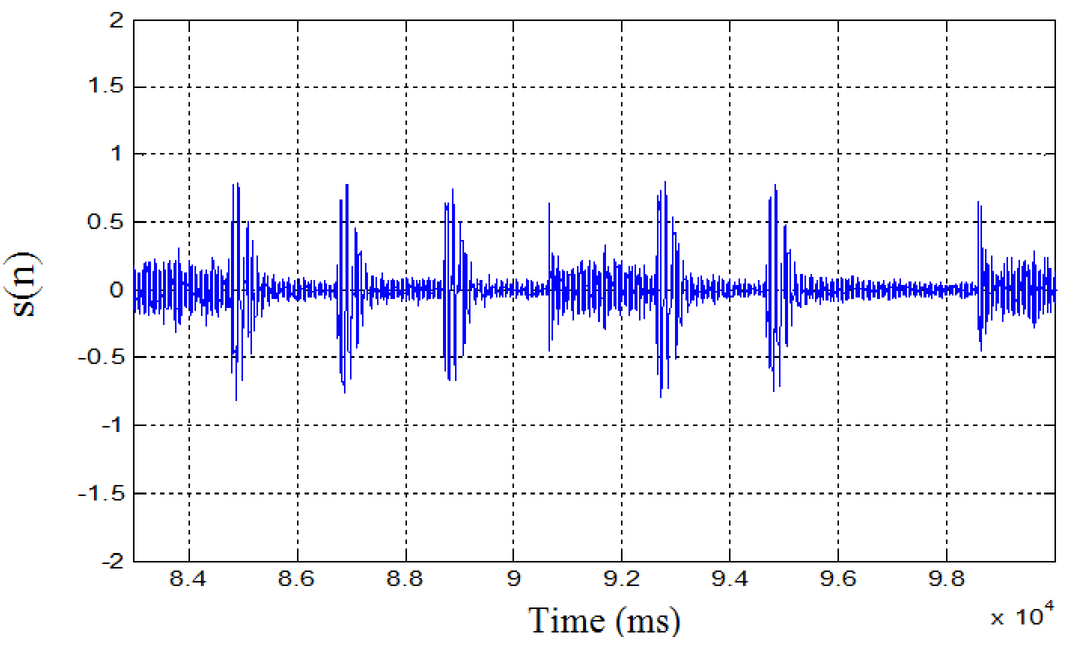


Figure 5.29 Recovered signal by RLS algorithm: N = 10 taps, (β = 1).

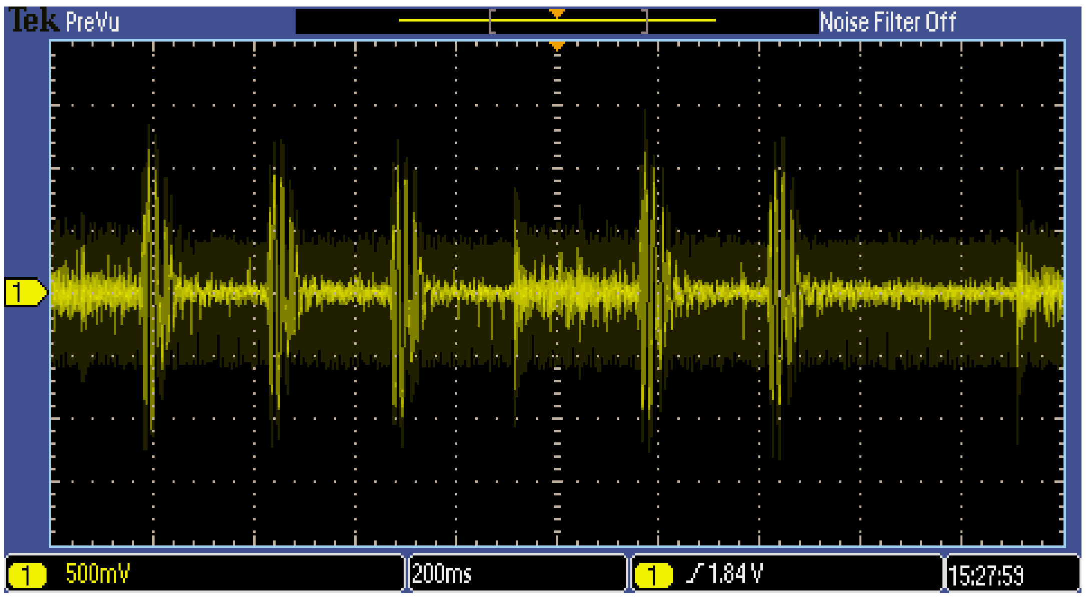


Figure 5.30 Recovered signal by RLS algorithm on oscilloscope: N = 10 taps, (β = 1).

Chapter 6

CONCLUSIONS

6.1 Conclusions

In this thesis, a performance comparison between the LMS, NLMS and RLS algorithms under different step-sizes and filter lengths has been investigated using the SIMULINK package. The simulations have been done with different input signals (mainly a sinusoid and general music signals were used). Simulations have shown that the RLS algorithm outperforms the other algorithms; of course, this high performance is with a trade-off with the high computational complexity of the RLS algorithm. NLMS algorithm provides very good performance (better than the LMS and close to that of the RLS) with almost the same computational complexity of that of the LMS algorithm.

A real time implementation is also carried out by a TI TMS320C6416T DSP by transferring the SIMULINK schemes to the DSP board which let it work alone in real time independent of MATLAB. Furthermore, the performance of the aforementioned algorithms as seen on oscilloscope was compatible to what has been investigated by the software.

6.2 Future Work

As a future work, different algorithms' performances can be compared by the investigated ones and their simulation results can be compared by their hardware ones. Also, one of the main disadvantages of the RLS algorithm is its stability if the autocorrelation matrix is singular; hence a way to make the RLS more stable could be investigated.

REFERENCES

[1] S. Haykin, *Adaptive Filter* Theory, 4th ed., Prentice Hall, Upper Saddle River, NJ, 2002.

[2] B. F. Boroujeny, Adaptive Filters: Theory and Applications, JohnWiley, BaffinsLane, Chichester, 1998.

[3] B.Widrow and E.Walach, Adaptive Inverse Control, Reissue Edition: A Signal Processing Approach, Wiley-IEEE Press, 2007.

[4] B. Widrow and S. D. Stearns, Adaptive Signal Processing, Prentice Hall, Eaglewood Cliffs, N.J., 1985.

[5] D. Starer and A. Nehorai, “Newton algorithms for conditional and unconditional maximum likelihood estimation of the parameters of exponential signals in noise,” IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. 40, No. 6, 1992, pp. 1528-1534.

