ADAPTIVE ECHO CANCELLATION ANALYSIS

by

Yongdong Sun
(B.Eng., Shenyang, China, June 1993)

A Project
presented to Ryerson University
in partial fulfillment of the
requirements for the degree of

Master of Engineering

in the Program of
Electrical and Computer Engineering

Toronto, Ontario, Canada, 2004

Author's Declaration

I hereby declare that I am the sole author of this Research Paper.

I authorize Ryerson University to lend this Research Paper to other institutions or individuals for the purpose of scholarly research.

Signature [illegible]

I further authorize Ryerson University to reproduce this Research Paper by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signature [illegible]

Borrow List

Abstract

Title: Adaptive Echo Cancellation Analysis

Echo cancellation is a classic problem in DSP and digital communication. Adaptive echo cancellation is an application of adaptive filtering to the attenuation of undesired echo in the telecommunication network. This is accomplished by modeling the echo path using an adaptive filter and subtracting the estimated echo from the echo-path output.

In this project, the concept of echo cancellation and echo cancellation systems are studied, simulated, and implemented in Matlab and TI TMS320C6711 DSK. The LMS, NLMS, Fast Block LMS and RLS algorithm are investigated for echo canceller and two double talk detection algorithms: the Geigel algorithm and the normalized cross-correlation algorithm are presented and combined with NLMS adaptive algorithm against double talk. Finally The adaptive echo cancellation system successfully developed by the NLMS and normalized cross-correlation DTD algorithms meet the general ITU G. 168 requirements and show excellent robustness against double talk.

Acknowledgements

Many thanks to Dr. Mike Kassam, my supervisor, for his thoughtful advice, assistance and comments.

Also, thanks to my wife and my family for supporting me during this project.

Table of Contents

Chapter 1

Introduction

1.1 Adaptive Echo Cancellation

Echo is a phenomenon in which a delayed and distorted version of an original sound or electrical signal is reflected back to the source. In real life, echoes often occurrence among conversations. The echoes of speech waves can be heard as they are reflected from the floor, wall and other neighboring objects. In such a case when the reflected wave arrives a few tens of milliseconds delay after the direct sound, it can be heard as an obvious echo. These echoes are bothering and may unexpectedly interrupt a conversation. Thus it is desired to eliminate these echoes. In telephone communication, there are two main types of echo: network and acoustic echoes. The network echo results from the impedance mismatch at points along the transmission line, for example, at the hybrids of a public switched telephony network (PSTN) exchange, where the subscriber two-wire lines are connected to four-wire lines. Acoustic echo is due to the acoustic coupling between the loudspeaker and microphone in hands-free telephone systems, for example, if a communication is between one or more hands-free telephones (or speaker phones), then acoustic feedback paths are set up between the telephone's loudspeaker and microphone at each end. Acoustic Echo is more hostile than network echo mainly because of the different nature of echo paths.

The solution to these echo problems is to eliminate the echo with an echo suppressor or echo canceller.

The problems with echo suppressor are:

- only one speaker can talk at a time
- clips speech
- take time to detect the beginning of speech

Now using adaptive filter to reduce the echo and increase communication quality is a common technology in communication system, but there are still some challenges :

- many existing adaptive algorithms give different performance
- different adaptive algorithms need different parameter setting
- overcome double talk disturbance

This project focuses on the following achievements:

(1) Providing an easy approach to compare and review various adaptive algorithms based on their convergence rate, steady state ERLE and complexity of implementation, etc.

(2) Supplying different pre-designed double talk detection methods that can be easily configured with different parameter to obtain the expected performance.

(3) Implementation of AEC and DTD algorithms to work practically and efficiently using smooth estimation and the pre-set parameters selected from experiments.

Adaptive cancellation of such acoustic echo has became very important in hands-free communication systems, e.g. tele-conference, video-conference and PC telephony systems.

Echo canceller is a better solution to the acoustic echo problem, which allows both speakers to talk at the same time.

The Basic Structure of Adaptive Echo Canceller is shown in Figure 1.1 and the estimated echo signal $\hat{y}(t)$ generated by an adaptive filter eliminates the echo signal $y(t)$. The coefficients of the adaptive filter are adjusted by adaptive algorithm according to the estimation error signal $e(t)$.

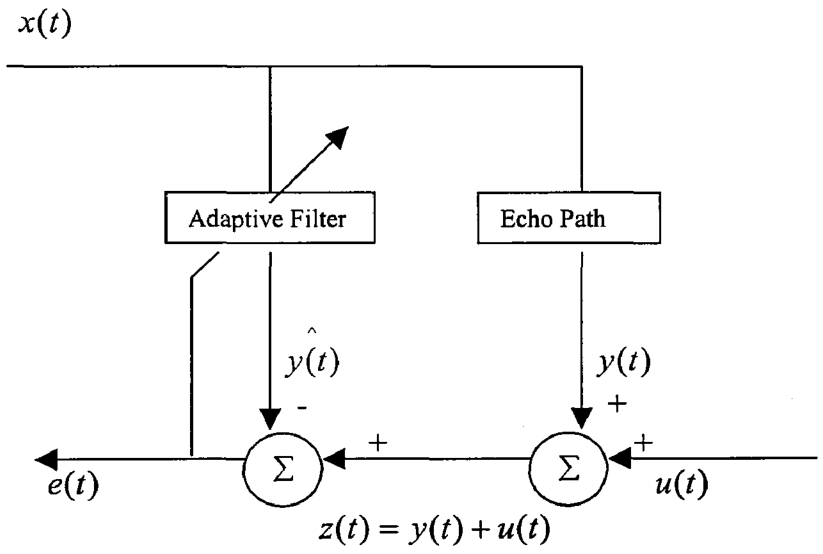


Figure 1.1 Basic structure of adaptive echo canceller: far end signal $x(t)$, echo signal $y(t)$, near end signal $u(t)$, received signal $z(t)$, estimated echo signal $\hat{y}(t)$ and error signal $e(t)$.

1.2 Adaptive Filters

Adaptive usually deploys a traversal Finite Impulse Response (FIR) structure due to its guaranteed stability and the adaptation of the FIR filter coefficients is controlled by an adaptive algorithm. The adaptive algorithm is the heart of an AEC, which decides the convergence behavior and tracking behavior of the AEC. The tracking behavior indicates how fast the adaptive filter can follow enclosure dislocations, whereas the convergence behavior is studied as an initial adjustment of the adaptive filter to the impulse response of the room or car.

In Figure 1.2, w represents the coefficients of the FIR filter tap weight vector, x(k) is the input vector samples, z^{-1} is the delay unit, y(k) is the adaptive filter output, d_k is the desired echoed signal and ε_k is the estimation error at time k.

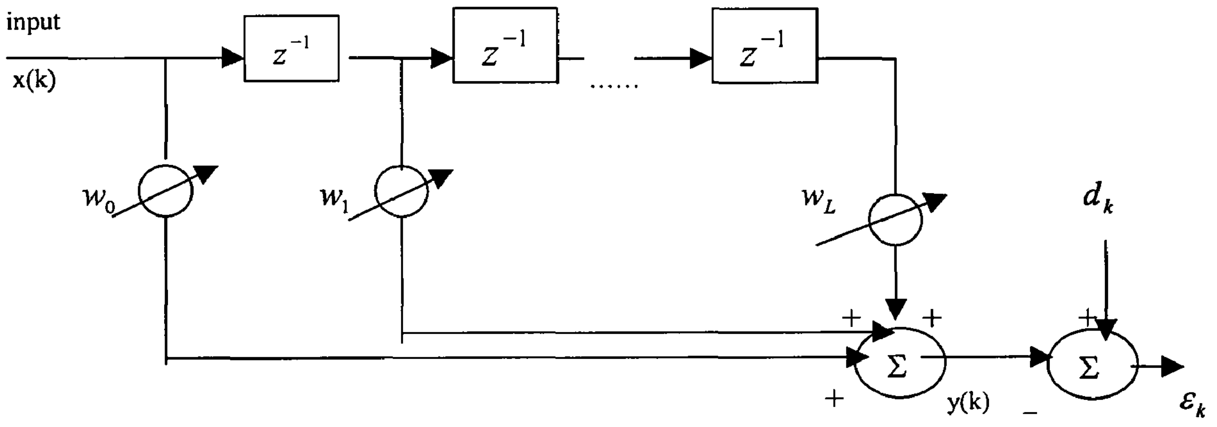


Figure 1.2 Adaptive filter block diagram

The adaptive filter is used to calculate the difference between the desired signal and the adaptive filter output, ε_k. This error signal is fed back into the adaptive filter and its coefficients are changed algorithmically to minimize the cost function that is a function of ε_k. In acoustic echo cancellation, the optimal output of the adaptive filter is the value of the unwanted echoed signal. When the adaptive filter output is equal to desired signal, the error signal becomes 0, in this ideal situation, the echo signal will be completely cancelled and the far end user will not hear his original speech returned to him.

1.3 Project Objective

In the project, the LMS, NLMS, Fast Block LMS and RLS algorithm are studied and alternatively used in adaptive acoustic echo canceller. The NLMS with two different Double Talk detection algorithms are integrated in the adaptive echo canceller against double talk. The performance and parameters of these adaptive algorithms such as filter length, step size and convergence speed are studied in details and the two Double Talk algorithms: detection algorithms: the Geigel algorithm and the normalized cross-correlation algorithm are studied and simulated with NLMS algorithm in Matlab and TI TMS320C6711 DSK. The ITU G.168 standard is used to study the performance of the adaptive echo canceller.

Chapter 2

Echo Cancellation Adaptive Methods

Most echo cancellers use variants of the LMS adaptation algorithm [1] [2][3]. The attractions of the LMS are its relatively low memory and computational requirements and its ease of implementation and monitoring. In practice, LMS adaptation has produced effective line echo cancellation systems.

2.1 LMS Algorithms

In Figure 1.1, the error signal ε_k is

$$\varepsilon_k = d_k - y_k \tag{2.1.1}$$

and

$$y_k = X_k^T W = W^T X_k \tag{2.1.2}$$

So,

$$\varepsilon_k = d_k - X_k^T W = d_k - W^T X_k \tag{2.1.3}$$

We can square ε_k to get the instantaneous squared error,

$$\varepsilon_k^2 = d_k^2 + W^T X_k X_k^T W - 2 d_k X_k^T W \tag{2.1.4}$$

$$E[\varepsilon_k^2] = E[d_k^2] + W^T E[X_k X_k^T] W - 2E[d_k X_k^T] W \tag{2.1.5}$$

Let R be defined as the autocorrelation matrix

$$R = E[X_k X_k^T] = E \left\{ \begin{array}{cccc} x_k x_k & x_k x_{k-1} & \cdots & x_k x_{k-N+1} \\ x_{k-1} x_k & x_{k-1} x_{k-1} & \cdots & x_{k-1} x_{k-N+1} \\ \vdots & \vdots & & \vdots \\ x_{k-N+1} x_k & x_{k-N+1} x_{k-1} & \cdots & x_{k-N+1} x_{k-N+1} \end{array} \right\} \tag{2.1.6}$$

Let P be similarly defined as the cross-correlation matrix

$$P=E[d_k X_k^T] = E\left\{\begin{matrix} d_k x_k \\ d_k x_{k-1} \\ \vdots \\ d_k x_{k-N+1} \end{matrix}\right\} \tag{2.1.7}$$

And the mean-square error can be designated by ξ and we can obtain the following expression:

$$\mathrm{MSE} \cong \xi = E[\varepsilon_k^2]=E[d_k^2]+W^T RW-2PW \tag{2.1.8}$$

Now it is clear that the mean-square error ξ is a quadratic function of the weight vector W. When we expand this expression, the elements of W will appear in first and second degree only. If we have two weights, we can get the quadratic error function, or performance surface. (the vertical axis represents the mean-square error and the horizontal axes the values of the two weights.) The point at the "bottom of the surface" is projected onto the weight-vector plane and then we can obtain the optimal weight vector W^* or the minimum mean-square error point. And there is only a single global optimum in this performance surface. To search the performance surface for minimum point, we can do gradient methods:

$$\nabla(\xi) \cong \frac{\partial \xi}{\partial W} = \left[\frac{\partial \xi}{\partial w_0} \quad \frac{\partial \xi}{\partial w_1} \cdots \frac{\partial \xi}{\partial w_{N-1}}\right]^T = 2RW-2P \tag{2.1.9}$$

To obtain the minimum mean-square error, the weight vector W is set at its optimal value W^* and where the gradient is zero.

$$\nabla = 0 = 2RW^* - 2P \tag{2.1.10}$$

$$RW^* = P \tag{2.1.11}$$

Then the

$$W^*=R^{-1}P. \tag{2.1.12}$$

To develop the Least Mean Square (LMS) algorithm, we take ε_k^2 itself as an estimate of ξ_k. Then, for each iteration in the adaptive process, we have a gradient estimate of the form

$$\nabla_k = \begin{bmatrix} \frac{\partial \varepsilon_k^2}{\partial w_0} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial \varepsilon_k^2}{\partial w_{N-1}} \end{bmatrix} = 2\,\varepsilon_k \begin{bmatrix} \frac{\partial \varepsilon_k}{\partial w_0} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial \varepsilon_k}{\partial w_{N-1}} \end{bmatrix} = -2\,\varepsilon_k X_k \tag{2.1.13}$$

With this simple estimate of the gradient, we can have

$$W_{k+1} = W_k - \mu\,\nabla_k = W_k + 2\,\mu\,\varepsilon_k X_k \tag{2.1.14}$$

This is the LMS algorithm. μ is the gain constant that regulates the speed and stability of adaptation. The LMS algorithm can be implemented in a practical system without squaring, averaging, or differentiation and is simple and efficient. For each iteration the LMS algorithm requires 2N additions and 2N+1 multiplications (N is the filter length, N for calculating the output, one for $2\mu\,\varepsilon_k$ and an additional N for the scalar by vector multiplication).

2.2 NLMS Algorithms

When μ is optimized as described by $\mu(n) = \dfrac{\alpha}{\beta + x_M^T(n)x(n)}$, $\alpha \in (0,2), 0 \leq \beta$, (2.2.1)

β guarantees that the denominator never becomes zero, while α is a relaxation factor, the normalized LMS algorithm results. In this case, μ is time varying.

The echo canceller coefficients $w_k(m)$ are adapted to minimize the energy of the echo signal on a telephone line. Assuming that the speech signals $x_A(m)$ and $x_B(m)$ are uncorrelated, the energy on the telephone line from B to A is minimized when the echo canceller output $\hat{x}_A^{echo}(m)$ is equal to the echo $\hat{x}_B^{echo}(m)$ on the line. The echo canceller coefficients can be adapted using the least mean squared error (LMS) adaptation algorithm or one of the most widely used algorithms for adaptation of the coefficients of an echo canceller is the normalized least mean square error(NLMS) method. The time-updated equation describing the adaptation of the filter coefficient vector is

$$w(m) = w(m-1) + \mu \frac{e(m)}{x(m)_A^T x_A(m)} x_A(m) \tag{2.2.2}$$

where $x_A(m) = [x_A(m),...,x_A(m-N+1)]$ and $x(m) = [x_0(m),...,x_{P-1}(m-N+1)]$ are the input signal vector and the coefficient vector of the echo canceller, and $e(m)$ is the difference between the signal and the echo line and the output of the echo synthesizer. The normalized quanity $x(m)_A^T x_A(m)$ is the energy of the input speech to the adaptive filter. The scalar μ is the adaptation step size, and controls the speed of convergence, the steady-state error and the stability of the adaptation process.

Each iteration of the NLMS algorithm requires 3N+1 multiplications (N is filter length), this is only N more than the standard LMS algorithm.

2.3 Fast Block LMS Algorithm

In the normalized LMS algorithm, the filter coefficients of a finite-duration impulse (FIR) filter are adapted in the time domain. Because the *Fourier transform* maps time-domain signals into the frequency domain and the inverse Fourier transform provides the inverse mapping that takes the signals back into the time domain, it is workable to perform the adaptation of filter coefficients in the frequency domain, which is called frequency-domain adaptive filtering.

In a block-adaptive filter, shown in Figure 2.1, the input data sequence x(n) is sectioned into L -point blocks by means of a serial-to-parallel converter, and the blocks of input data so produced are applied to an FIR filter of length M , one block at a time. The coefficients of the filter are updated after the collection of each block of data samples, so that adaptation of the filter proceeds on a block-by-block basis rather than on a sample-by-sample basis as in the conventional LMS algorithm.

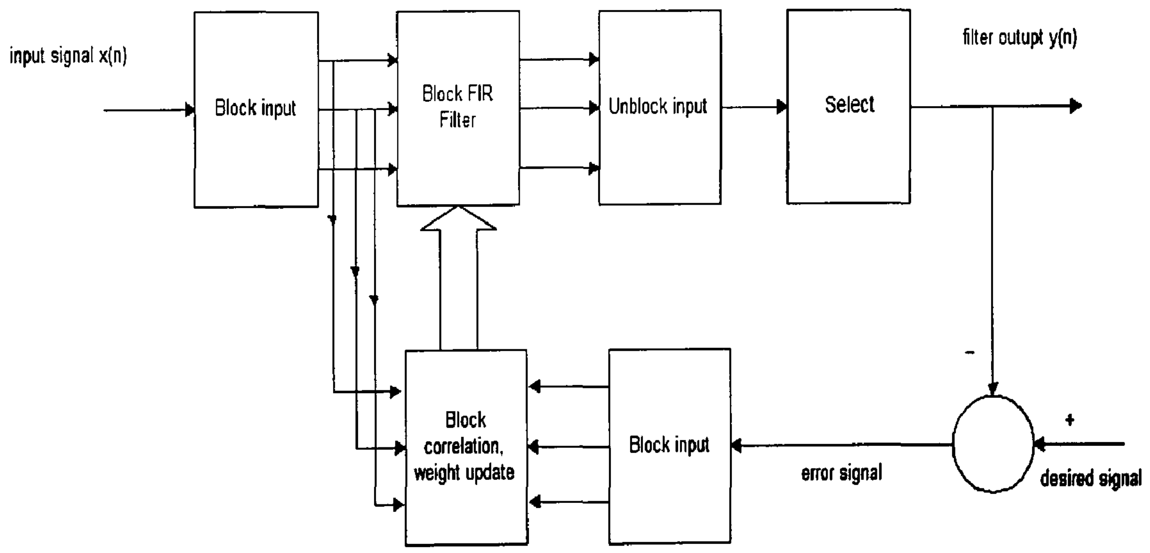


Figure 2.1 Block-adaptive filter

The Fast Block LMS algorithm represents a precise frequency-domain implementation of the block LMS algorithm and its convergence properties are identical to those of the block LMS algorithm.