[6] M. G. Bellanger, Adaptive Digital Filters, 2nd ed., Marcel Dekker, New York, 2001.

[7] C. V. Sinn and J. Gotze, “Comparative study of techniques to compute FIR filter weights in adaptive channel equalization,” IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP03), vol. 6, April 2003, pp. 217-220.

[8] X. Guan, X. Chen; and G. Wu, “QX-LMS adaptive FIR filters for system identification,” 2nd International Congress on Image and Signal Processing (CISP2009), 2009, pp. 1-5.

[9] B. Allen and M. Ghavami, Adaptive Array Systems: Fundamentals and Applications, John Wiley & Sons Ltd, West Sussex, England, 2005.

[10] J. I. Nagumo and A. Noda, “A Learning method for system identification,” IEEE Transactions on Automation and Control, vol. AC-12, 1967, pp. 282-287.

[11] A. E. Albert and L. S. Gardner, Stochastic Approximation and Nonlinear Regression, MIT Press, Cambridge, MA, 1967.

[12] R. R. Bitmead and B. D. O. Anderson, “Lyapunov techniques for the exponential stability of linear difference equations with random coefficients,” IEEE Transactions on Automation and Control, vol. AC-25, 1980, pp. 782-787.

[13] R. R. Bitmead and B. D. O. Anderson, “Performance of adaptive estimation algorithms in independent random environments,” IEEE Transactions on Automation and Control, vol. AC-25, 1980, pp. 788-794.

[14] G. Tummarello, F. Nardini and F. Piazza, “Step size control in NLMS acoustic echo cancellation using a neural network approach,” International Symposium on Circuits and Systems, Vol. 5, May 2003, pp. 705-708.

[15] R. Hastings-James, R. and M. W. Sage, “Recursive generalised-least-squares procedure for online identification of process parameters,” Proceedings of the Institution of Electrical Engineers, vol. 116, no. 12, 1969, pp. 2057-2062.

[16] V. Panuska, “An adaptive recursive-least-squares identification algorithm,” 8th IEEE Symposium on Adaptive Processes and Decision and Control, vol. 8, part 1, 1969, pp. 65-69.

[17] G. O. Glentis, K. Berberidis and S. Theodoridis, “Efficient least squares adaptive algorithms for FIR transversal filtering,” IEEE Signal Processing Magazine, July 1999, pp. 13-41,

[18] G. O. Glentis, “Efficient fast recursive least-squares adaptive complex filter using real valued arithmetic,” Elsevier Signal Processing, vol. 85, 2005, pp. 1759-1779.

[19] S. Makino and Y. Kaneda, “A New RLS algorithm based on the variation characteristics of a room impulse response,” IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-94), vol. 3, 1994, pp. 373-376.

[20] K. Maouche and D. T. M. Slock, “Fast subsampled-updating stabilized fast transversal filter (FSU SFTF) RLS algorithm for adaptive filtering,” IEEE Transactions on Signal Processing, vol. 48, no. 8, 2000, pp. 2248-2257.

[21] E. M. Eksioglu and A. K. Tanc, “RLS algorithm with convex regularization,” IEEE Signal Processing Letters, vol. 18, no. 8, 2011, pp. 470-473.

[22] J. Wang, “A variable forgetting factor RLS adaptive filtering algorithm,” 3rd IEEE International Symposium on Microwave, Antenna, Propagation and EMC Technologies for Wireless Communications, 2009, pp. 1127-1130.

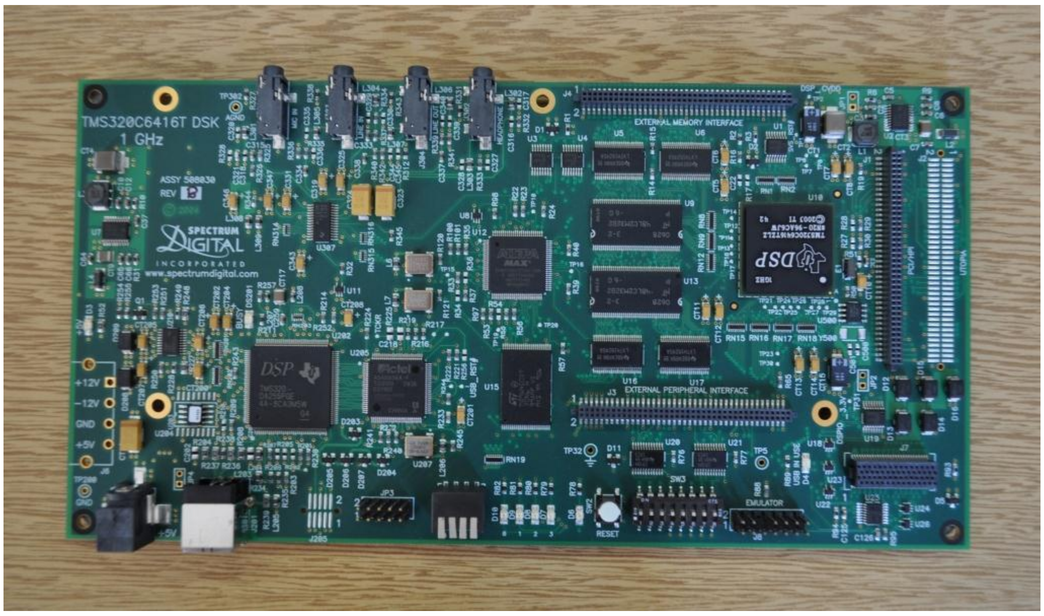
[23] A. H. Sayed, Adaptive Filters, John Wiley & Sons, NJ, 2008.

APPENDICES

Appendix A: Texas Instruments

Steps for using the software and Hardware and how can we connect both of them to make real time implementation noise cancellation:-

For using Texas Instruments DSK6416 (TMS320C6416 (1 GHZ)).



System requirements for DSK 6416 are:-

A) You should install MATLAB R2006a.

B) 500MB of free hard disk space.

C) 128MB of RAM.

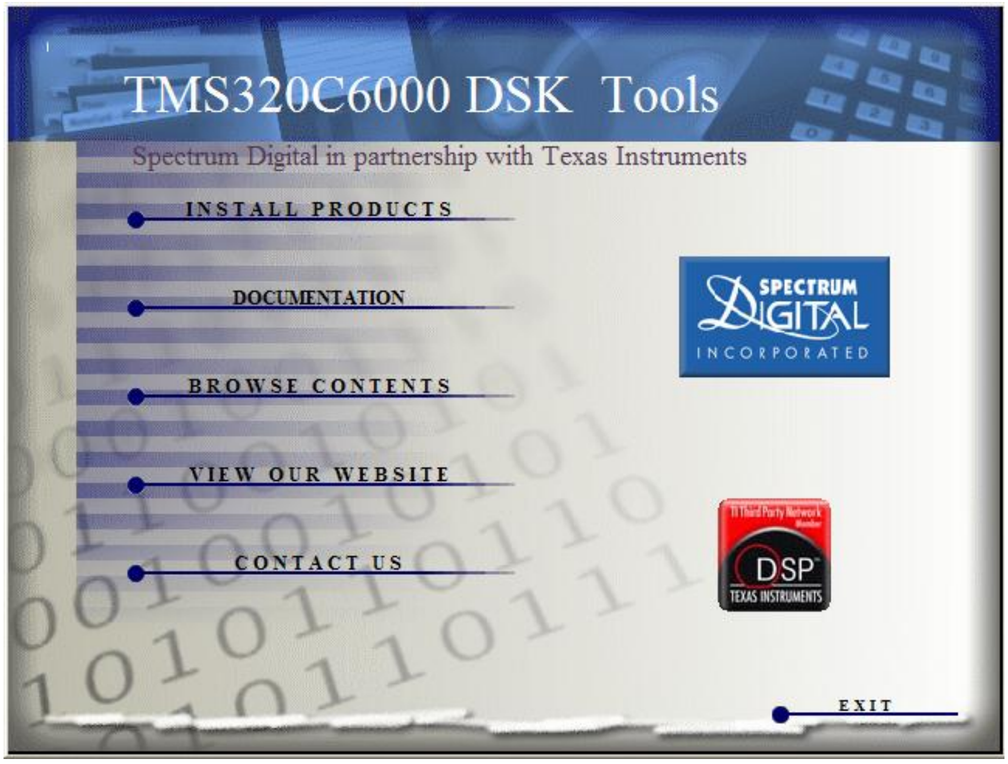
D) 16-bit color display.

E) CD-ROM Drive.

F) Hardware installation using operating systems as windows XP or windows 2000.

PART A: Software Installation

A.1: Insert the code composer studio installation CD into the CD-ROM drive. An install menu (see below) should appear. If it does not, manually run Launch.exe from the install products option from the menu.

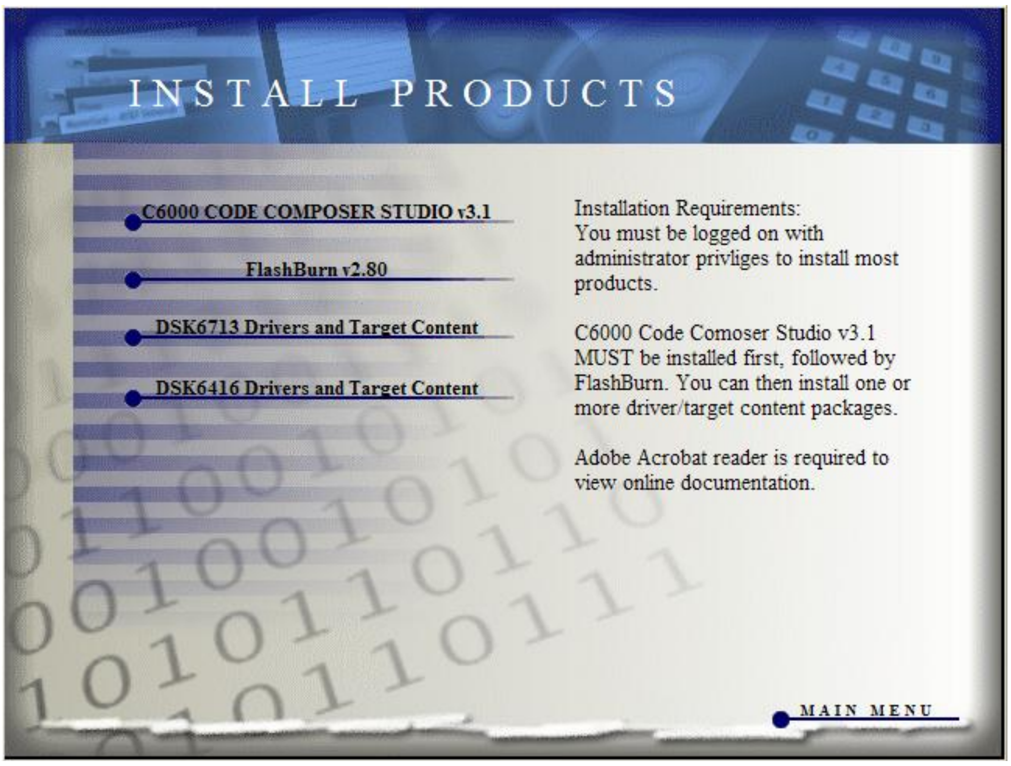


A.2: Install any components you need. To debug with the DSK you must have:-

A) A copy of code composer studio.

B) The target content package for your board.

C) A copy of the Flash Burn plug-in.


INSTALL PRODUCTS
C6000 CODE COMPOSER STUDIO v3.1
FlashBurn v2.80
DSK6713 Drivers and Target Content
DSK6416 Drivers and Target Content
MAIN MENU

Installation Requirements:
You must be logged on with administrator privliges to install most products.

C6000 Code Comoser Studio v3.1 MUST be installed first, followed by FlashBurn. You can then install one or more driver/target content packages.

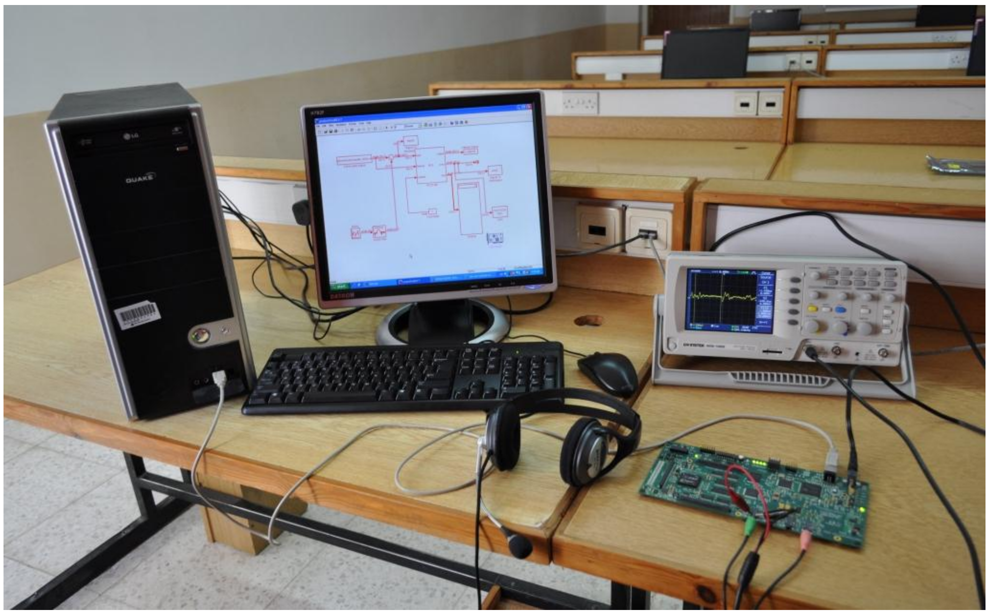
Adobe Acrobat reader is required to view online documentation.