Fast Block LMS Algorithm is based on Overlap-Save Sectioning (Assuming Real-Valued Data).

Initialization:

$\hat{w}(0)$ = 2M by 1 zero vector (2.3.1)

$Pi(0) = \delta_i$ (2.3.2)

where the δ_i are small positive constants and $i = 0, \ldots, 2M-1$

Notations:

0	= M by 1 zero vector
FFT	= fast Fourier transformation
IFFT	= inverse fast Fourier transformation
α	= adaptation constant

Computation: For each new block of M input samples, compute:

Filtering

$$U(k) = diag\{FFT[u(kM-M),\ldots,u(kM-1),u(kM),\ldots,u(kM+M-1)]^T\} \quad (2.3.3)$$

$y^T(k)$ = the last M elements of IFFT $[U(k)\hat{W}(k)]$ (2.3.4)

Error estimation

$$e(k) = d(k) - y(k) \quad (2.3.5)$$

$$E(k) = FFT\begin{bmatrix} 0 \\ e(k) \end{bmatrix} \quad (2.3.6)$$

Signal-power estimation

$$Pi(k) = \gamma Pi(k-1) + (1-\gamma)|Ui(k)|^2 \quad , \quad i = 0,1,\ldots,2M-1 \quad (2.3.7)$$

$$D(k) = diag[P_0^{-1}(k), P_1^{-1}(k), \ldots, P_{2M-1}^{-1}(k)] \quad (2.3.8)$$

Filter coefficient adaptation

$\phi(k)$ = the first M elements of IFFT$[D(k)U^H(k)E(k)]$ (2.3.9)

$$\hat{W}(k+1) = \hat{W}(k) + \alpha FFT\begin{bmatrix} \phi(k) \\ 0 \end{bmatrix} \quad (2.3.10)$$

Choice of Block Size:

The block size L in relation to the length M of the adaptive filter. There are three possible choices that can be chosen, each with its own practical implications:

- $L = M$ which is optimum choice
- $L < M$ advantage of reduced processing delay and if the block size smaller than the filter length, the adaptive filtering algorithm computationally is still more efficient than the conventional LMS algorithm.
- $L > M$ increased redundant operations in the adaptive process, because the estimation of the gradient vector now uses more information than the filter itself.

In the fast block LMS algorithm there are 5 FFT transforms, requiring approximately 2M log(2M) real multiplications each, and also other 16M operations (when updating the parameters, computing the errors, element-wise multiplications of FFT transformed vectors), so the total is 10M log(2M) + 16M = 10M log(M) + 26M.

2.4 RLS algorithm

The Recursive Least Square (RLS) algorithm is used to minimize the cost function:

$$\zeta = \sum_{k=1}^{n} \lambda^{n-k} e_n^2(k) \tag{2.4.1}$$

Where k=1 is the time at which the RLS algorithm starts and λ is a small positive constant very close to, but smaller than 1.

Unlike the LMS algorithm and its derivatives, the RLS algorithm directly considers the values of previous error estimations. In fact, it is impossible to use all previous values of the estimation error from the start of the algorithm with the cost equation in real FIR implementation for the computation complexity. In practice, only a finite number of previous values are considered, this number corresponds to the order of the RLS FIR filter, N.

$$y_n(k) = w^T(n)x(k),\ e_n(k) = d(k) - y_n(k) \tag{2.4.2}$$

$$d(n) = [d(1), d(2) \ldots d(n)]^T\ ,\ e(n) = [e_n(1), e_n(2) \ldots e_n(n)]^T \tag{2.4.3}$$

$$e(n) = d(n) - y(n) \tag{2.4.4}$$

$$\zeta = \sum_{k=1}^{n} \lambda^{n-k} e_n^2(k) = e^T(n)\,\tilde{\Lambda}\,e(n) = d^T\,\tilde{\Lambda}\,d - 2\,\tilde{\theta}_\lambda^T\,w + w^T\,\tilde{\Phi}_\lambda\,w \tag{2.4.5}$$

where $\tilde{\theta}_\lambda(n) = x(n)\,\tilde{\Lambda}\,d(n)$, $\tilde{\Phi}_\lambda(n) = x(n)\,\tilde{\Lambda}(n)x^T(n)$

$$\tilde{w}(n) = \tilde{\Phi}_\lambda^{-1}(n)\ \tilde{\theta}_\lambda(n) \tag{2.4.6}$$

Find the inverse matrix using recursive form,

$$\tilde{\Phi}_\lambda^{-1}(n) = \lambda\,\tilde{\Phi}_\lambda^{-1}(n-1) + x(n)x^T(n) = \lambda^{-1}\,(\tilde{\Phi}_\lambda^{-1}(n-1) - k(n)x^T(n)\ \tilde{\Phi}_\lambda^{-1}(n-1)) \tag{2.4.7}$$

where $k(n) = \dfrac{\lambda^{-1}\,\tilde{\Phi}_{\lambda}^{-1}(n-1)x(n)}{1+\lambda^{-1}x^{T}(n)\,\tilde{\Phi}_{\lambda}^{-1}(n-1)x(n)} = \dfrac{1}{\lambda + x^{T}(n)u(n)}u(n)$

$$\tilde{\theta}_{\lambda}(n) = \lambda\,\tilde{\theta}_{\lambda}(n-1) + x(n)d(n) \qquad (2.4.8)$$

so RLS algorithm filter weight update vector

$$\bar{w}(n) = \tilde{\Phi}_{\lambda}^{-1}(n)\;\tilde{\theta}_{\lambda}(n)$$

$$= \tilde{\Phi}_{\lambda}^{-1}(n-1)\;\tilde{\theta}_{\lambda}(n-1) - k(n)x^{T}\;\tilde{\Phi}_{\lambda}^{-1}(n-1))\;\tilde{\theta}_{\lambda}(n-1) + k(n)d(n)$$

$$= \bar{w}(n-1) + k(n)\,\bar{e}_{n-1}(n) \qquad (2.4.9)$$

where $\bar{e}_{n-1}(n) = d(n) - \bar{w}^{T}(n-1)x(n)$

Each iteration of the RLS algorithm requires 4N^2 multiplication operations (N is the filter length).

Chapter 3

Double Talk Detection Algorithms

During double-talking periods, there exists the other end speaker's adaptive signal $v(n)$ which acts as a very large interference to the adaptive filter. If the adaptive filter continues to adjust its coefficients during double-talking periods, the adaptive filter will be greatly disturbed and will quickly diverge from its convergence state. Therefore, double-talk detectors are used in adaptive voice echo cancellers to detect the double-talking periods, and the adaptive filter coefficients adjustment is prohibited during these periods to prevent the echo canceller from being disturbed by the other end speaker's signal. Double-talk detection plays a very important part in adaptive acoustic echo cancellation. The basic requirement for a double-talk detector is that it can detect double-talking quickly and accurately and it should also have the ability to distinguish the double-talking conditions from echo path variations and quickly track variations in the echo path.

3.1 The Geigel algorithm

A simple approach is to measure the power of the received signal and compare it to the power of the far-end signal; as shown in Figure 1.1, $z(t)$ is the received signal, $x(t)$ is the far end signal, this is the Geigel algorithm [4] and the decision variable is defined as

$$d_G(t) = \frac{|z(t)|}{\max\{|x(t)|,\ldots,|x(t-n+1)\}} . \qquad (3.1.1)$$

If $d_G(t)$ is larger than some preset threshold, T_G, it is treated that Double Talk is occurring, otherwise not. The Geigel detector is computationally simple and need little memory, but the choice of T_G is not easy to obtain good performance.

3.2 Normalized cross-correlation algorithm

As shown in Figure 1.1, the power of the received signal can be written as

$$\sigma_z^2(t) = h_t^T R_x(t) h_t + \sigma_v^2(t) \tag{3.2.1}$$

where $R_x(t) = E\{x_t x_t^T\}$ is the $L \times L$ covariance matrix of the far-end signal,

$\sigma_v^2(t)$ is the power of the near-end signal,

$\sigma_z^2(t)$ is the power of the received signal,

h_t is room acoustic response.

As $y(t) = h_t^T x_t$, (3.2.2)

Then $r_{xy}(t) = E\{x_t y(t)\} = R_x(t) h_t$, (3.2.3)

Yielding,

$$h_t = R_x^{-1}(t) r_{xy}(t) \tag{3.2.4}$$

So,

$$\sigma_z^2(t) = r_{xy}^T(t) R_x^{-1}(t) r_{xy}(t) + \sigma_v^2(t). \tag{3.2.5}$$

When there is no near-end signal is present, $v(t) = 0$, then $z(t) = y(t)$ and

$$\sigma_z^2(t) = r_{xy}^T(t) R_x^{-1}(t) r_{xy}(t), \qquad \text{with } r_{xz}(t) = E\{x_t z(t)\} \tag{3.2.6}$$

The detection variable is $d(t) = \dfrac{r_{xz}^T(t) R_x^{-1} r_{xz}(t)}{\sigma_z^2(t)} = \dfrac{r_{xz}^T(t) h}{\sigma_z^2(t)}$, (3.2.7) where h is the estimated room acoustic response and $r_{xz}(t)$ is the estimated cross-correlation between the far-end signal and the received signal.

The nominator is the power of the received signal if no near-end signal is present. The denominator is the actual power of the received signal. Thus, if no near end signal is present, $d(t) \approx 1$, otherwise $d(t) < 1$. The Double Talk decision is formed as $d(t) < T_n$, double talk present, otherwise not.

When there is no near-end talk, it is known that the decision variable d is =1 for v=0 and d < 1 for v $\neq$0 when there exist near-end talk. To calculate the decision variable d, in implementation, we use the following smoothed estimate:

$$r_{xz}(t) = \alpha r_{xz}(t-1) + (1-\alpha)X(t)z(t) \tag{3.2.8}$$

$$\sigma_z^2(t) = \alpha\sigma_z^2(t-1) + (1-\alpha)z(t)^2 \tag{3.2.9}$$

where α is a smoothing factor which lies in (0,1). In experiment, α is set to be 0.9 and

$$r_{xz}(t) = \sum_{k=0}^{L-1} X(t-k)z(t-k), \tag{3.2.10}$$

using a L length sliding window, the $r_{xz}(t)$ can be estimated [5] [6][7][8].

3.3 Evaluation procedure of double detectors

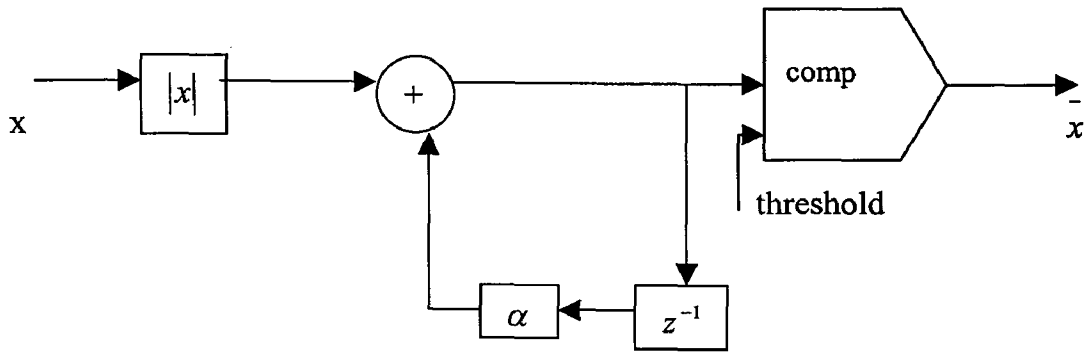


Figure 3.1 Voice activity detector

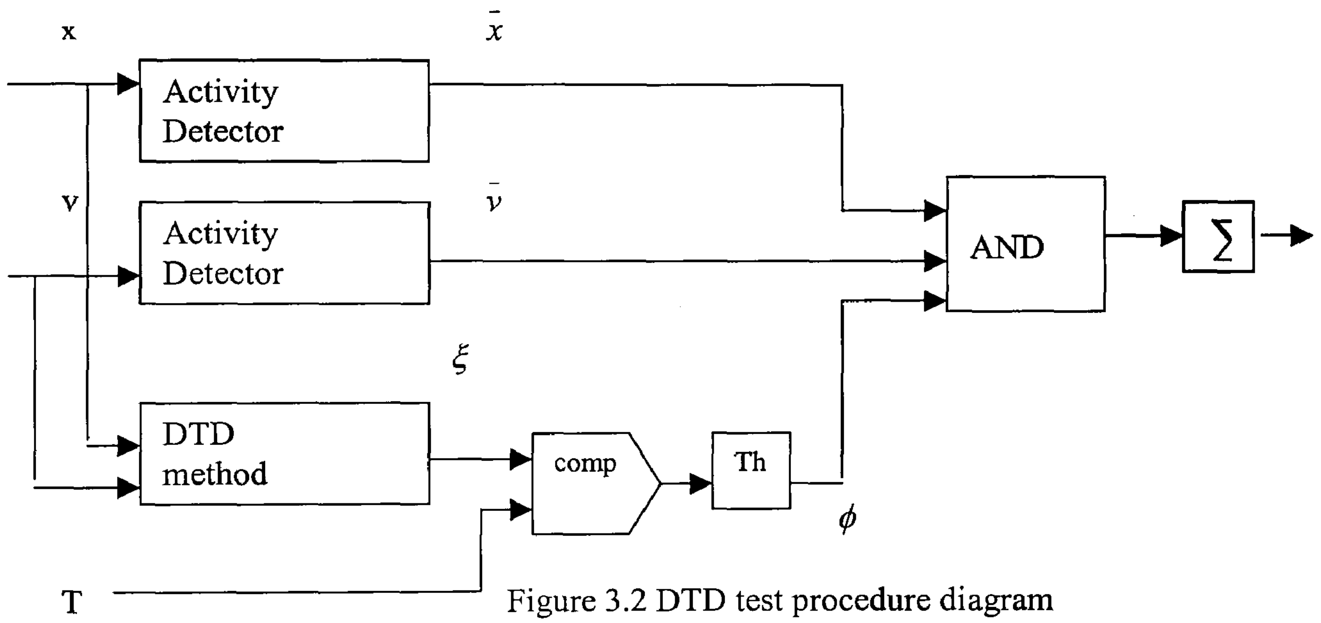


Figure 3.2 DTD test procedure diagram

In the DTD methods discussed in the previous section, the role of threshold T is essential to the performance. However, there hasn't been a systemic approach to select the value of T. In the paper [6][9][10], an objective technique is proposed for evaluating doubletalk detectors in acoustic echo cancellers. It view DTD as a binary detection problem and use it in actual operating environments. The general characteristics of a binary detection scheme are as follows.

Probability of False Alarm (P_f): Probability of declaring detection when a target is not present.

Probability of Detection (P_d): Probability of successful detection when a target is present.

Probability of Miss (P_m =1- P_d): Probability of detection failure when a target is present.

$$P_f = \frac{\sum \phi \bullet x}{N} \tag{3.3.1}$$

$$P_d = \sum \phi \bullet \bar{x} \bullet \bar{v} / \sum \bar{x} \bullet \bar{v} \tag{3.3.2}$$

$$P_m = 1 - \sum \phi \bullet \bar{x} \bullet \bar{v} \; / \; \sum \bar{x} \bullet \bar{v} \tag{3.3.3}$$

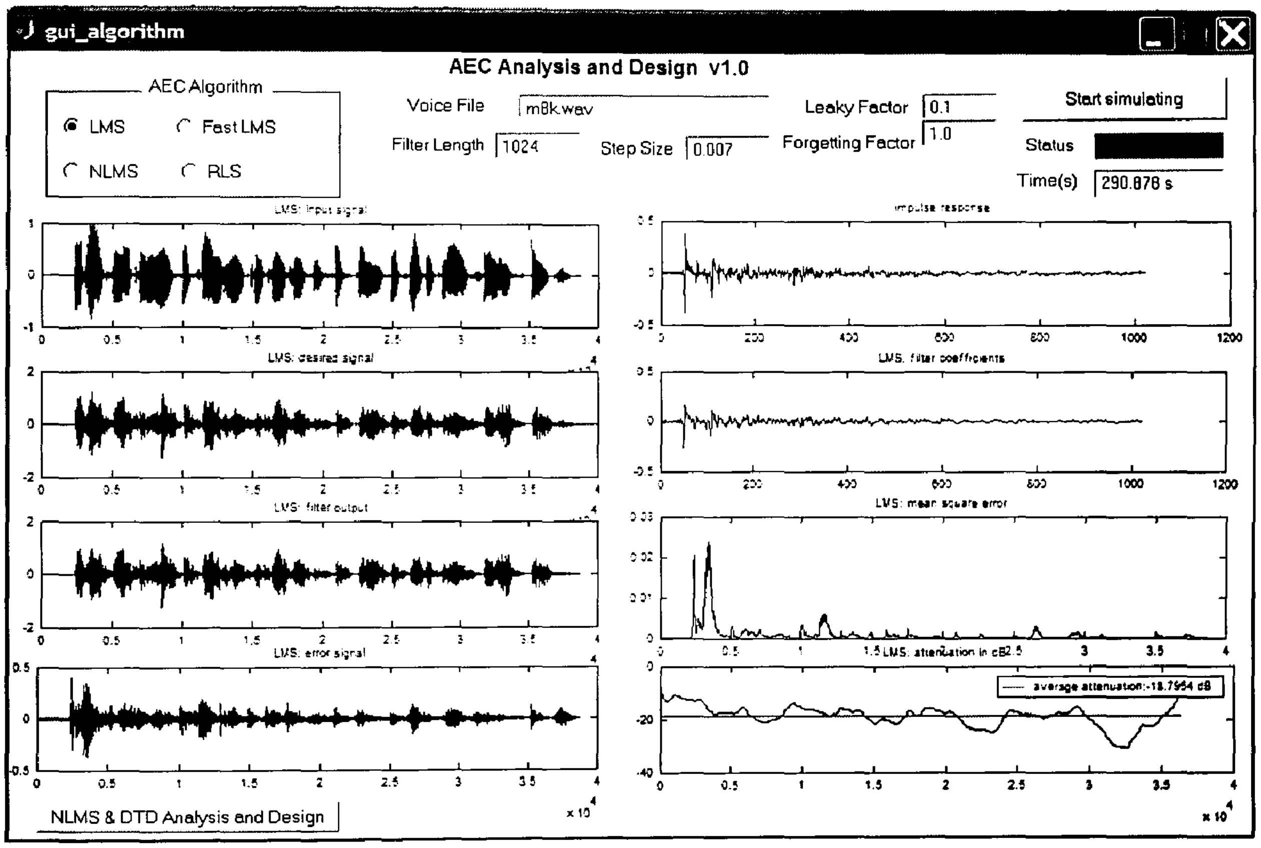
Chapter 4

Matlab Simulation & Analysis

4.1 Adaptive algorithms

Graphical User Interface

In this project, the command 'guide' is used in MATLAB v6.5 to make the Graphical User Interface (GUI) of the AEC Analysis and Design program to help study different AEC algorithms. The GUI of the program can be seen in Figure 4.1.