A.3: The installation procedure will create two icons on your desktop:-

6461 DSK CCStudio v3.1

6416 DSK Diagnostics Utility v3.1

PART B: Hardware Connection



A.4: Connect the supplied USB cable to your PC or laptop.

A.5: If you plan to connect a microphone, speaker. Or expansion card these must be plugged in properly before you connect power to the DSK board.

A.6: Connect the included 5V power adapter brick to your AC power cord.

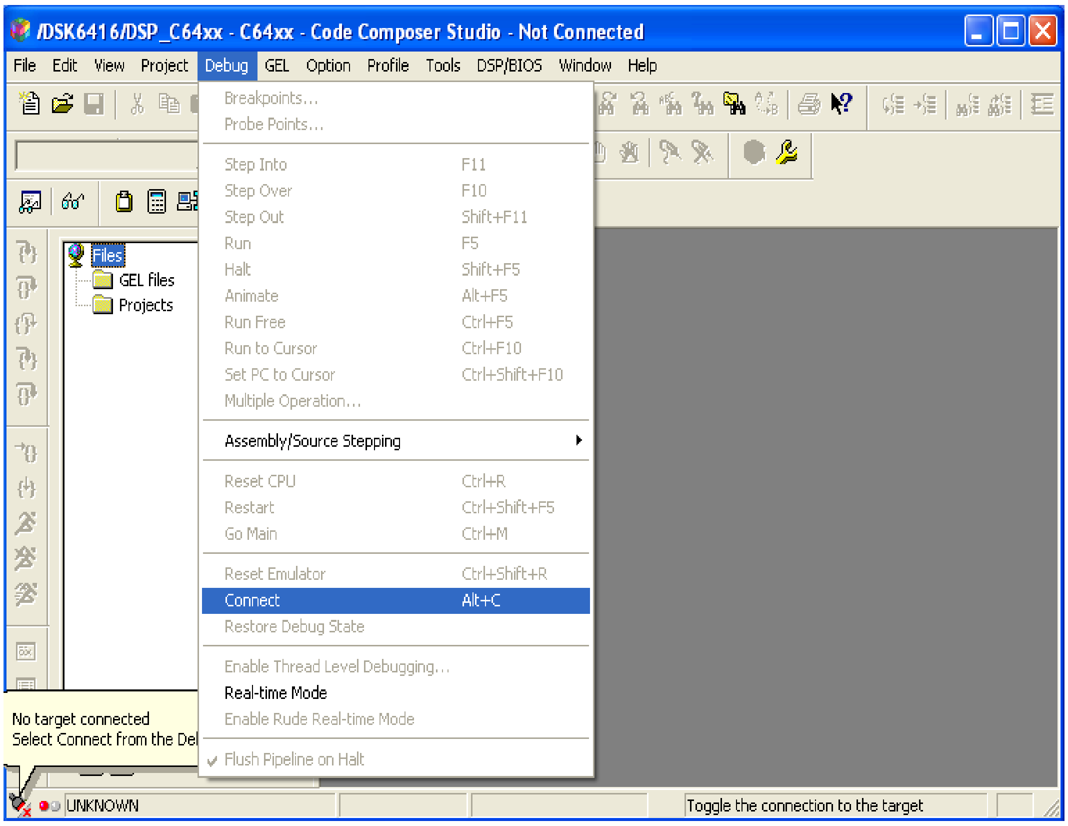
A.7: Apply power to the DSK by connecting the power brick to the 5V input on the DSK.

A.8: When power is applied to the board the power on self-Test (POST) will run.

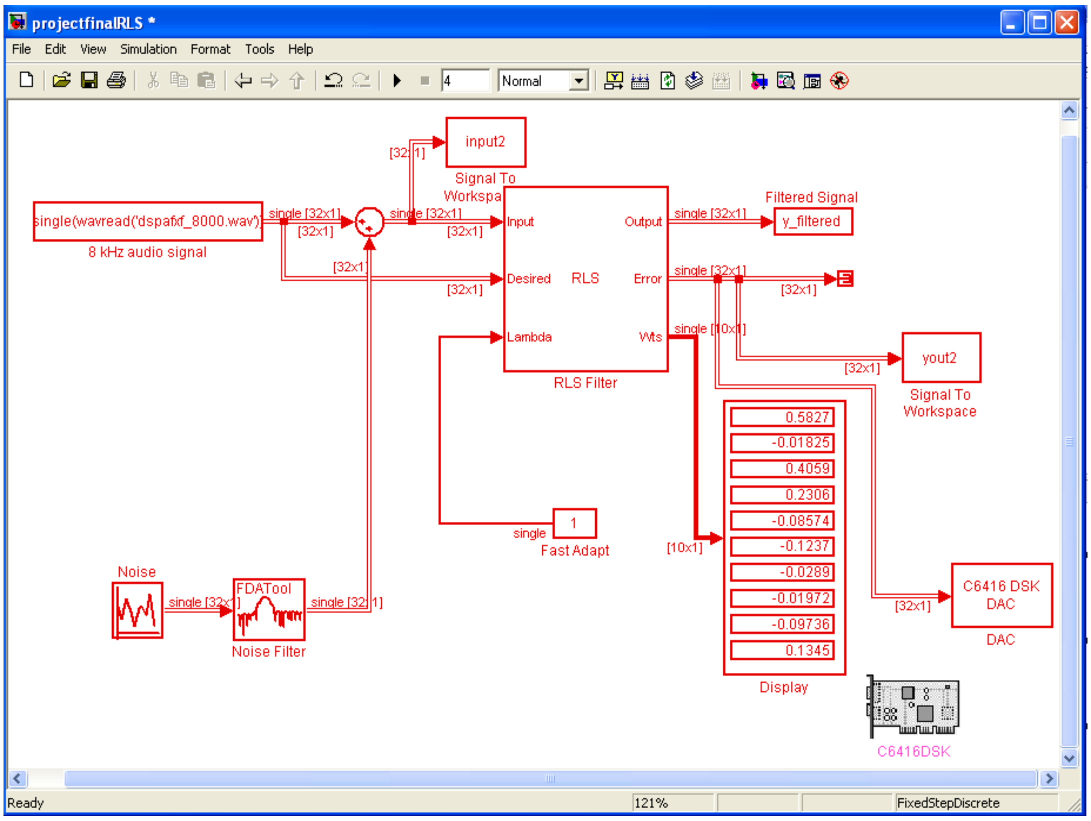
A.9: Make sure you DSK CD-ROM is installed in your CD-ROM drive. Now connect the DSK to your PC using the included USB.

A.10: Test you connection, if you want to test your DSK and USB connection you can launch the C6416 DSK Diagnostic Utility from the icon on your desktop.

A.11: Start Code Composer; double click the 6416 DSK CCStudio icon on your desktop and connect DSK6416, select Debug -> Connect:-

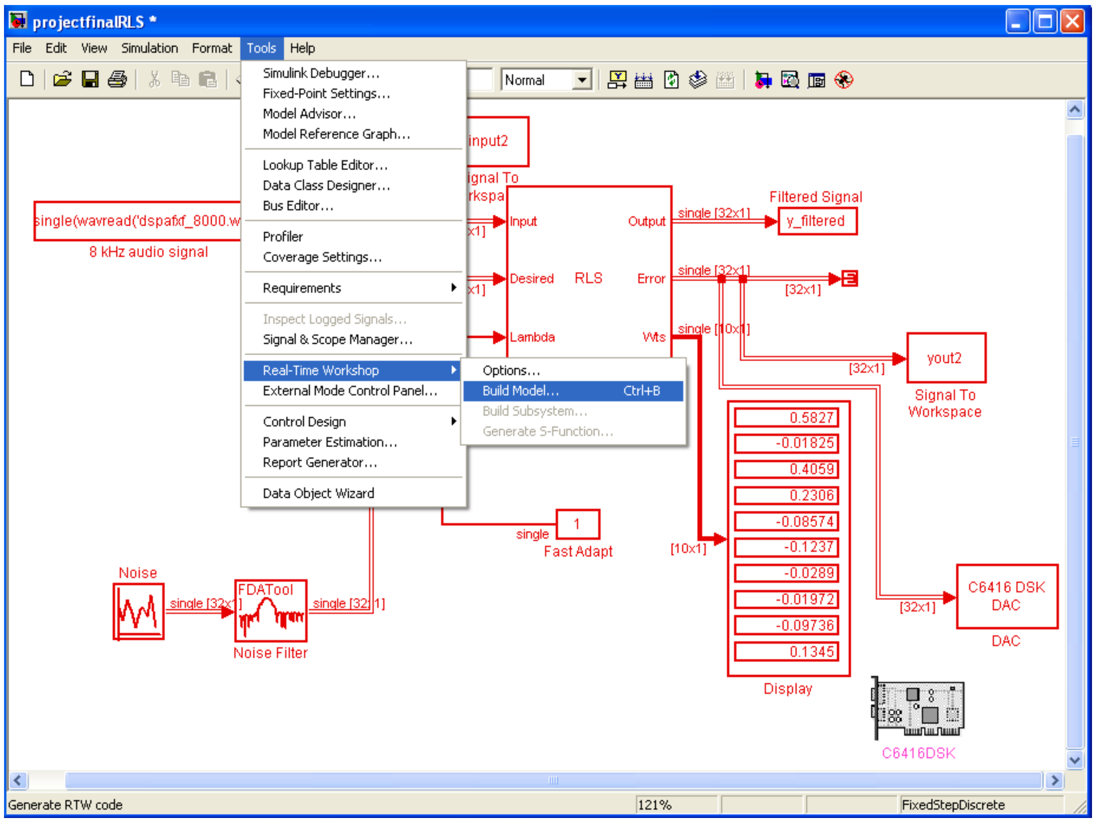


A.12: You should prepare your MATLAB SIMULINK and you should make it as you see from this SIMULINK:-



```
0.5827
-0.01825
0.4059
0.2306
-0.08574
-0.1237
-0.0289
-0.01972
-0.09736
0.1345
```

A.13: When you open the SIMULINK you should connect it to DSP kit, using this way: - Select Tools -> Real-Time Workshop -> Build Model.