gui_algorithm

AEC Analysis and Design v1.0

AEC Algorithm: [✓ LMS] Fast LMS NLMS RLS

Voice File: m8k.wav

Filter Length: 1024

Step Size: 0.007

Leaky Factor: 0.1

Forgetting Factor: 1.0

Status

Time(s): 290.878 s

Figure 4.1 AEC Analysis and Design program GUI

In this program, user can choose:

1. AEC Algorithm
   a. LMS algorithm
   b. NLMS Algorithm

c. Fast LMS Algorithm

2. Input signal file (wav format)

3. Filter Length

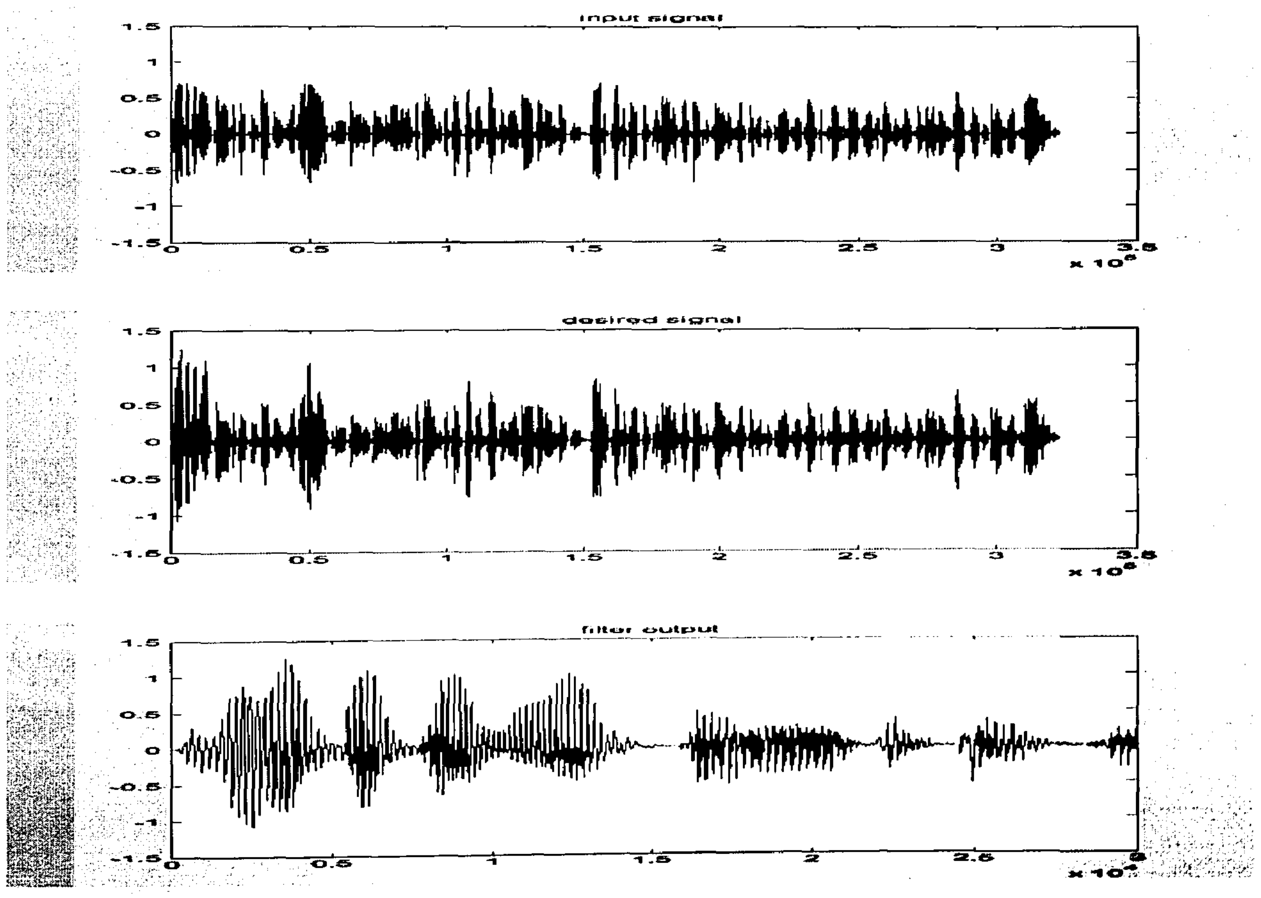
4. Step Size

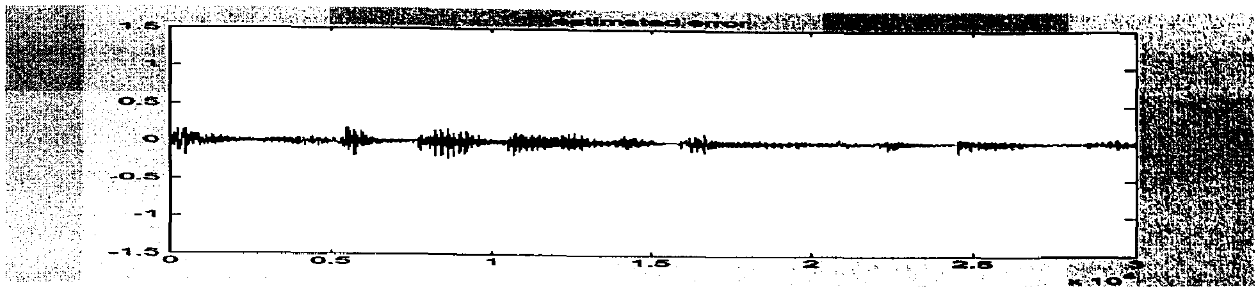
5. Leaky Factor

6. Forgetting Factor

4.1.1 LMS

The Figure 4.2 shows the desired signal, adaptive output signal, estimation error and mean square error for the LMS algorithm with FIR Filter length of 1000, step size of 0.007.





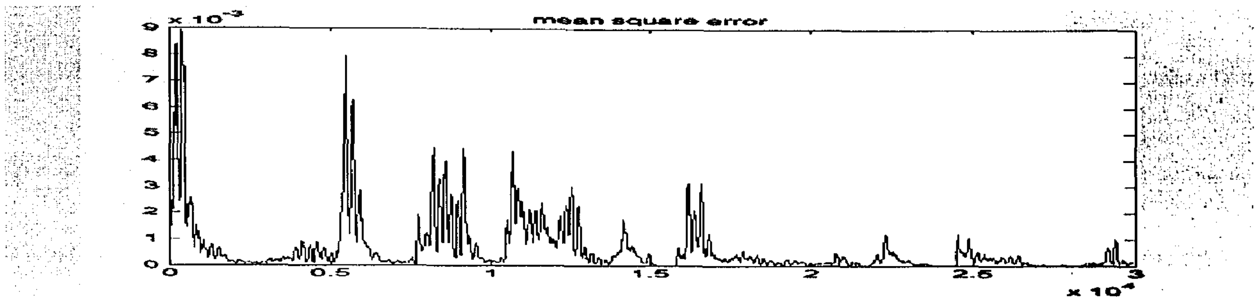
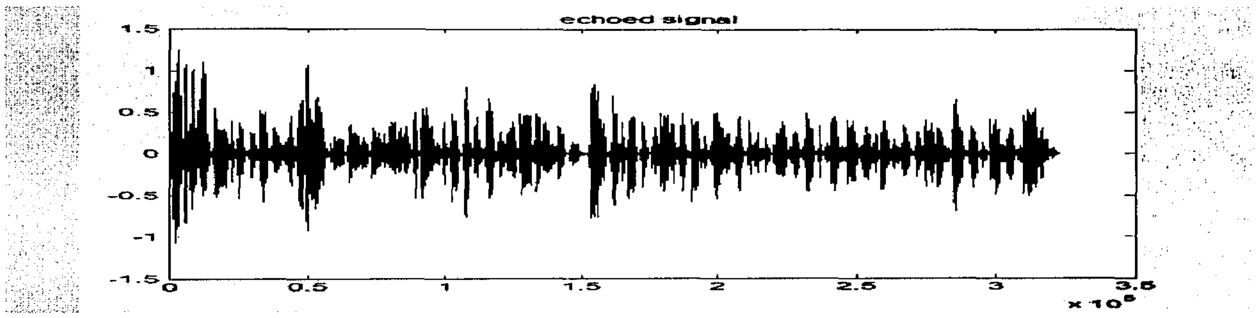


Figure 4.2 LMS algorithm



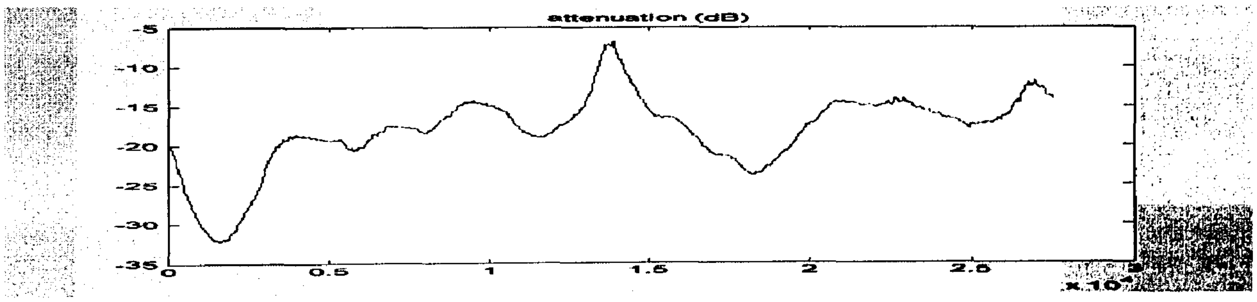
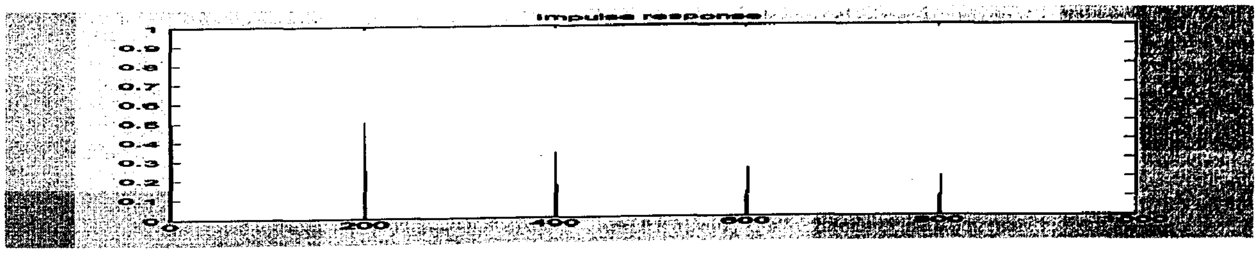


Figure 4.3 LMS echo signal attenuation

The average attenuation for LMS FIR filter is –18.16dB.



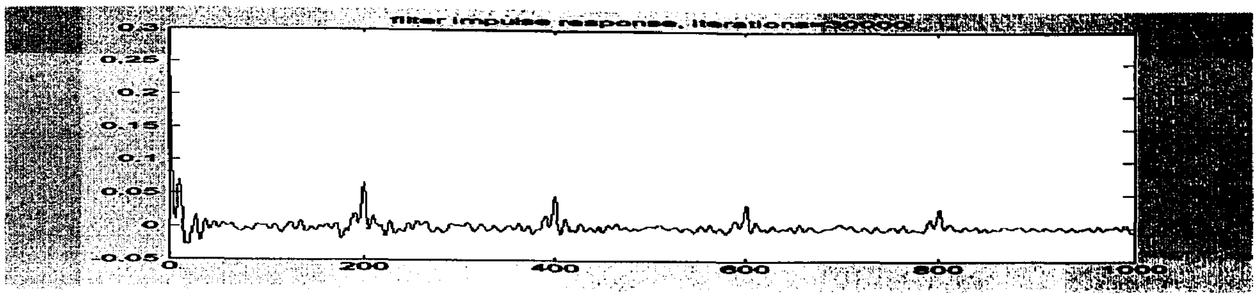
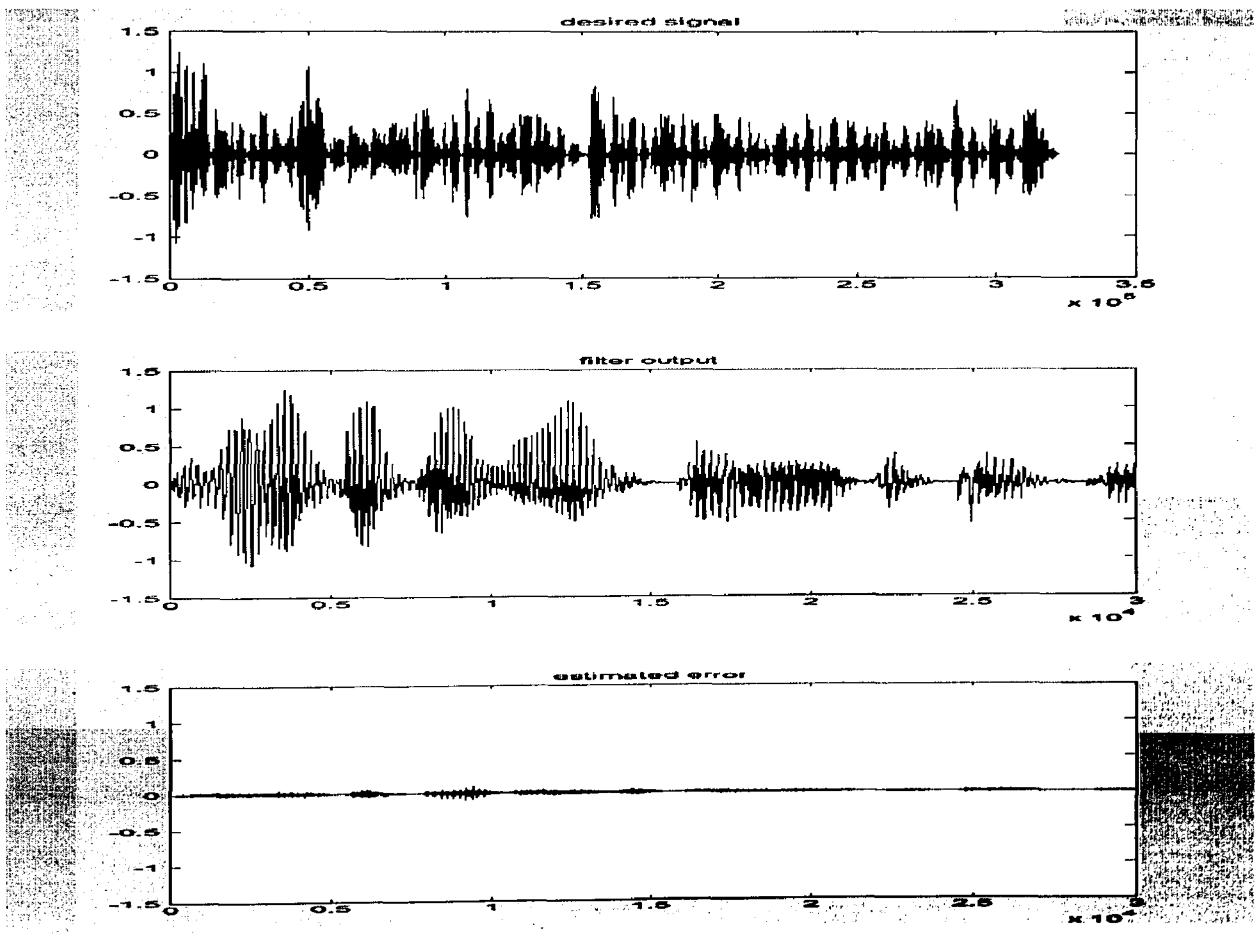


Figure 4.4 LMS FIR filter impulse response

4.1.2 NLMS

The Figure 4.5 shows the desired signal, filter output signal, estimation error and mean square error for the NLMS algorithm with FIR Filter length of 1000.