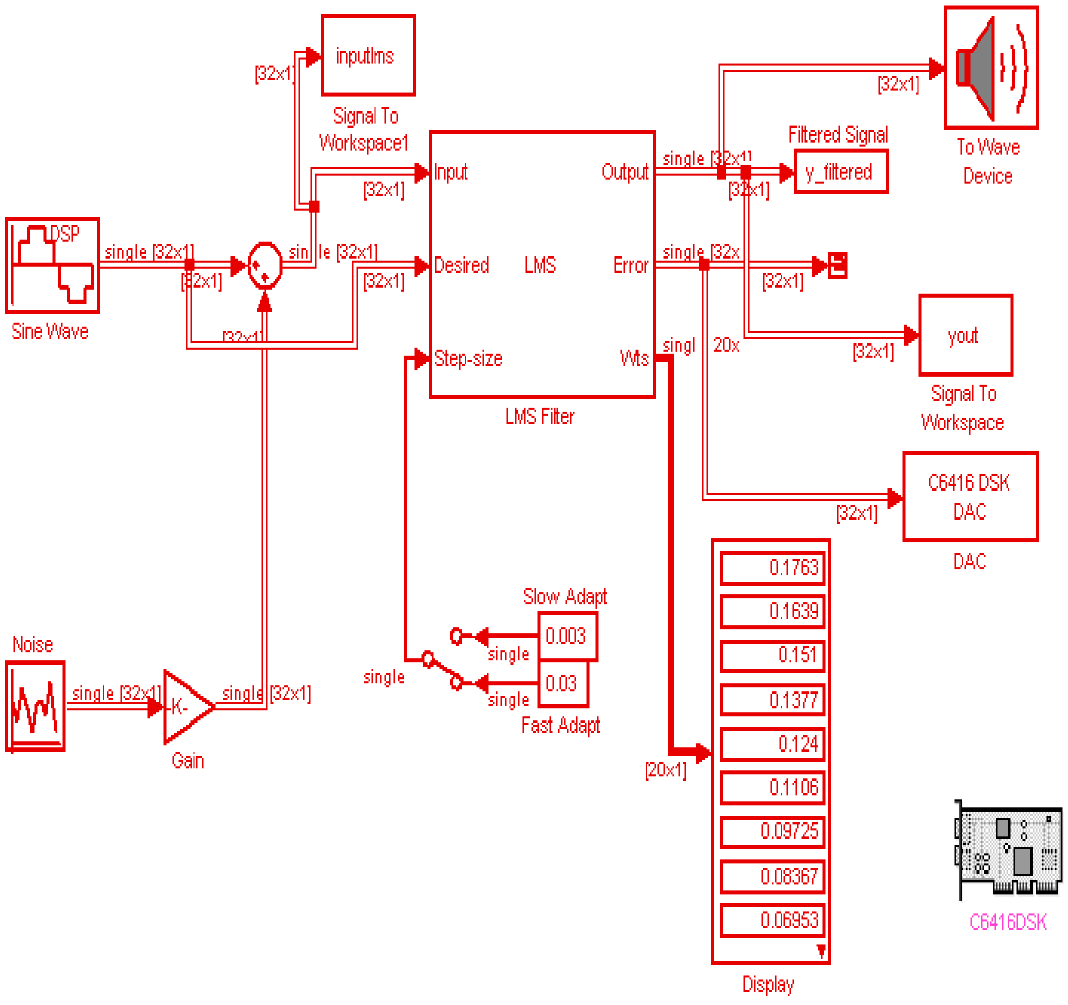


A.14: And then your work will transfer to DSK6416 card, than you can get the output filtering from output for DSK cart.

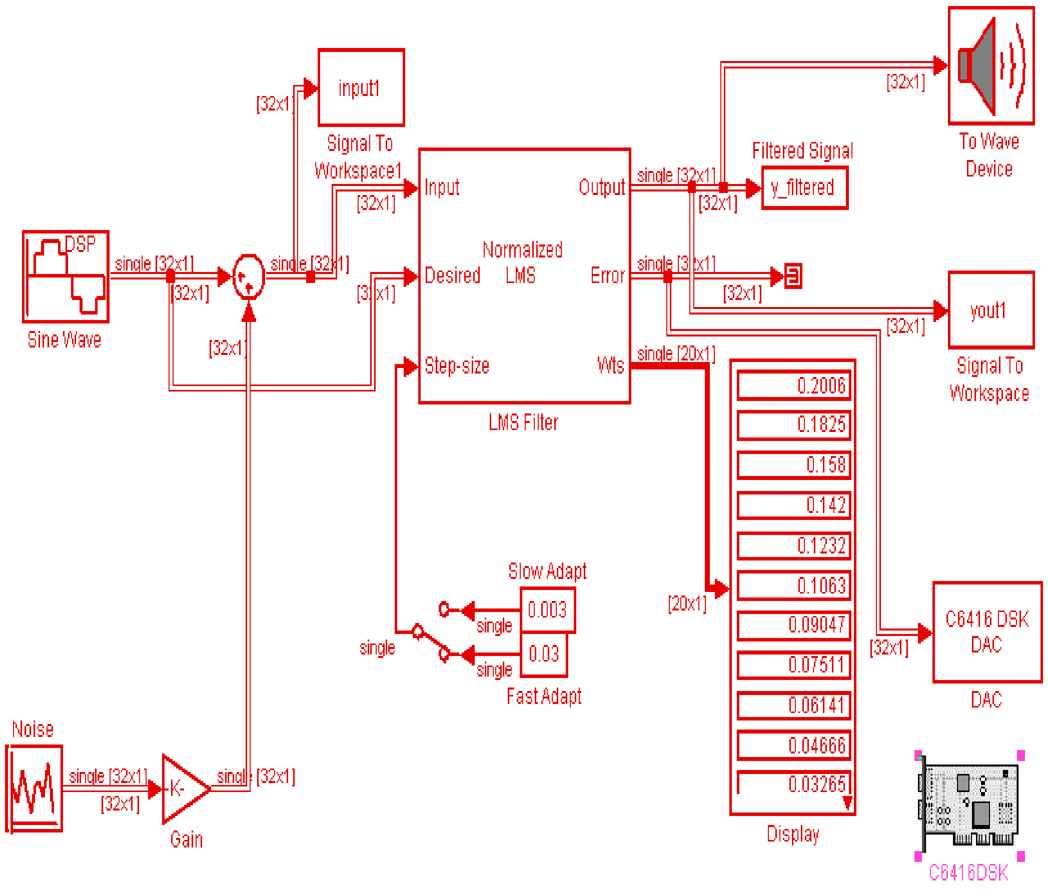
APPENDIX B: Matlab Simulink

PART A: First experiment for sinusoidal noise cancellation:-

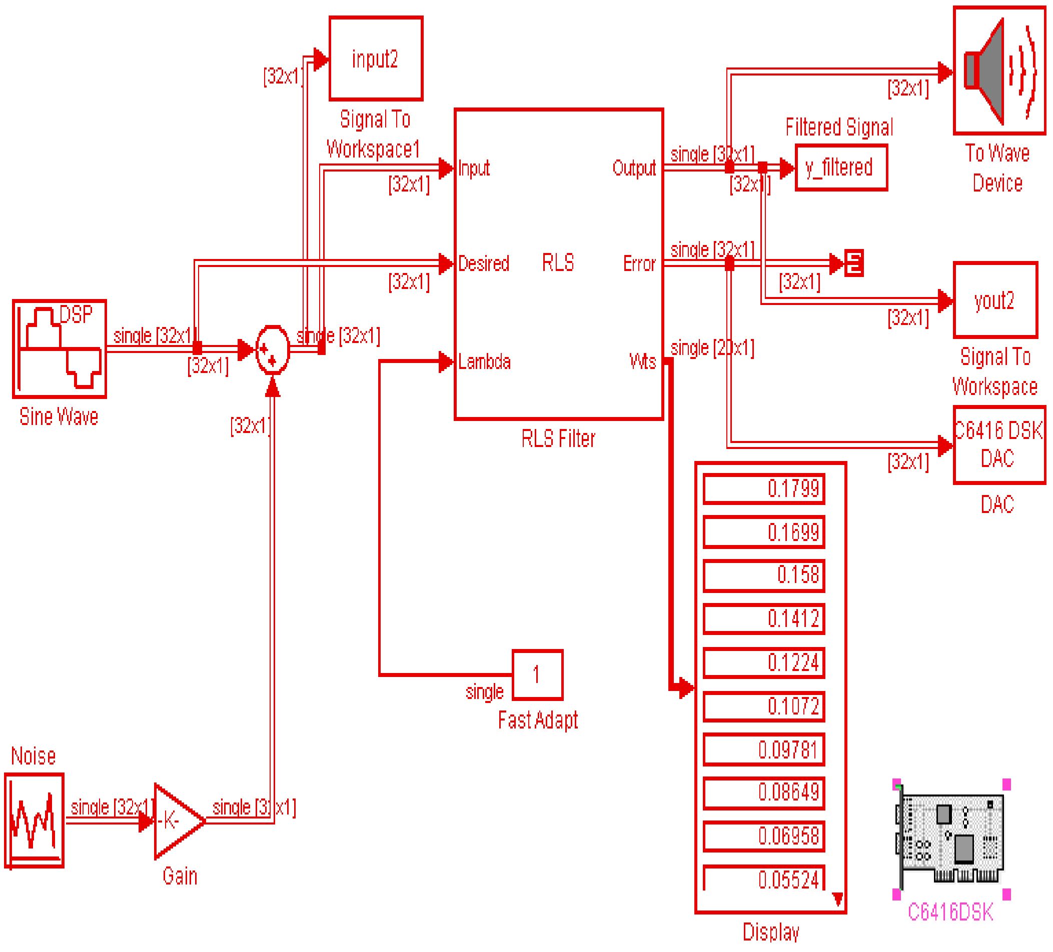
B.1: LMS Algorithm.



B.2: NLMS Algorithm.

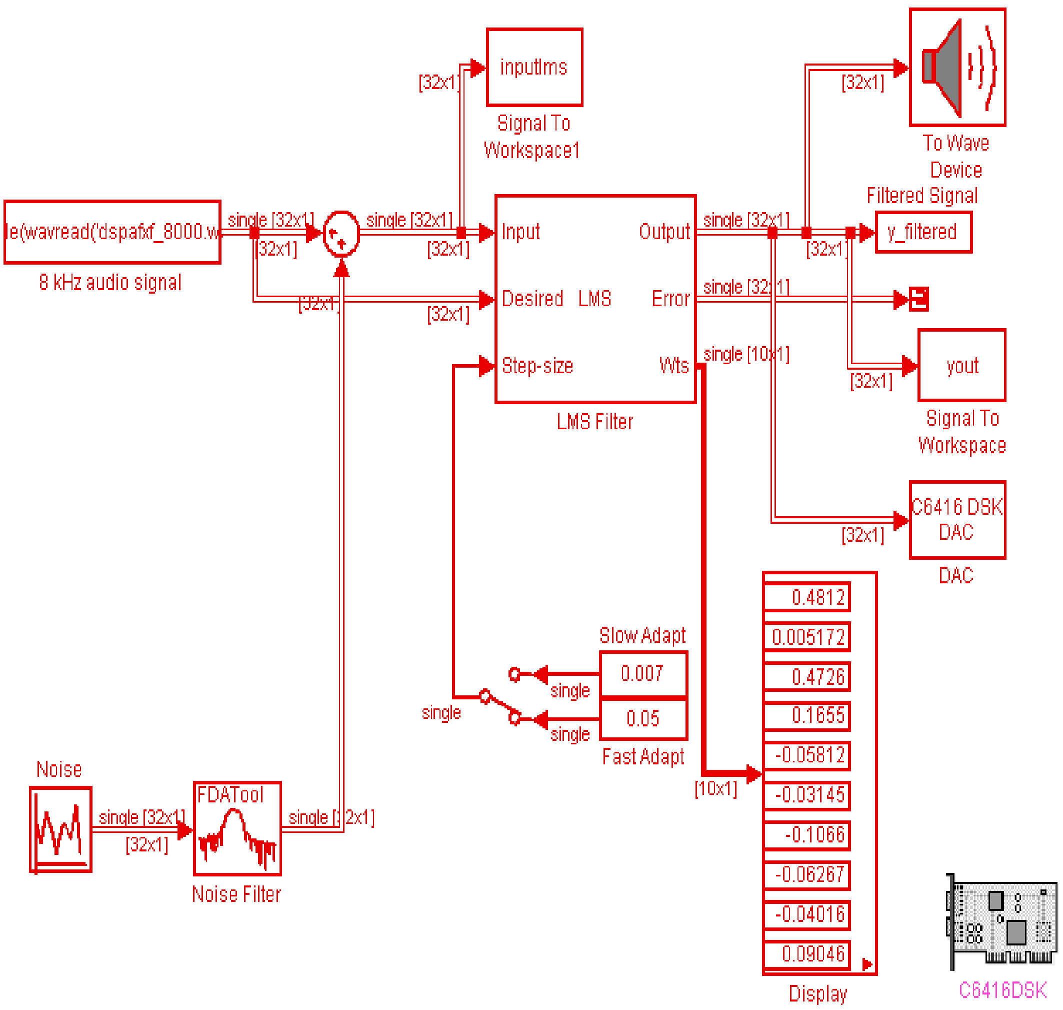


B.3: RLS Algorithm.