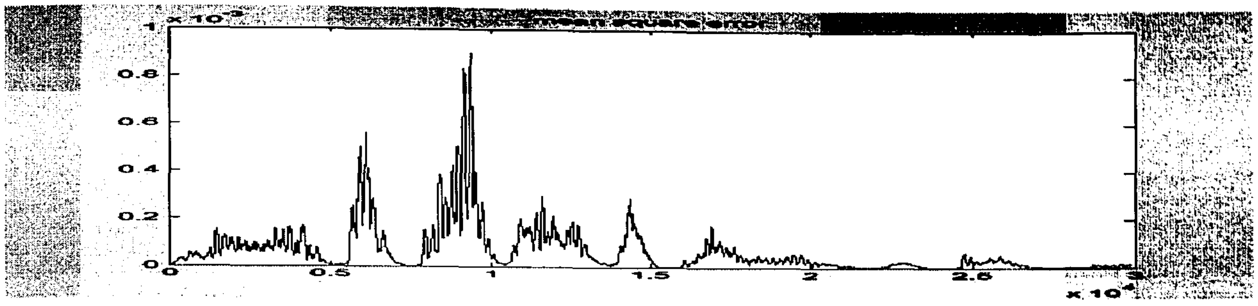


Figure 4.5 NLMS algorithm

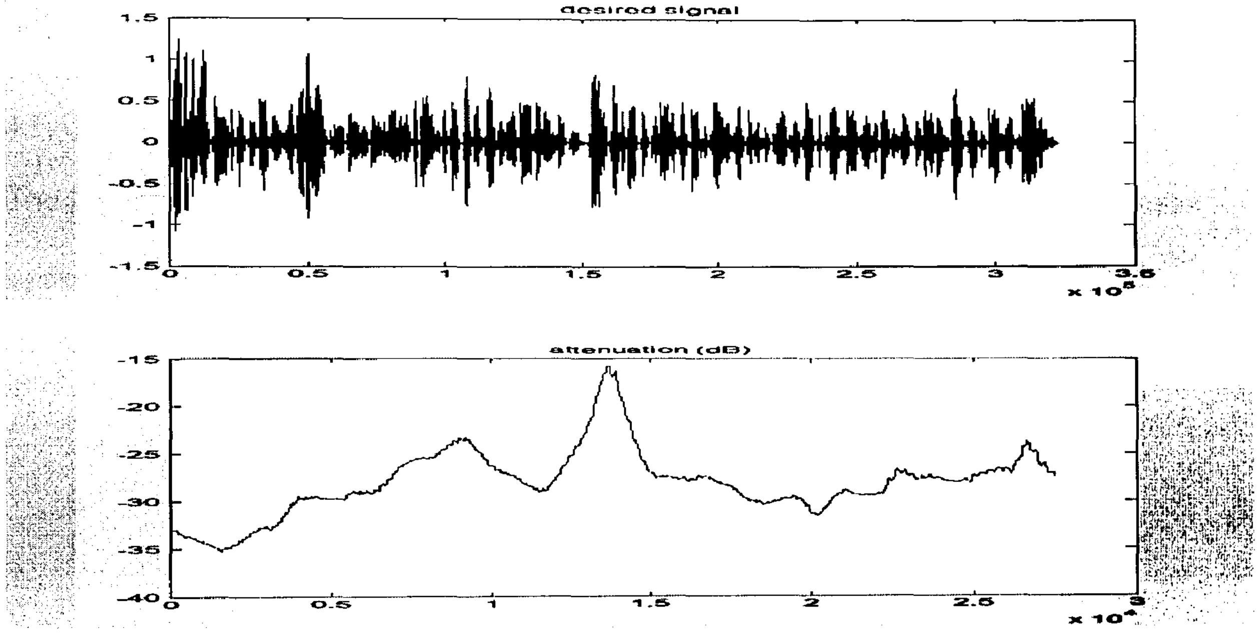
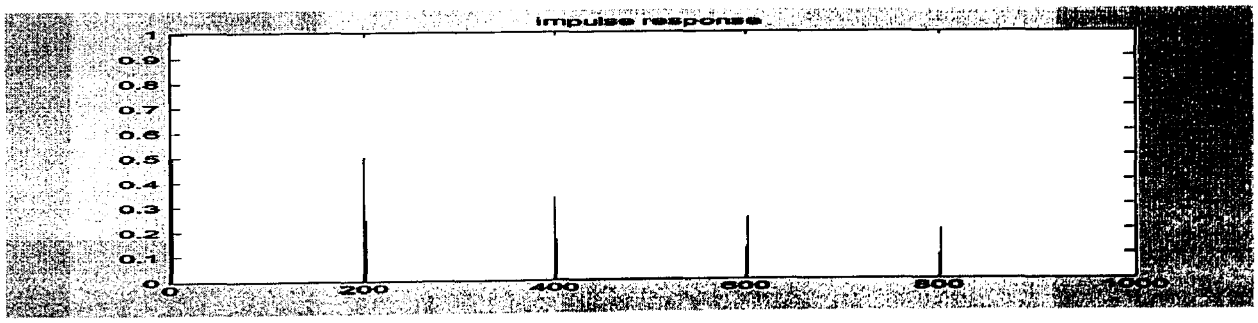


Figure 4.6 NLMS echo signal attenuation

The average attenuation for NLMS FIR filter is –27.99dB



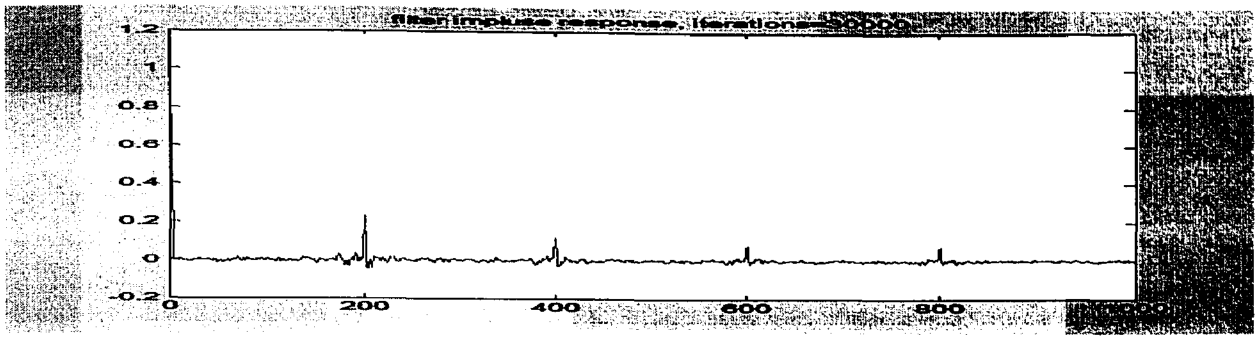
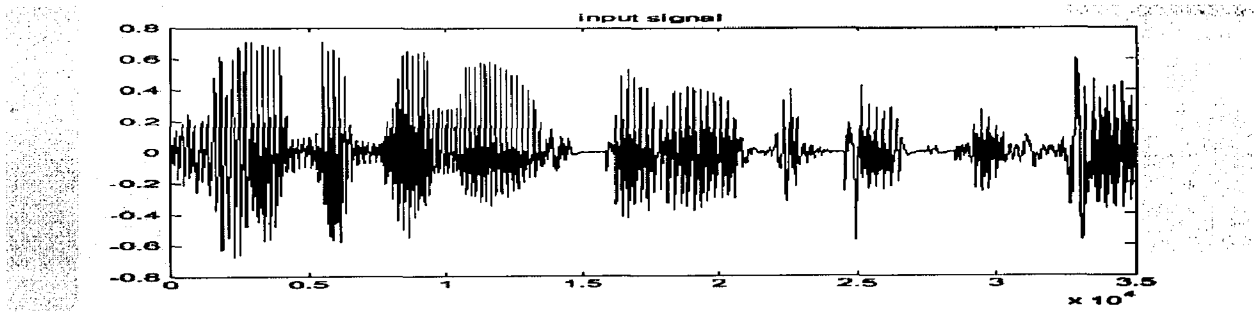
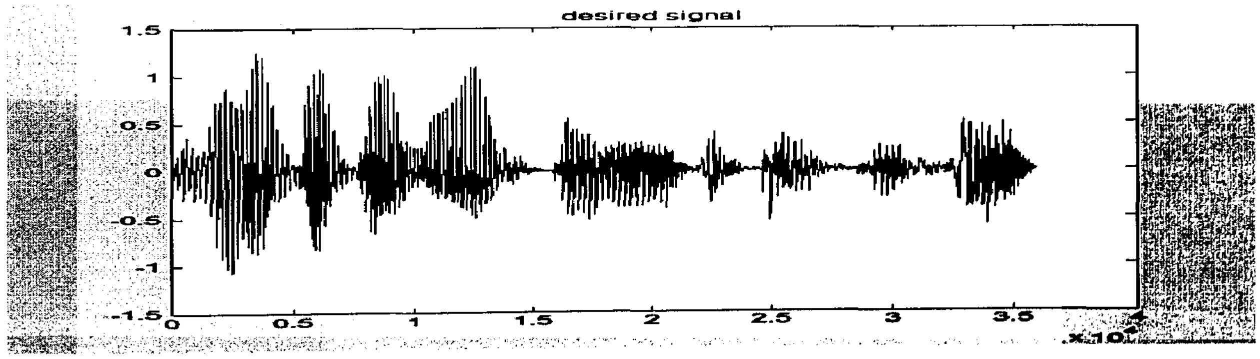


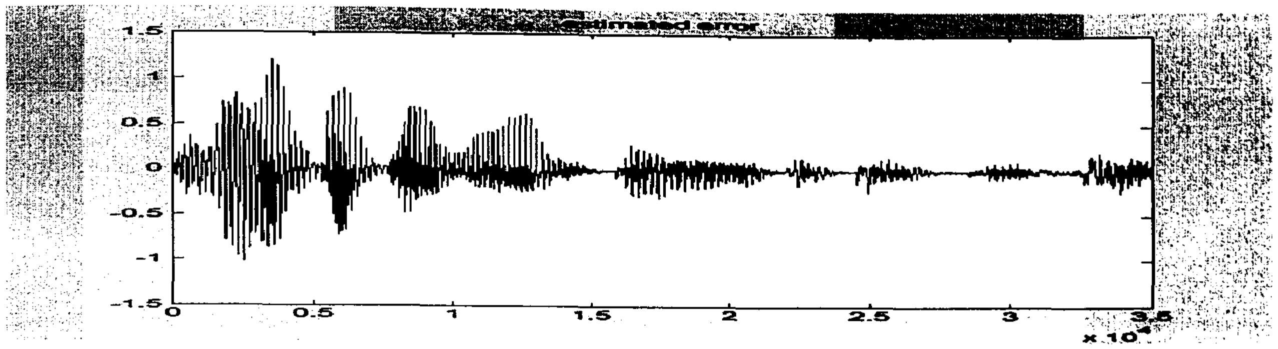
Figure 4.7 NLMS FIR filter impulse response

4.1.3 Fast Block LMS

The Figure 4.8 shows input signal, the desired signal, filter estimation error and mean square error for the FLMS algorithm with FIR Filter length of 1000.







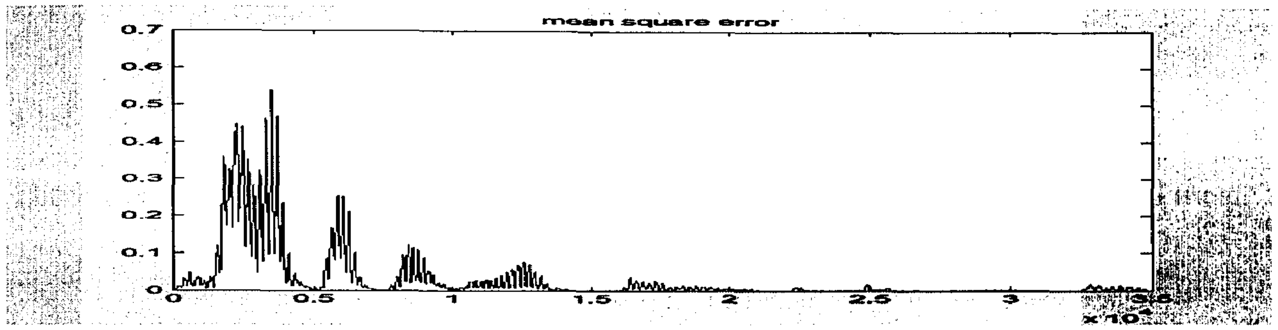
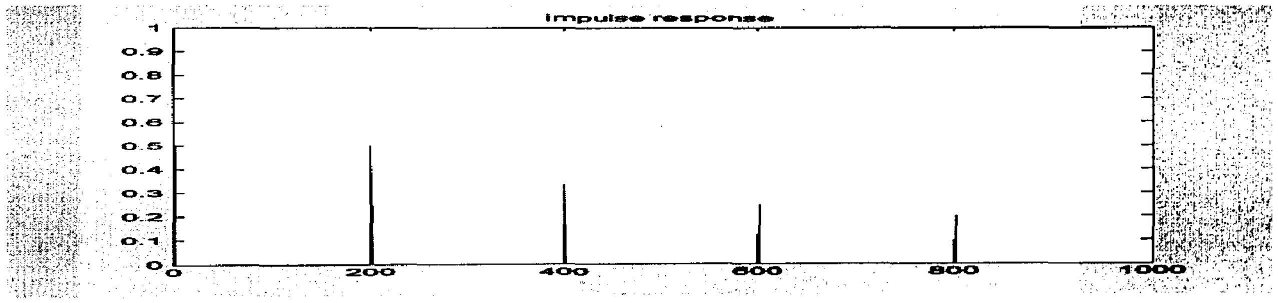


Figure 4.8 Fast Block LMS algorithm



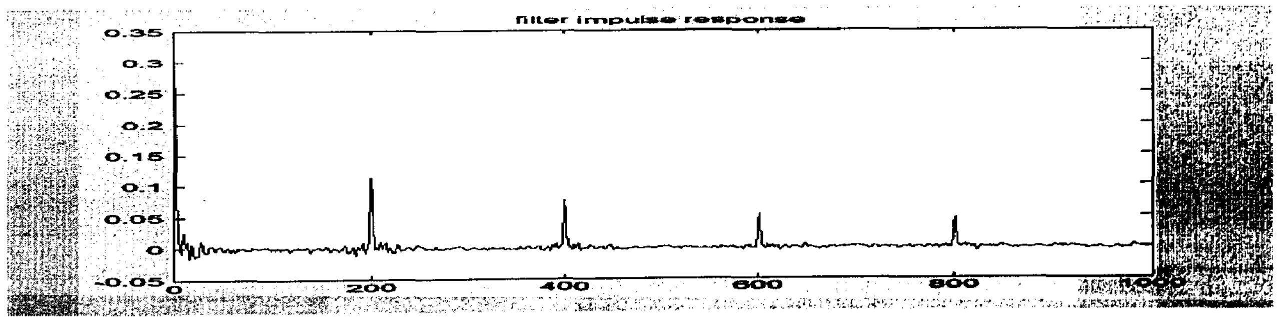


Figure 4.9 Fast Block LMS FIR filter impulse response

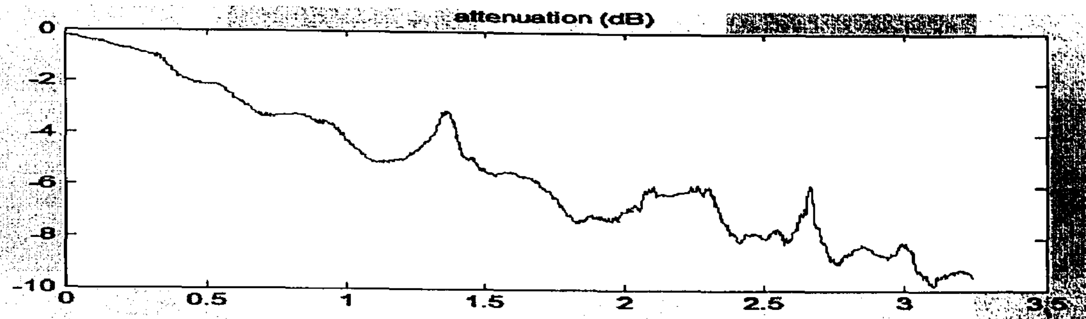
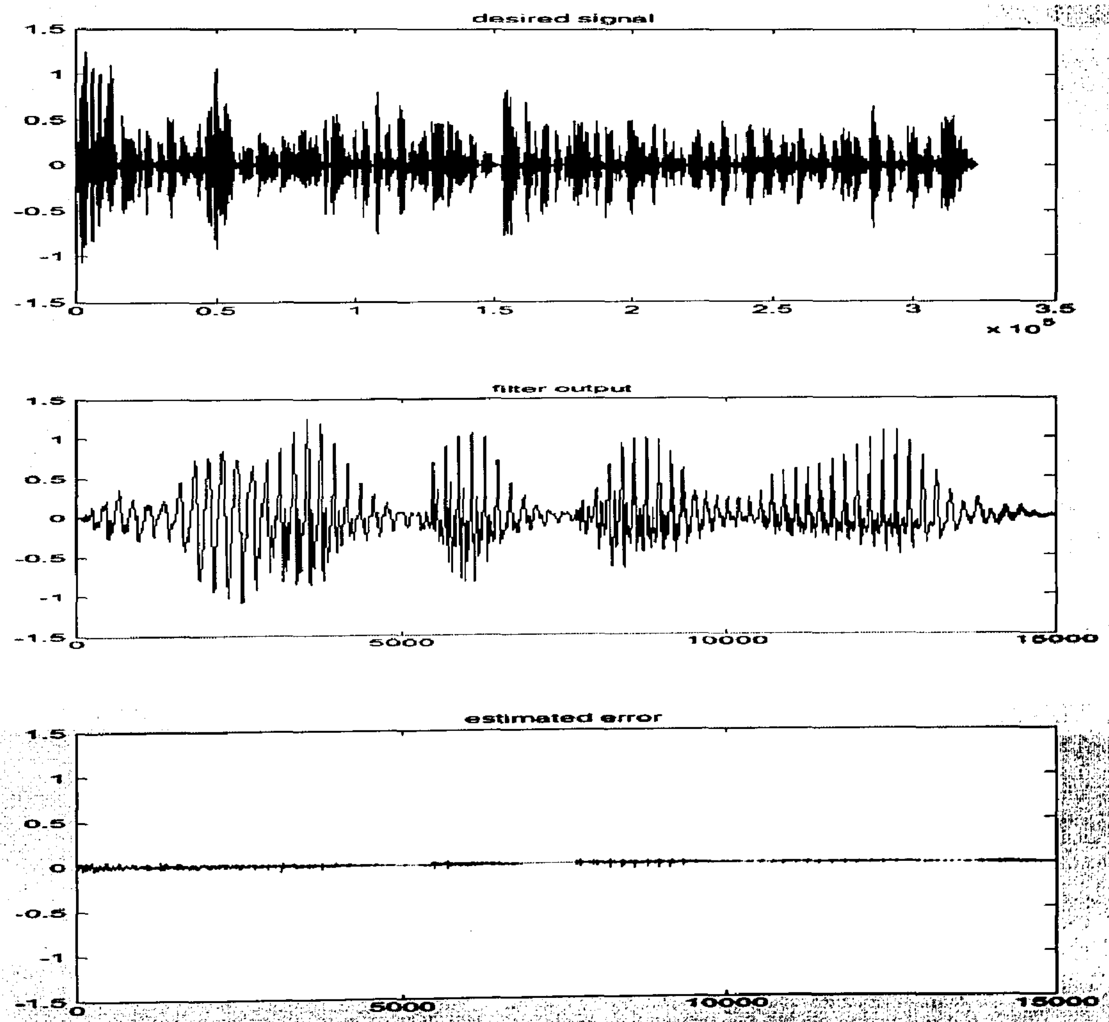


Figure 4.10 Fast Block LMS FIR filter echo signal attenuation

The average attenuation for NLMS FIR filter is -5.33dB.

4.1.4 RLS

The Figure 4.11 shows the desired signal, filter output signal, estimation error and mean square error for the RLS algorithm with FIR Filter length of 1000 and $\lambda = 1$.