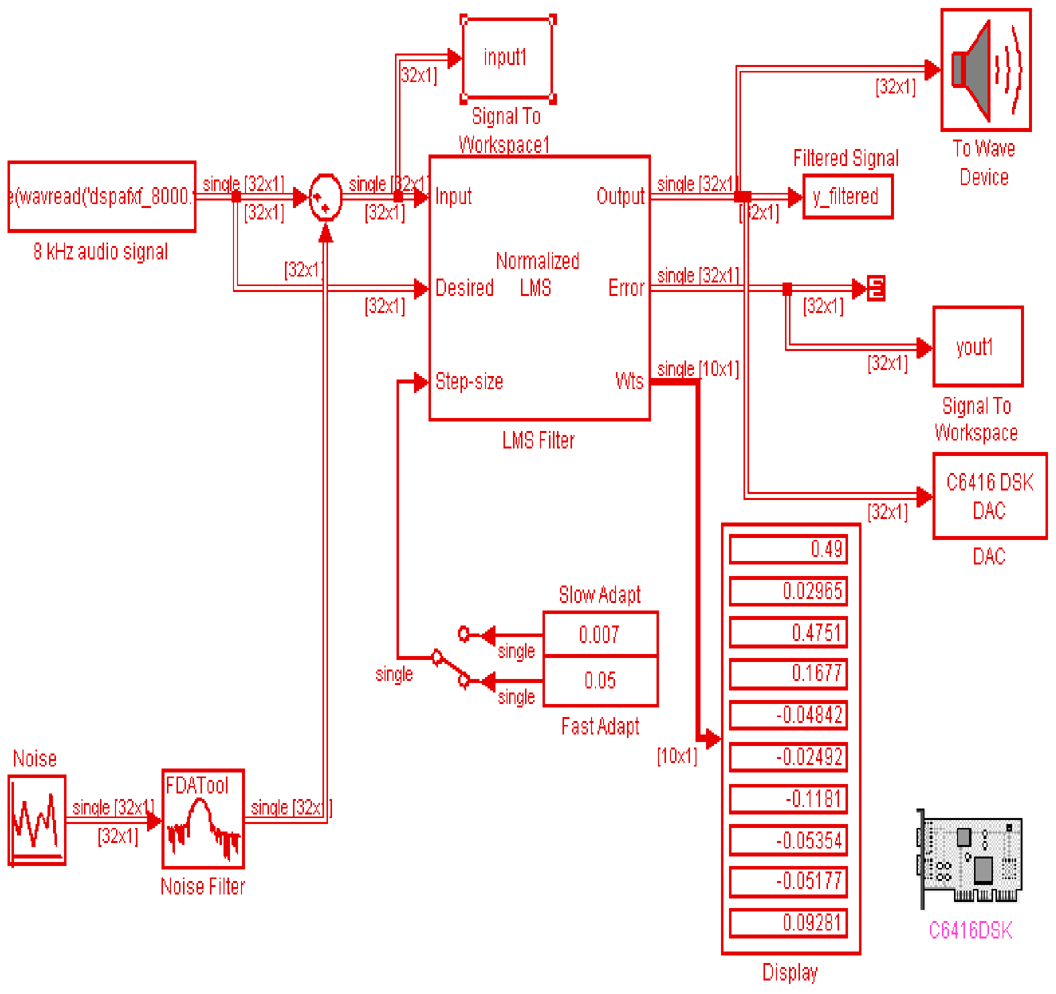


PART B: Hardware Experiment for Music Noise Cancellation:

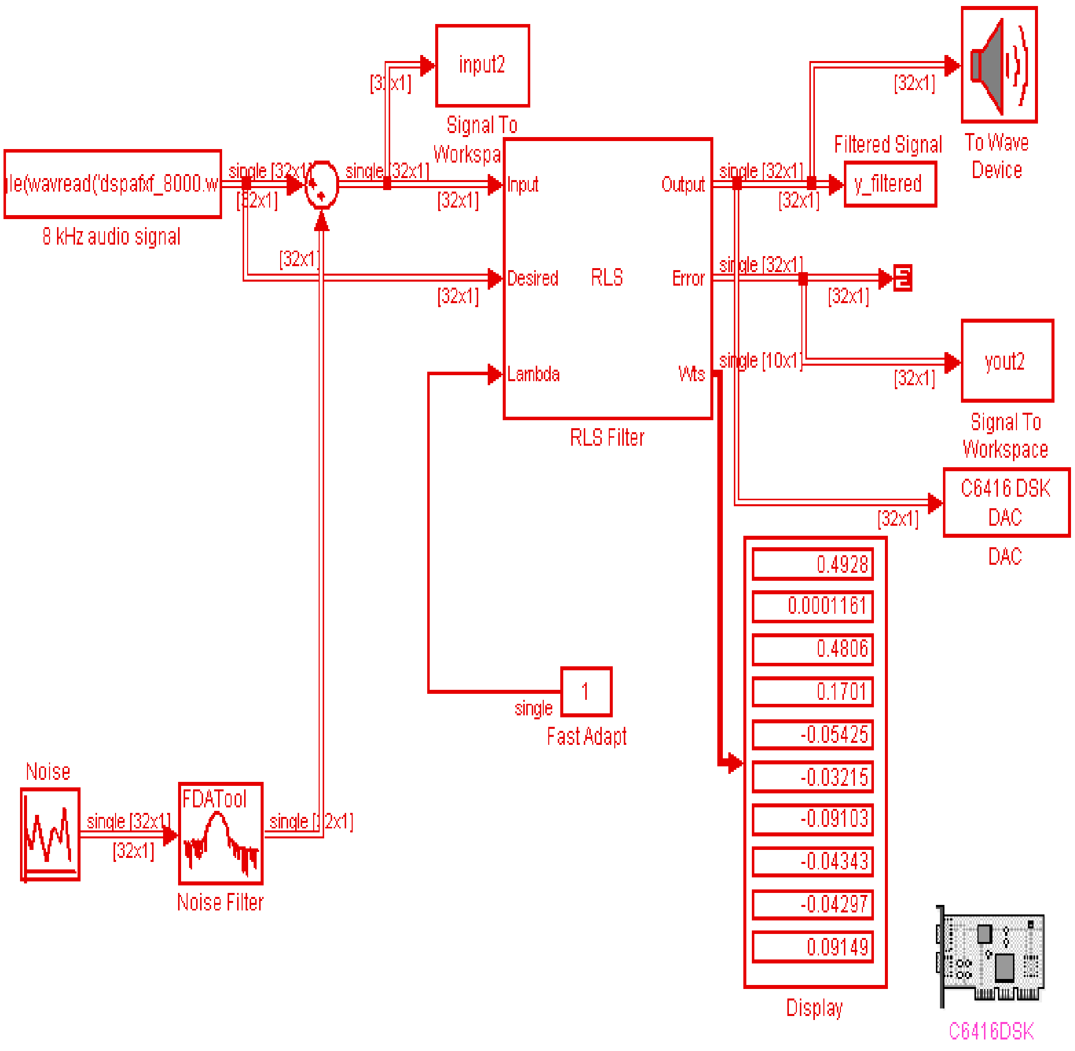
B.4: LMS Algorithm.



B.5: NLMS Algorithm.



B.6: RLS Algorithm.


input2
Signal To Workspa
[32x1]
le(wavread('dspafxf_8000.w
8 kHz audio signal
single [32x1]
[32x1]
single [32x1]
[32x1]
Input
Desired
RLS
Lambda
Output
Error
Wts
RLS Filter
single [32x1]
[32x1]
Filtered Signal
y_filtered
To Wave Device
single [32x1]
[32x1]
single [10x1]
yout2
Signal To Workspace
C6416 DSK DAC
DAC
0.4928
0.0001161
0.4806
0.1701
-0.05425
-0.03215
-0.09103
-0.04343
-0.04297
0.09149
Display
1
single
Fast Adapt
Noise
single [32x1]
[32x1]
FDATool
Noise Filter
single [32x1]
C6416DSK