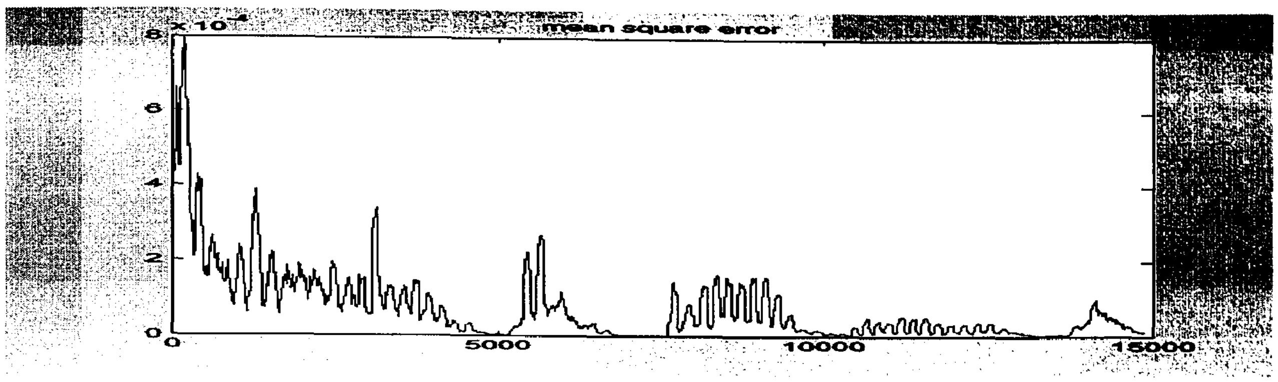


Figure 4.11 RLS algorithm

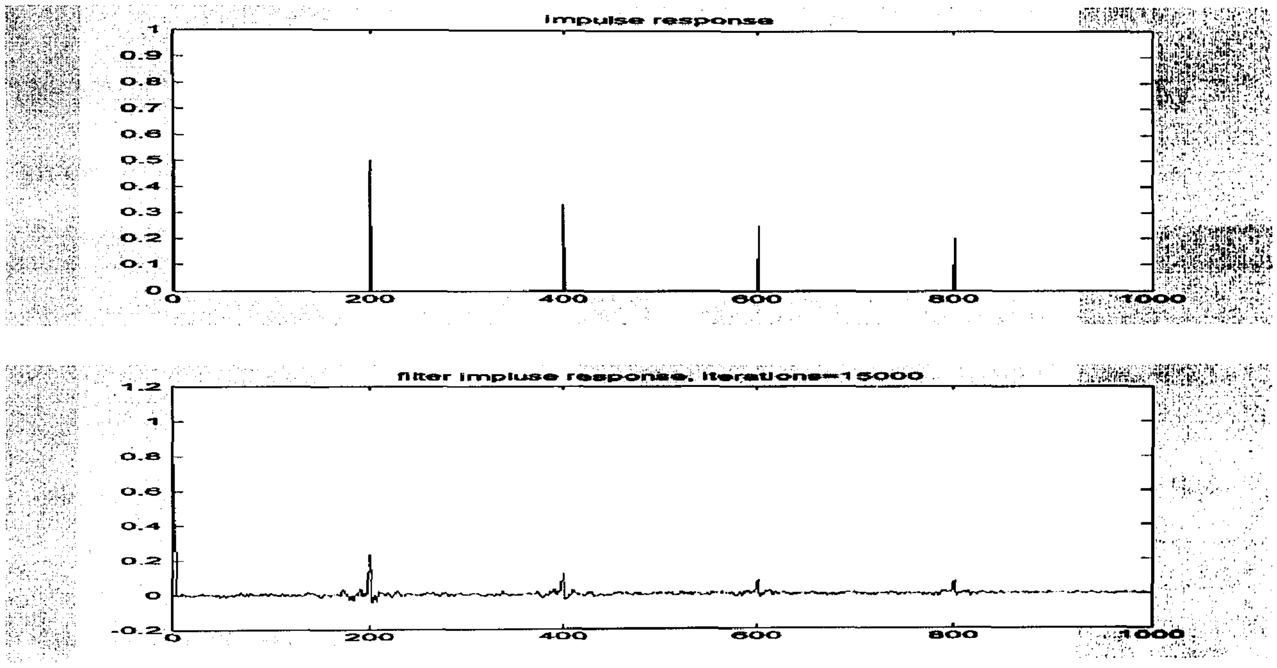
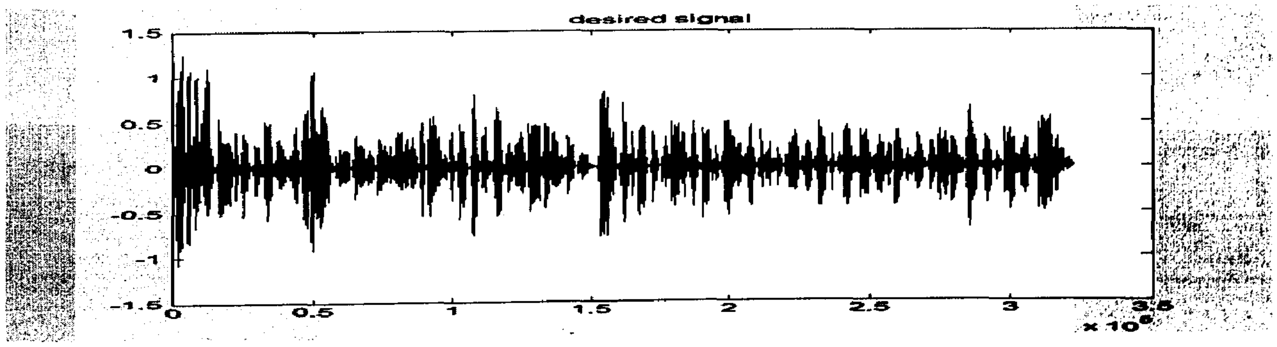


Figure 4.12 RLS FIR filter impulse response



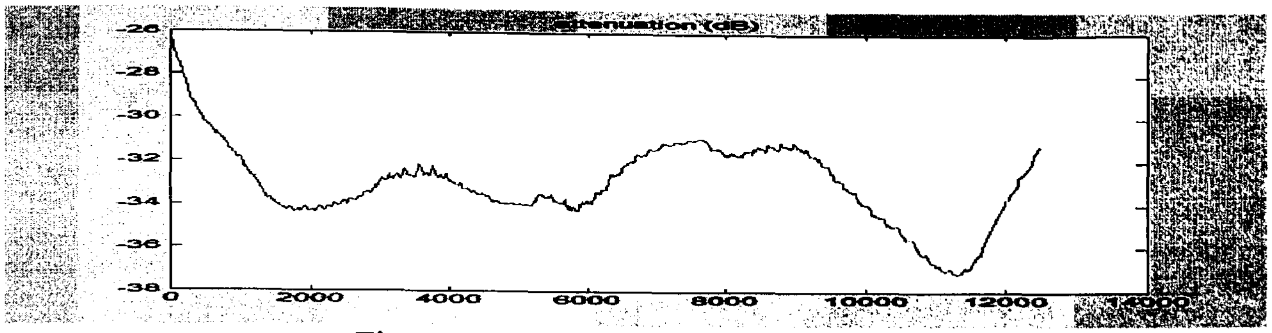
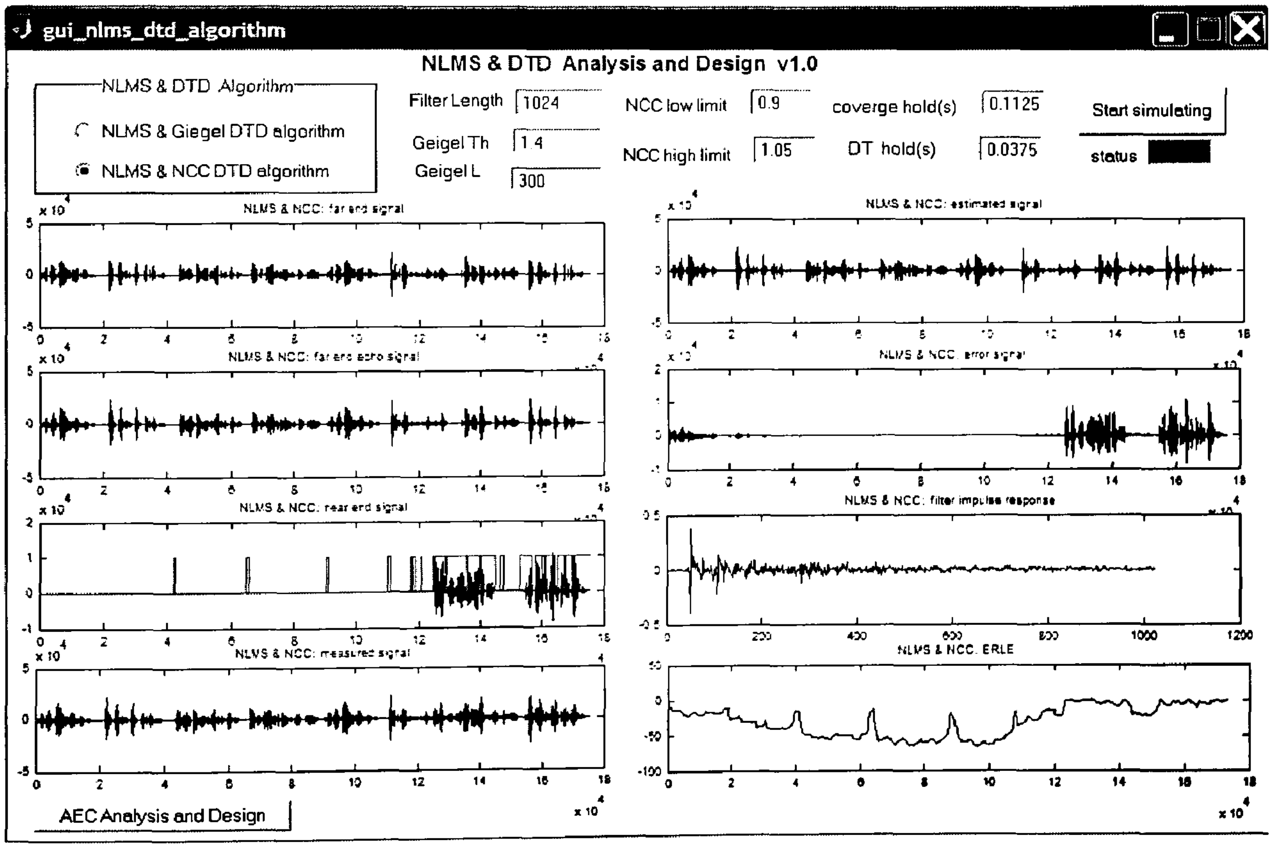


Figure 4.13 RLS echo signal attenuation

The average attenuation for RLS algorithm is –33dB.

4.2 NLMS and DTD Algorithms

The NLMS & DTD Analysis and Design program is another GUI program made by 'guide' to study NLMS and DTD performance, and shown in Figure 4.14.


gui_nlms_dtd_algorithm

NLMS & DTD Analysis and Design v1.0

NLMS & DTD Algorithm

NLMS & Giegel DTD algorithm [✓ NLMS & NCC DTD algorithm]

Filter Length 1024
Geigel Th 1.4
Geigel L 300
NCC low limit 0.9
NCC high limit 1.05
coverge hold(s) 0.1125
DT hold(s) 0.0375

Start simulating
status
NLMS & NCC: far end signal
NLMS & NCC: far end echo signal
NLMS & NCC: near end signal
NLMS & NCC: measured signal
NLMS & NCC: estimated signal
NLMS & NCC: error signal
NLMS & NCC: filter impulse response
NLMS & NCC: ERLE
x 10^4
AEC Analysis and Design

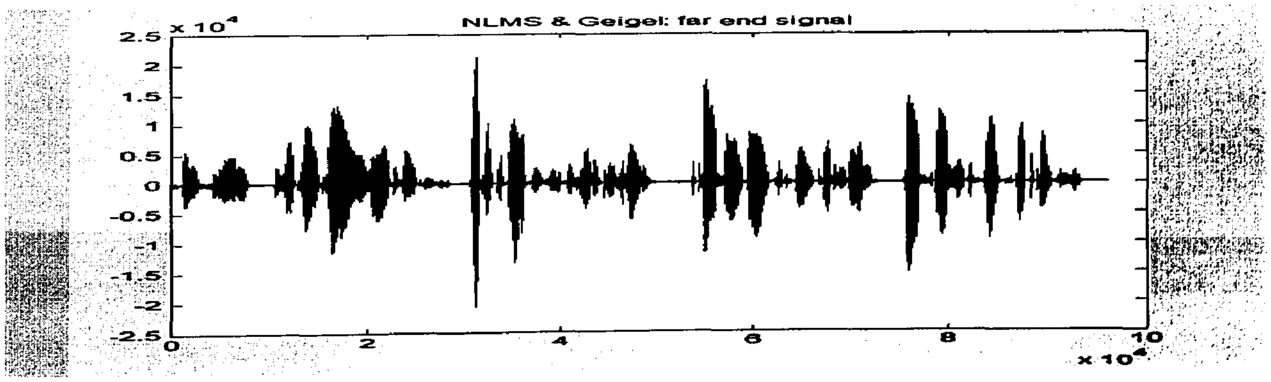
Figure 4.14 NLMS & DTD Analysis and Design program GUI

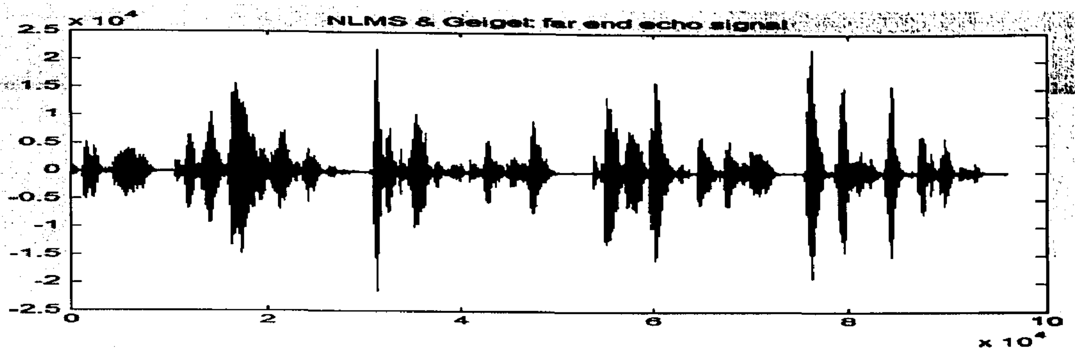
In this program, user can choose:

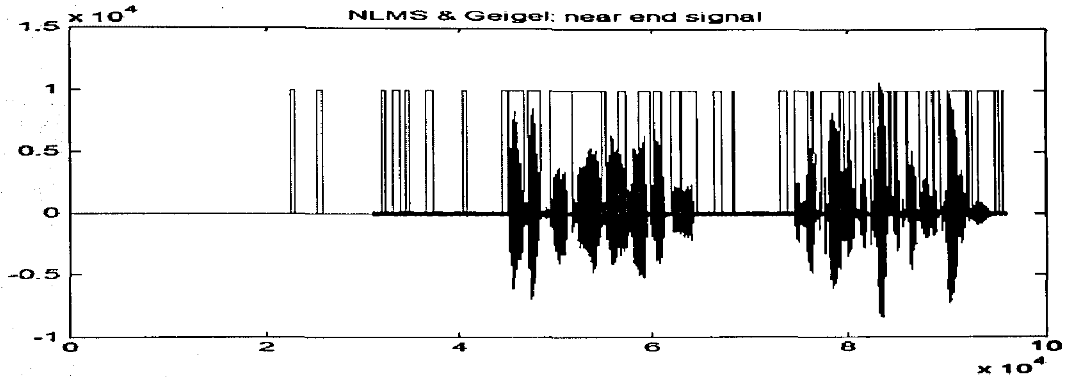
1. NLMS & DTD Algorithm
    a. NLMS and Giegel DTD algorithm
    b. NLMS and normalized cross-correlation (NCC) DTD algorithm
2. Filter Length
3. Geigel Threshold
4. Geigel Length
5. NCC low limit
6. NCC high limit
7. Converge hold time
8. Double Talk period hold time

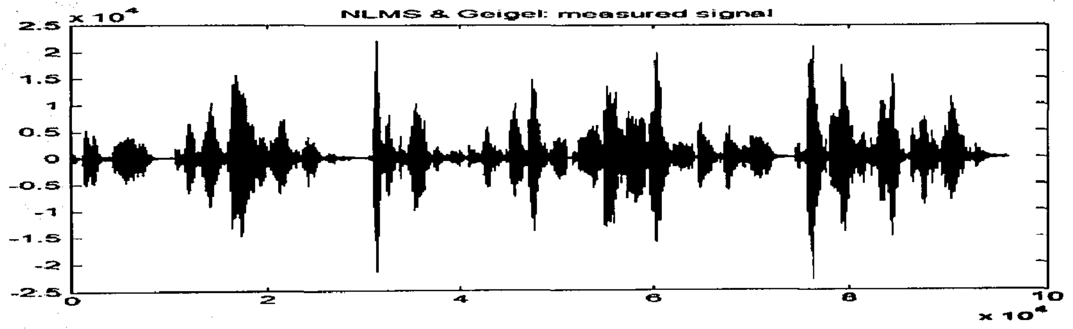
4.2.1 The Geigel DTD algorithm

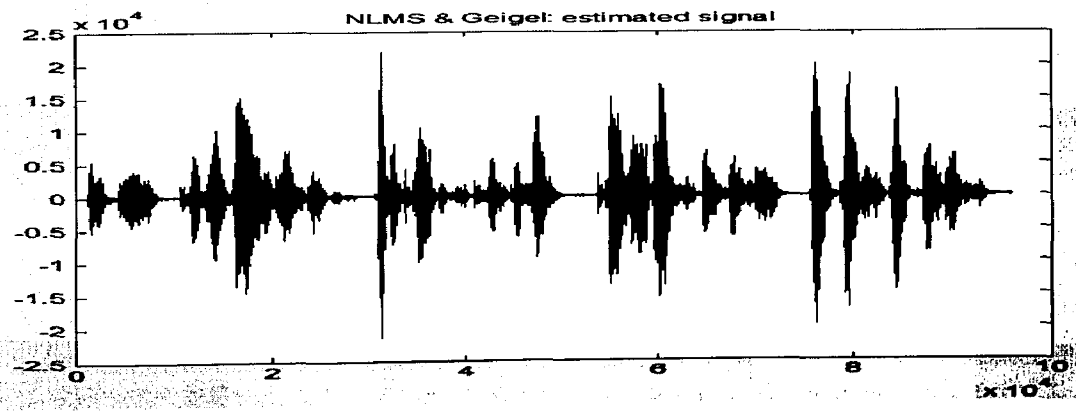
The Figure 4.15 shows the far end signal, far end echo signal, near end signal with DTD detection, received signal, error signal, filter impulse response, and ERLE for NLMS & Geigel DTD algorithm.




NLMS & Geigel: far end echo signal
x 10^4
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
2
4
6
8
10
x 10^4


NLMS & Geigel: near end signal
x 10^4
1.5
1
0.5
0
-0.5
-1
0
2
4
6
8
10
x 10^4


NLMS & Geigel: measured signal
x 10^4
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
2
4
6
8
10
x 10^4


NLMS & Geigel: estimated signal
x 10^4
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
2
4
6
8
10
x 10^4

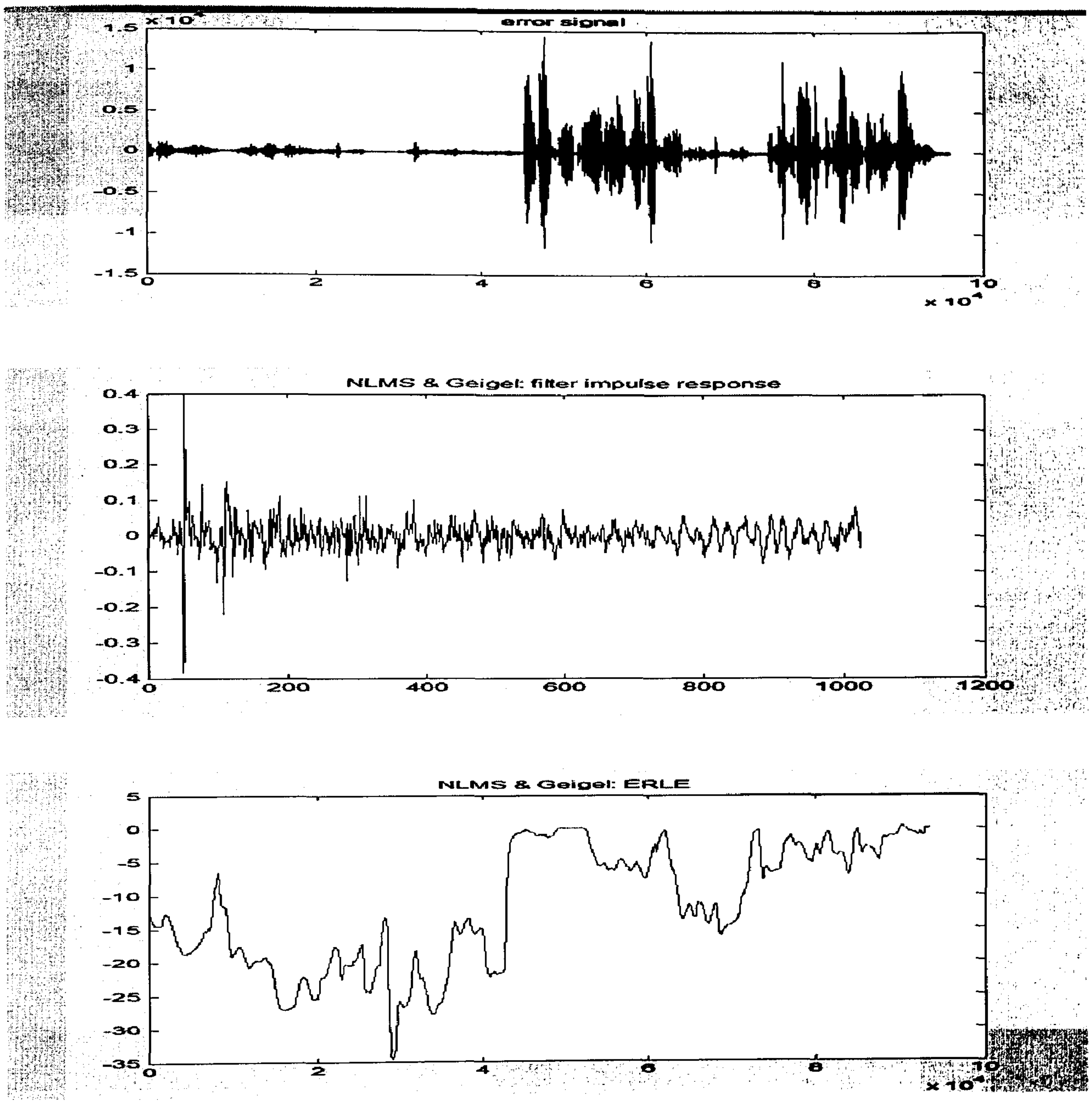


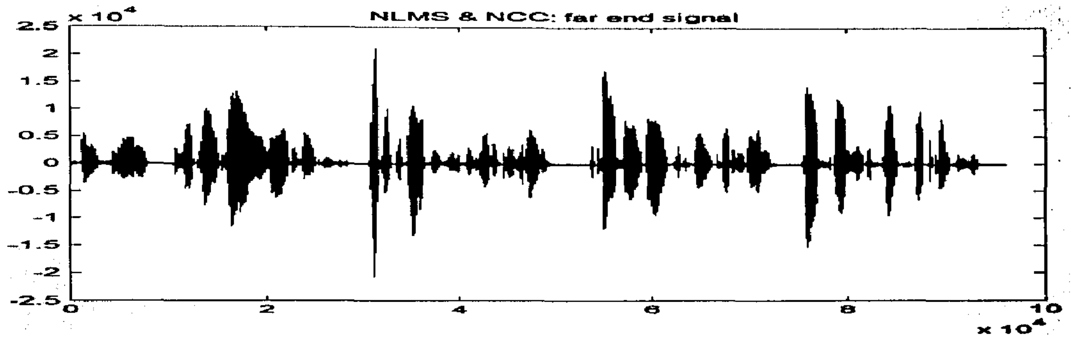
Figure 4.15 NLMS & Geigel DTD algorithm

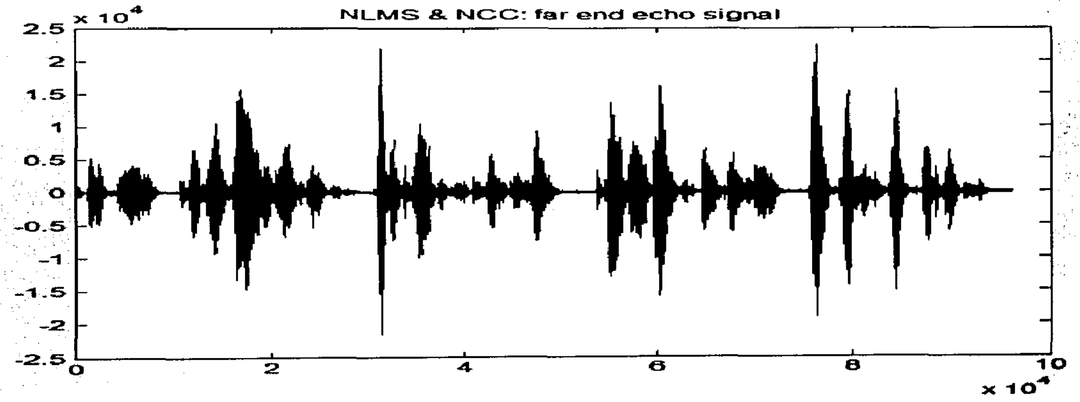
The Geigel DTD miss alarm rate is *Pm*=1- 21715 / 33368=35%. and its false alarm rate is

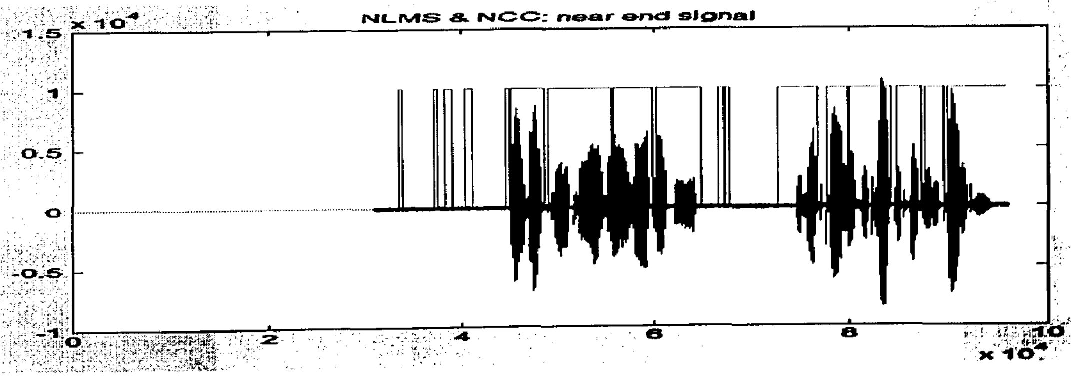
Pf= 15559/96009=16%.

4.2.2 NLMS and normalized cross-correlation DTD algorithm

The Figure 4.16 shows the far end signal, far end echo signal, near end signal with DTD detection, received signal, error signal, filter impulse response, and ERLE for . NLMS and The normalized cross-correlation DTD algorithm.







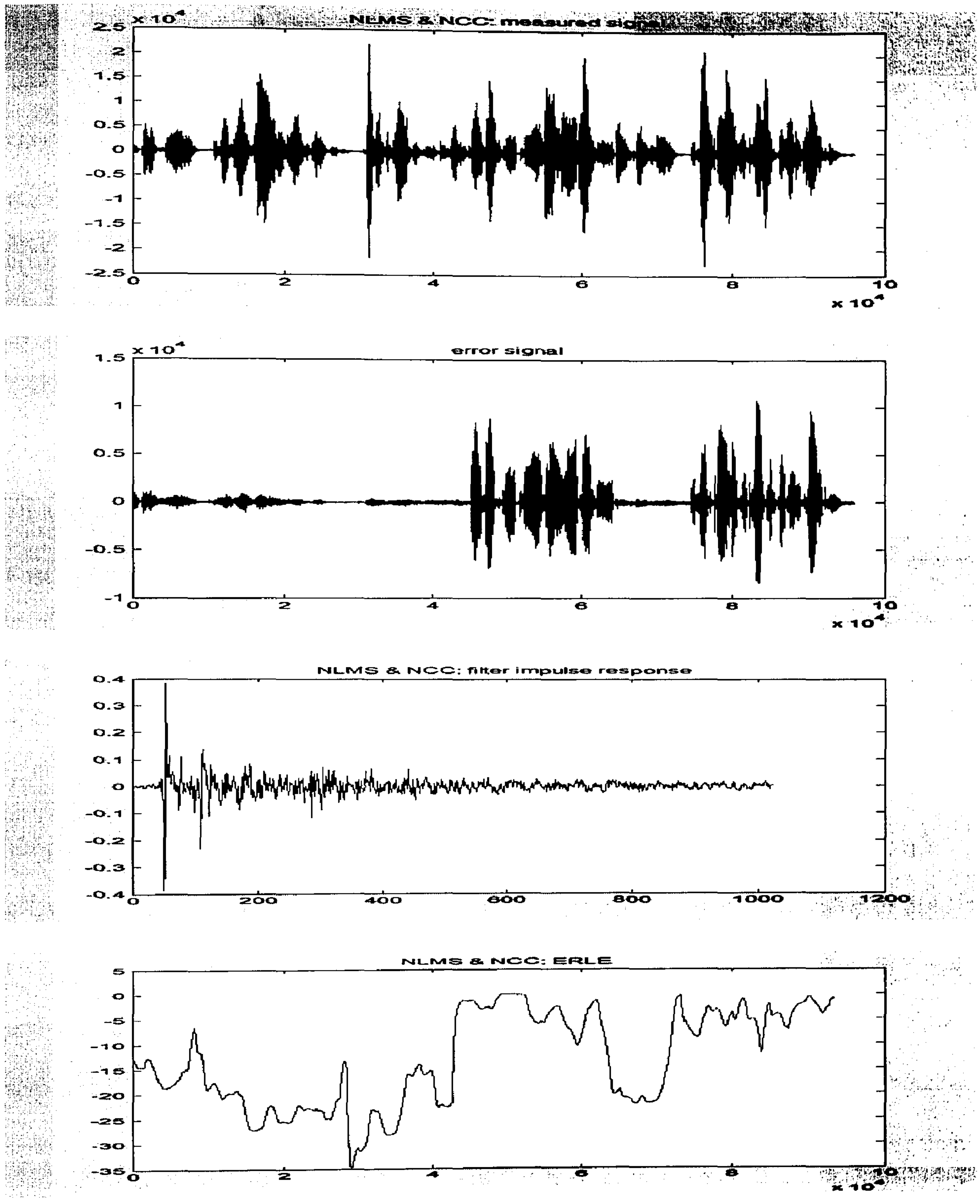


Figure 4.16 NLMS and the normalized cross-correlation DTD algorithm

The miss alarm rate is *Pm*=1-30118/ 33368=0.1=10% and the false alarm rate is *Pf*= 2432/96009=2.5%.

4.3 Matlab Simulation Summary

The **ERLE** (Echo Return Loss Enhancement) is used o assess the quality of an echo cancellation filter. ERLE, a function of the discrete-time index n, is defined as the ratio of the instantaneous power of signal d(n) and the instantaneous power of the residual echo e(n):

$$ERLE(n) = 10 * \log_{10}^{(P_d(n)/P_e(n))}$$

Algorithm	**Average ERLE**	**Multiplication (Filter Length: N)**
LMS	-18.16 dB	2N+1
NLMS	-27.99 dB	3N+1
Fast LMS	-5.3 dB	10 $N\log_2 N$+26N
RLS	-33 dB	4N^2

Table 4.1 Algorithm summary

The LMS algorithm belongs to the class of stochastic gradient algorithms and it's the simplest one. The mean square error of LMS in Figure 4.2 shows that the LMS filter's impulse response converges to the actual impulse response as the average value of the cost function decreases so that the filter could more accurately emulate the desired signal and more effectively cancel the echo signal. LMS is easily to be implemented and if the step size is correctly selected, it is stable. In Figure 4.5, the error signal and mean square error of the Normalized Least Mean Square (NLMS) algorithm is obviously smaller than those of LMS and has faster convergence speed than that of the LMS algorithm. The NLMS differs from the standard LMS algorithm in the sense that it varies the step size according to the power level of the far-end signal. Thus the convergence speed is independent of the input signal power. The average echo signal attenuation of NLMS algorithm is –27.99 dB and is much better than the LMS average echo signal attenuation –18.16 dB. Due to the presence of feedback in the NLMS algorithm, there exists a possibility of it becoming unstable. The stability of the algorithm depends on the step seize parameter β. For the NLMS algorithm, the step size β should satisfy: $0<\beta<2$ and the fastest convergence occurs when $\beta=1$. Since NLMS algorithm is easier to be implemented and the computation is not very intensive, as well as good echo cancellation performance, it is widely used in real time adaptive echo cancellation.

Figure 4.8 shows the Fast LMS error estimation signal is larger, the average echo signal attenuation is only –5.3 dB, but its converge speed is faster and computation complexity is lighter. The RLS algorithm is more effective than all other algorithms. Figure 4.11 shows its error estimation signal is also very small and its mean square error quickly approach to zero and its average echo signal attenuation is –33 dB, but each iteration of the RLS algorithm requires 4N^2 multiplication operations, it is more intensive computation so that in practice it is not popular to be implemented.

From Figure 4.11 and Figure 4.2, we can see that RLS algorithm has a much better convergence rate that the LMS algorithm, but it comes out with more computational complexity. In addition, echo cancellation generally requires large FIR order and thus the RLS algorithm is not suitable for real time implementation.

Figure 4.15 shows the simulation results of the NLMS FIR filter combined with The Geigel DTD algorithm. When the near-end speaker is silent, the FIR has a good performance to cancel the echo signal and keep the error signal in very low level. When both the near-end and the far-end are active - the Double Talk is occurring, the error signal not only contains the echo estimation error, but also the near-end signal.

At this time, the Geigel DTD algorithm will detect the DT period and not update the filter coefficients to prevent the echo canceller from being disturbed by the near end speaker's signal. From the miss alarm rate and false alarm rate, the Gegel algorithm is not very accurate to detect double talk.

For cross-correlation double talk detection algorithm, at the beginning, the decision variable is simplified as $d(t) = \frac{r_{xz}^T(t)h}{\sigma_z^2(t)} = \frac{\tilde{\sigma}_z^2(t)}{\hat{\sigma}_z^2(t)}$

where $\hat{\sigma}_z^2(t) = \hat{\sigma}_z^2(t-1) + z^2(t) - z^2(t-L)$

$$\tilde{\sigma}_z^2(t) = \tilde{\sigma}_z^2(t-1) + z(t)\hat{y}(t) - z(t-L)\hat{y}(t-L)$$

When double talk is presented, Figure 5.1 shows the filter output error signal is not correct and Figure 5.2 displays the double talk detection is not accurate. Through introducing the smoothed estimate and adjusting the smooth factor that described in 3.2, the better results can be achieved.

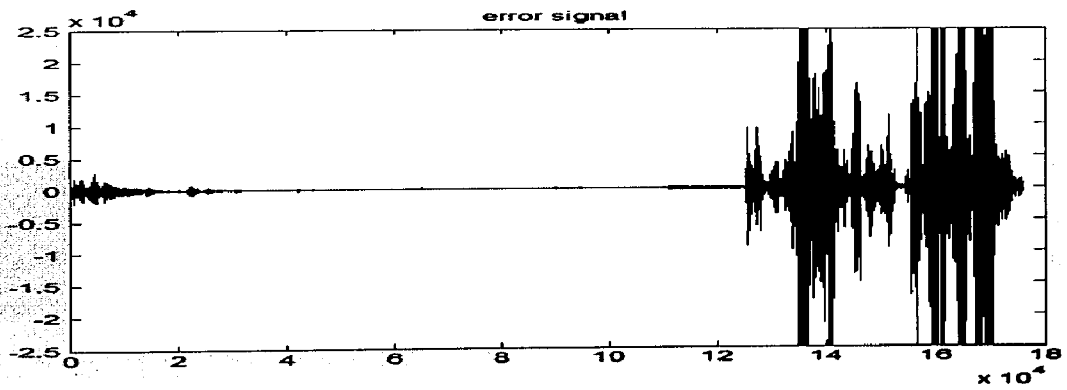


Figure 4.17 Error signal

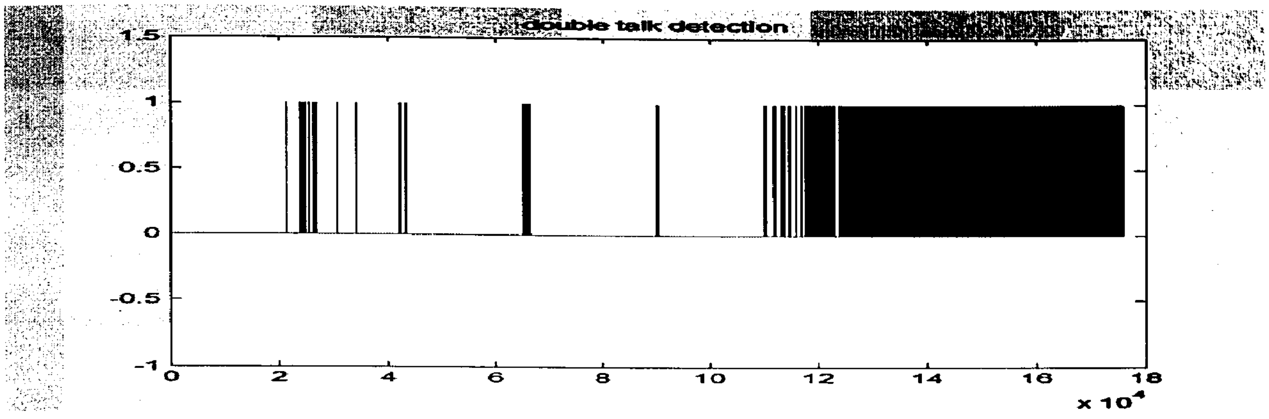


Figure 4.18 Double talk detection

Figure 4.16 shows the simulation results of the NLMS FIR filter combined with the selected normalized cross-correlation DTD algorithm described in 3.2. In no Double Talk period, the filter echo cancellation keep the same as the Geigel DTD algorithm. Figure 4.16 shows the normalized cross-correlation DTD has a better decision in distinguishing DT period. Though it causes more intensive computation, the miss alarm rate is 10% and the false alarm rate 2.5% is much better than Geigel algorithm.

Chapter 5

Adaptive Echo Canceller with DTD Structure

5.1 Adaptive Echo Canceller with DTD

Based on above adaptive algorithm analysis and DTD studies, the NLMS algorithm and normalized cross-correlation DTD algorithm are finally selected in designing the AEC as shown in Figure 6.1.

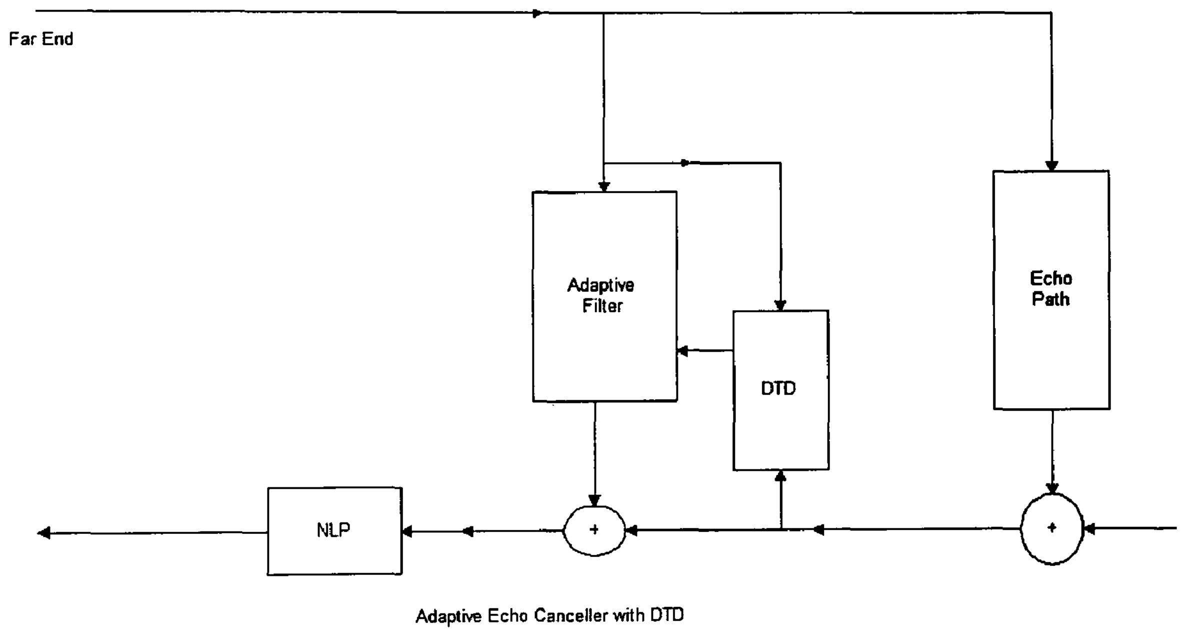


Figure 5.1 Adaptive echo canceller with DTD

The adaptive filter is based on the normalized least mean square (NLMS) algorithm and in the NLMS coefficient update equation (2.2.2),

$$w(m) = w(m-1) + \mu \frac{e(m)}{x(m)_A^T x_A(m)} x_A(m),$$

the variable μ is the step size, which is usually between (0, 2) to maintain the system stable.

The length of the adaptive filter depends on the reverberation time constant of the room. The system is working with sample rate of 8KHZ and the predicted echo path is about 128msec. Thus adaptive filter length will be set as:

Filter length = echo path * sample rate /1000 = 1024.

With the equations (3.2.7), (3.2.8) and (3.2.9) of the normalized cross-correlation DTD algorithm, the smoothing factor α is set to be 0.9 in the equations (3.2.8) and (3.2.9) from experiments. Thus the received decision variable d is a good estimate of the theory value only after the adaptive filter converge, as well as r_{xz} , σ_z become good statistical estimates. Hence the adaptive filter has to converge first to make this DTD algorithm effective. However, it won't cause much problem; because the adaptive algorithm takes relative short time to converge and remain converge most of the time. Initially, the DTD has to be switched off until the adaptive filter first time converges. To detect the convergence of the adaptive filter, we still use the same decision variable d. We know that the decision variable d is about 1 only when the adaptive has converged and there is no double-talk. Hence, the convergence is claimed when the decision variable d approaches 1 and remaining approaching 1 for a certain time. The range of approaching 1 is set to be *(0.9, 1.05)* and the certain time is set to be *0.125* second in practice. In this project, the AEC takes about 25472 samples to converge and the convergence time is calculated as:

Initial Convergence Time= 25472/8000=3.2 sec.

Once the convergence is claimed, we switch on the DTD. After that DTD monitors the decision variable d and it claims double-talk occurs when the decision variable d is out of the range ***(0.75, 1.35)***. Once double talk is declared, the detection is held for a minimum period of time. While the detection is held, the filter adaptation and non-linear processor is frozen. After the hold time, the DTD resumes monitoring again. The hold time is necessary due to the noisy behavior of the detection statistic and is set to 38 ms in implementation.

5.2 Non-linear processor

Since the residual echo is inevitable for the non-linearity of the echo path, the non-linear processor (NLP) is used to degrade the residual echo to an inaudible level. Only during single talk, NLP is active and it is controlled by the DTD. The NLP in this project is set as a controlled attenuator, which attenuates the echo during single-talk by 20dB.

5.3 Software Workflow

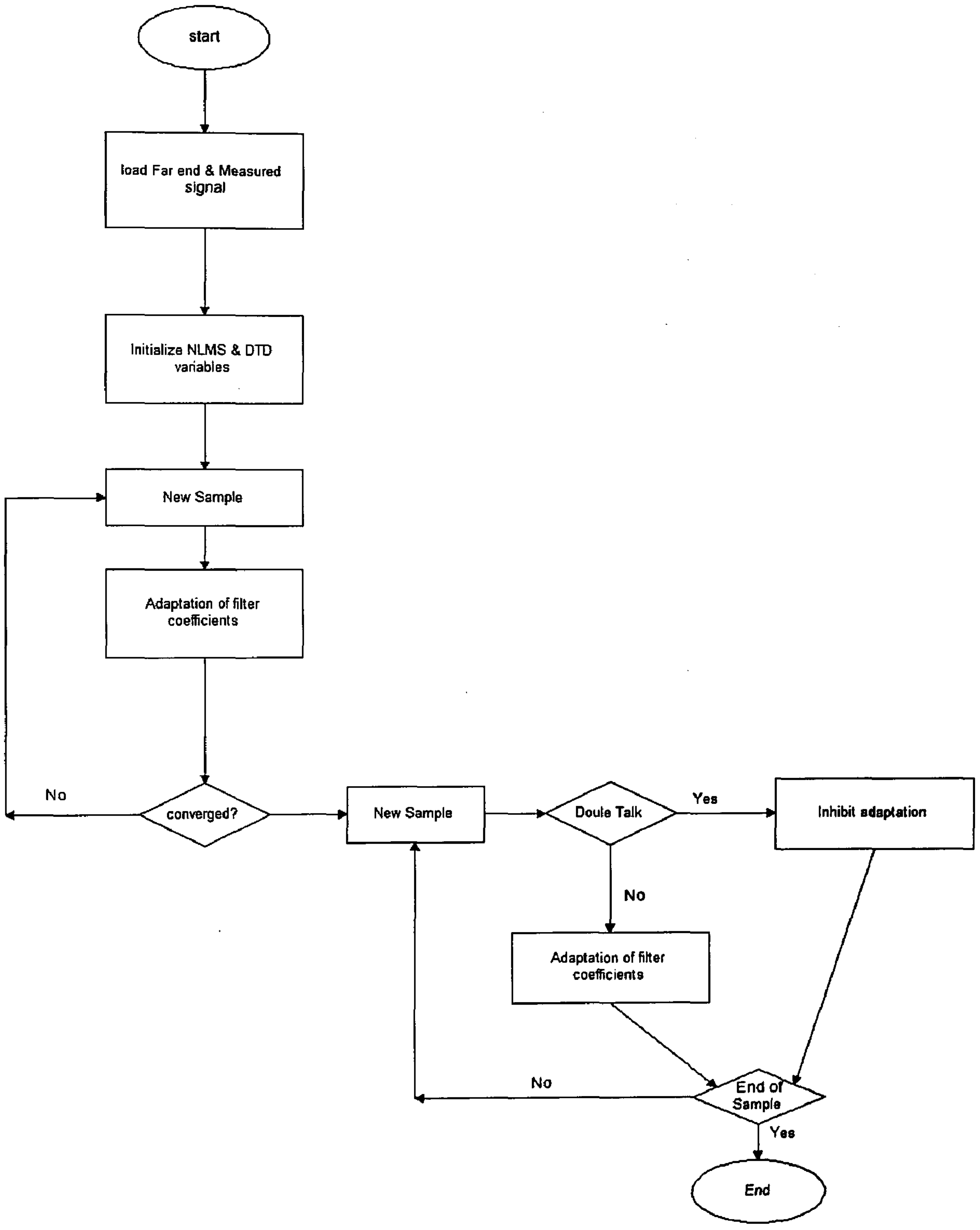

start
load Far end & Measured signal
Initialize NLMS & DTD variables
New Sample
Adaptation of filter coefficients
No
converged?
New Sample
Doule Talk
Yes
Inhibit adaptation
No
Adaptation of filter coefficients
No
End of Sample
Yes
End

Figure 5.2 Adaptive echo canceller with DTD software workflow

Chapter 6

ITU G.168 Test Cases

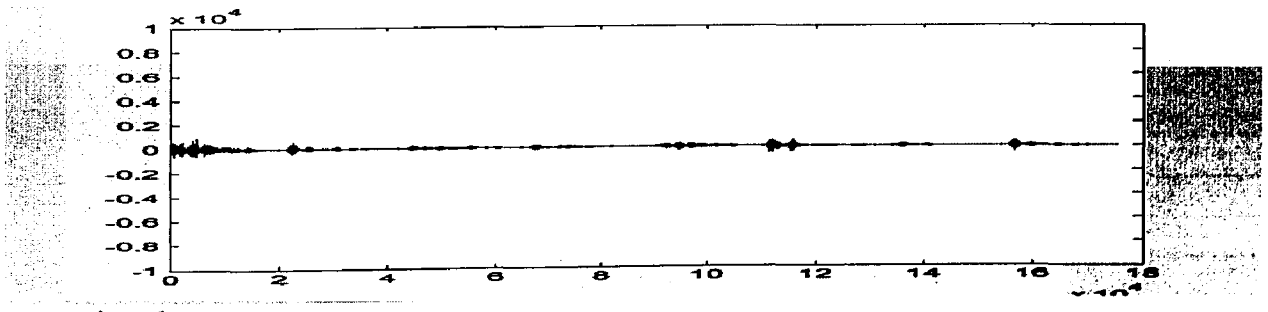
The International Telecommunications Union (ITU), as well as the European Telecommunication Standard Institute (ETSI) regulated the specifications of telephone systems. The ITU-T recommendations: G.167 specifies the performance requirements of acoustic echo control devices. G.165 specifies the requirements of network echo canceller. G.168 is an enhanced version and specifies the new requirements of digital network echo canceller.

6.1 Test 1: Steady state residual and residual echo level test

The G.168 Requirements for Test 1: In the Table 6.1 below, "L(Rin) Input", are the various input levels. "L(Res), NLP Dis, Output Req." are the requirements per G.168 (2000). "Sample" column is for the users to record their test results.

L(Rin) Input	**L(Res), NLP Dis Output Req.**	**Result**	**L(Res), NLP enab Output Req.**	**Result (NLP Enabled)**
-10 dBm0	-37 dBm0	-46.15 dBm0	-65 dBm0	-66.15 dBm0

Table 6.1 Test 1 results



error signal

Figure 6.1 Test 1

6.2 Test 2A and 2B: Convergence test with NLP enabled and disabled

6.2.1 Test 2A: Convergence test with NLP enabled

The G.168 Requirements for Test 2A: the echo path delay is considered as time "zero". For 50 ms beyond the echo path delay, the EC is required to do no better than the minimum ERL of 6 dB. Between 50 ms and 1 sec, the performance of the EC must increase from the minimum of 6 dB to up to 20 dB. Beyond 1 sec, the EC performance is dependent on the input signal level and should be 55 dBm0 or greater for input signal levels of 0 dBm0 and –10 dBm0.

Time (ms)	**L (Rin)-L (Ret) (NLP Enabled)**	**Result (NLP Enabled)**
0	6 dB	48 dB
50ms-1s	6 dB~20 dB	50 dB
1s+	@-10dBm0 55dB	57 dB

Table 6.2 Test 2A results

6.2.1 Test 2B: Convergence test with NLP disabled

The G.168 Requirements for Test 2B: the echo path delay is considered time "zero". From 0 to 50 ms, the EC is not required to perform, therefore the 6 dB minimum requirement (equal to the minimum ERL). Between 50 ms to 1 sec, the EC is required to increase its performance to 20 dB of loss and maintain at least this requirement for up to 10 sec. Beyond 10 sec, the requirement is dependent on signal level.

Time	**L (Rin)-L (Ret) (NLP Disabled)**	**Result (NLP Disabled)**
0	6 dB	8.8 dB
50ms-1s	6 dB~20 dB	9 dB ~20 dB
1s-10s	20 dB	40 dB
10s+	@-10dBm0 27dB	42dB

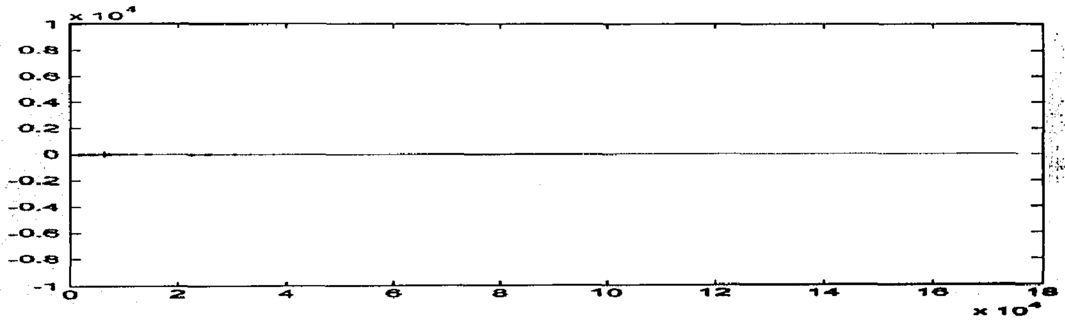
Table 6.3 Test 2B results

6.3 Test 2C: Convergence test in the presence of background noise

The G.168 Requirements for Test 2C Convergence Test with NLP Enabled (part (a)). The columns are interpreted as follows. "L(Rin) Input" is the input signal level. There are three separate files provided for levels –0 dBm0, -10 dBm0, and –20 dBm0. The "Noise Level L(Rin) –15" is the noise level mixed in along with the echo. Note that the maximum level of the noise is -30 dBm0. "L(Ret) Requirement" is the appropriate requirement from G.168 (2000) specification.

L(Rin) Input	**Noise Level L(Rin)-15**	**L(Ret) Requirement**	**Result (NLP Enabled)**
-10 dBm0	-30 dBm0	-30 dBm0	-46 dBm0

Table 6.4 Test 2C(a) results



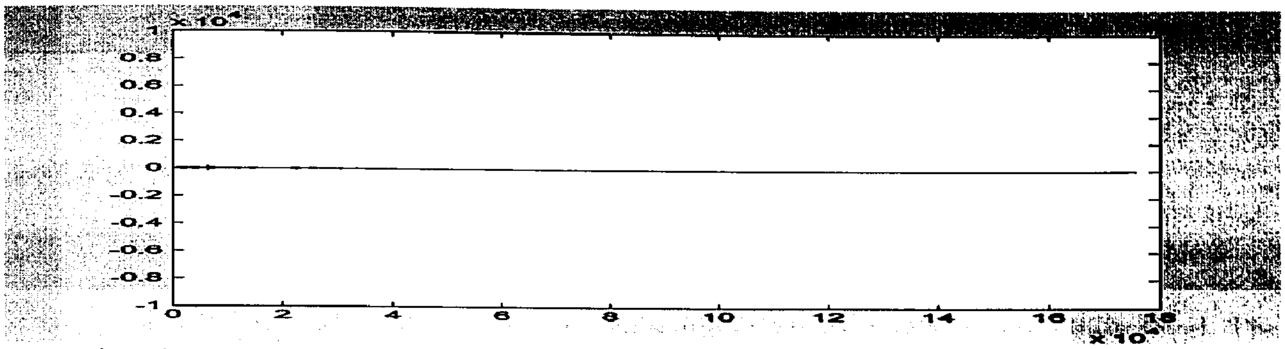
error signal

Figure 6.2 Test 2C(a)

The G.168 Requirements for Test 2C Steady State Test with NLP Enabled (part (b)). The columns are interpreted as follows. "L(Rin) Input" is the input signal level. The "Noise Level –55 dBm0" is the noise level mixed in along with the echo. "L(Ret) Requirement" is the appropriate requirement from G.168 (2000) specification.

L(Rin) Input	**Noise Level**	**L(Ret) Requirement**	**Result (NLP Enabled)**
-10 dBm0	-55 dBm0	-38 dBm0	-46 dBm0

Table 6.5 Test 2C(b) results



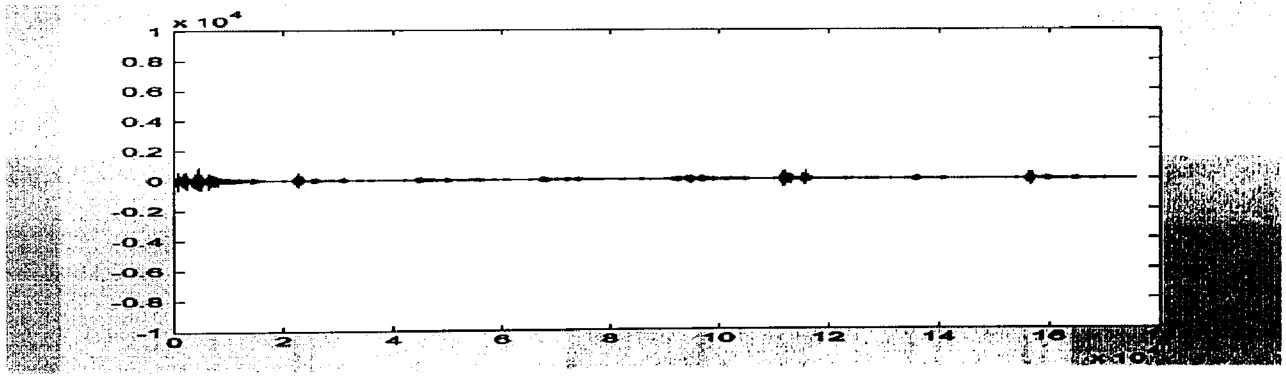
error signal

Figure 6.3 Test 2C(b)

The G.168 Requirements for Test 2C with NLP disabled (part (c)). The columns are interpreted as follows. The input requirements with NLP disabled are similar to those for NLP enabled. The corresponding requirements are given in the column "L(Ret) Requirement".

L(Rin) Input	**Noise Level L(Rin)-15**	**L(Ret) Requirement**	**Sample (NLP Disabled)**
-10 dBm0	-30 dBm0	-30 dBm0	-37 dBm0

Table 6.6 Test 2C(c) results



error signal

Figure 6.4 Test 2C(c)

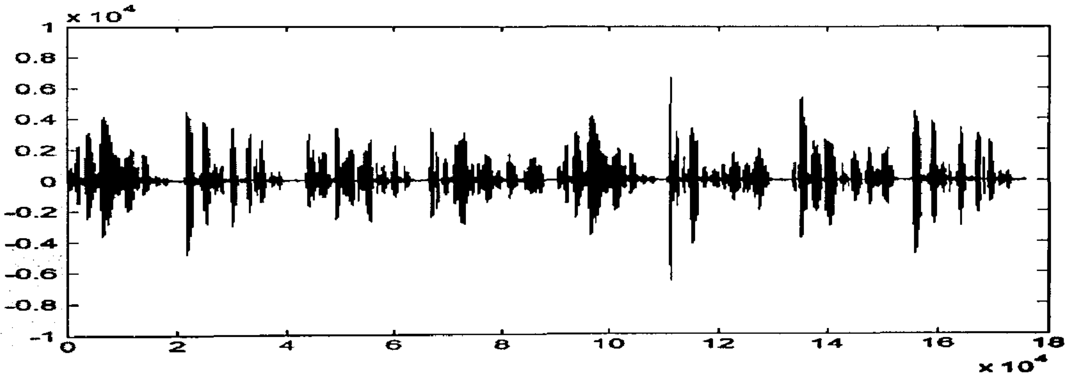
6.4 Test 3: Performance under conditions of double talk

6.4.1 Test 3A: Double talk test with low near end levels

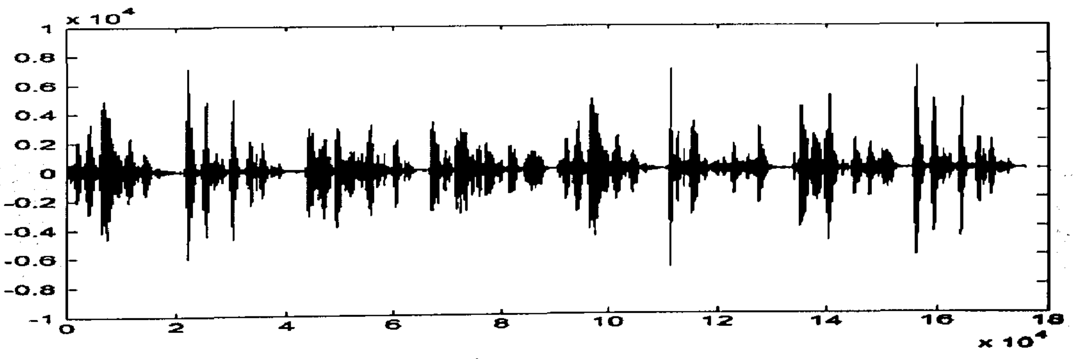
The G.168 Requirements for Test 3A: the requirement as given in the Table 6.7 below is such that the residual echo level should be equal to lower than the doubletalk level. The convergence occurs during the periods when the echo and doubletalk signals do not overlap.

L(Rin) Input	**Low Near-End L(Rin)-15**	**Level L(Res) Requirement**	**Result (NLP Disabled)**
-10 dBm0	-25 dBm0	-25 dBm0	-34 dBm0

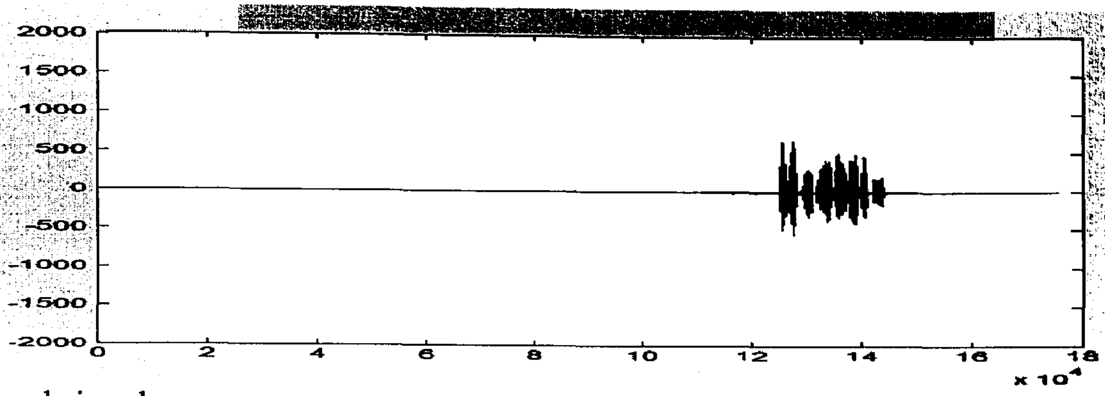
Table 6.7 Test 3A results



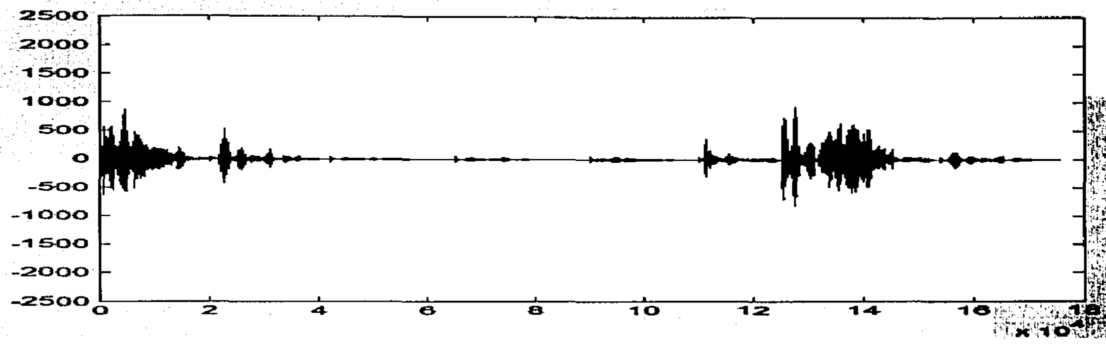
far end signal



fare end echoed signal



near end signal



error signal

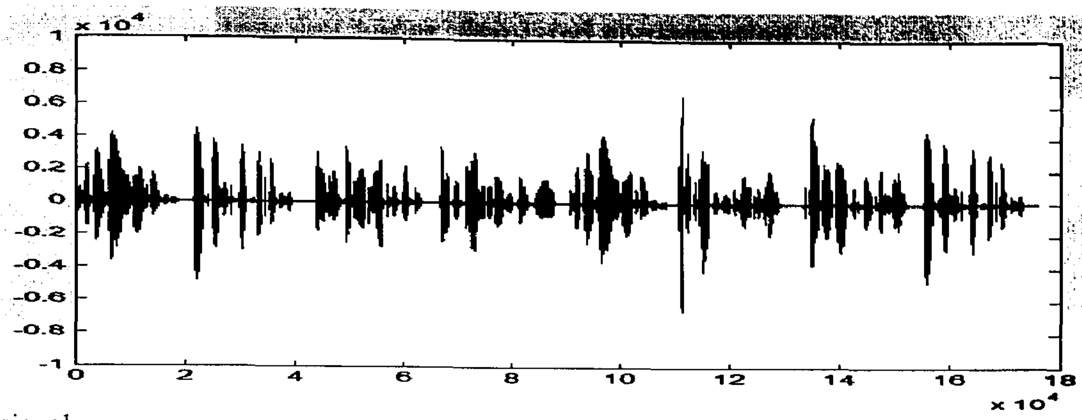
Figure 6.5 Test 3A

6.4.2 Test 3B: Double talk test with high near end levels

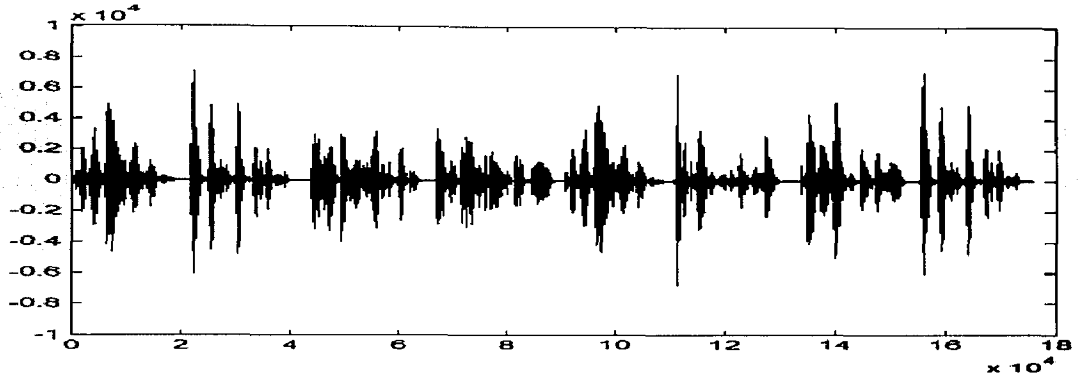
The G.168 Requirements for Test 3B: notice that the doubletalk level should be at least as great as the input level. The residual echo requirement is relaxed by 10 dB from the steady state requirements.

L(Rin) Input	**Doubletalk N≥L(Rin)**	**Level L(Ret) Requirement**	**Result (NLP Disabled)**
-10 dBm0	-10 dBm0	-27 dBm0	-35 dBm0

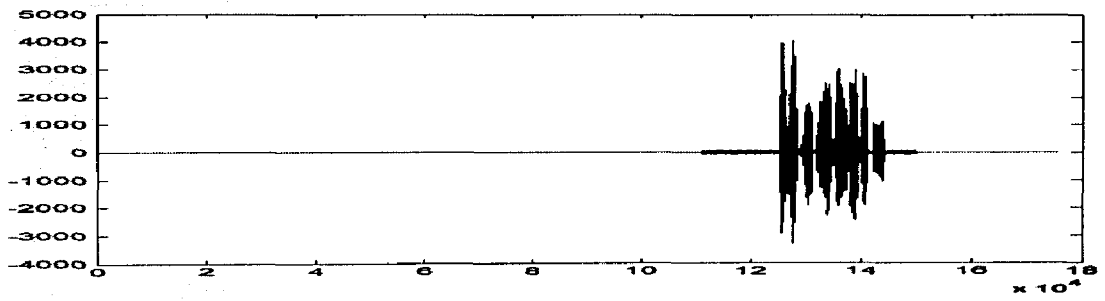
Table 6.8 Test 3B results



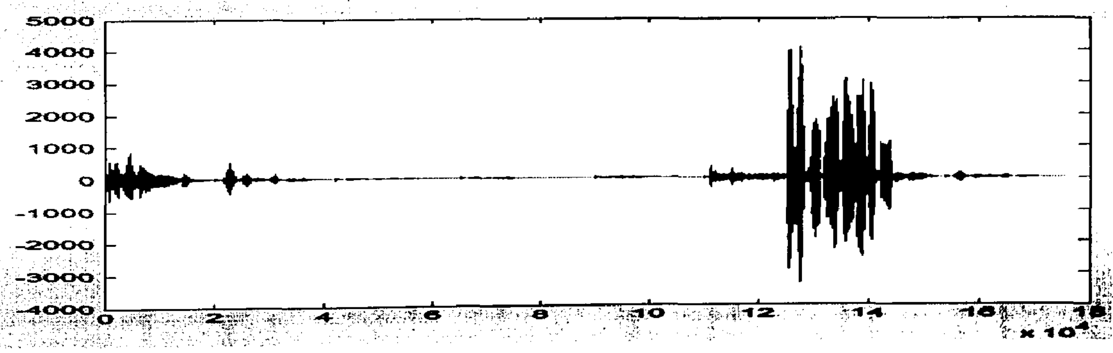
far end signal



far end echoed signal



near end signal



error signal

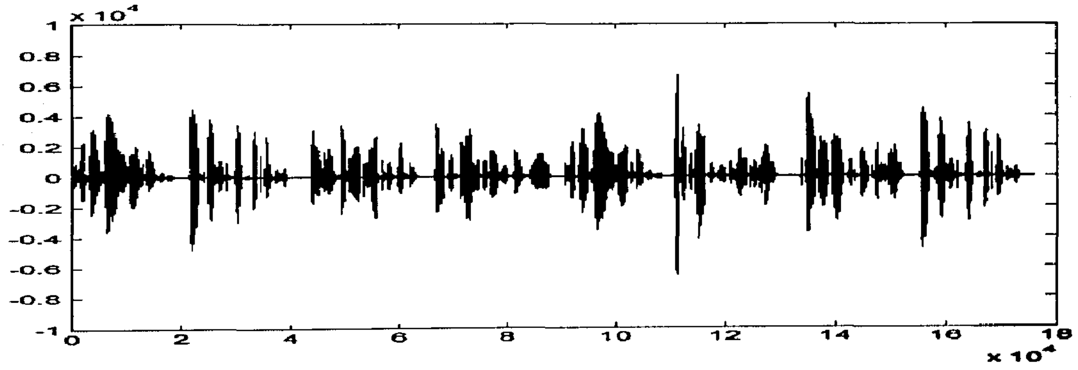
Figure 6.6 Test 3B

6.5 Test 3C: Double talk test under simulated conversation

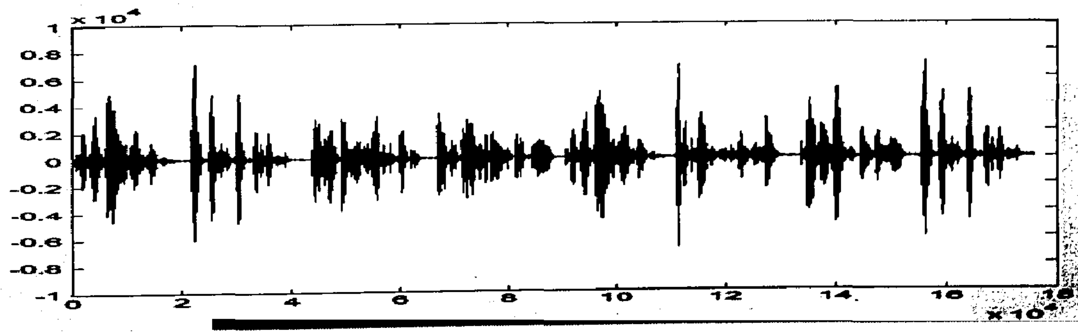
The G.168 Requirements for Test 3C: in the Table 6.9 below, note that the table continues, i.e. there are actually eight (8) columns. The doubletalk signal is applied simultaneously with the input signal. The resultant signal is divided into 5 periods (refer to G.168 (2000) spec) and denoted as t_1, t_2, t_3, t_4, and t_5. The requirements for these different periods are given in the table below.

L(Rin) Input	Doubletalk Level N Performance	t2(Req.)	Result	t3(Req.) Performance	Result	t4(Req.)	Result
-10 dBm0	-10 dBm0	no peaks>N	meets req.	-65 dBm0	-52 dBm0	no peaks>N+6 dBm0	meets req.

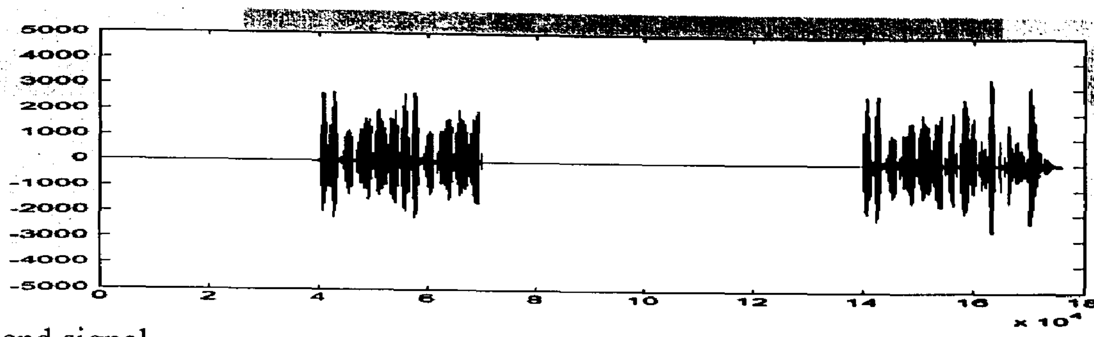
Table 6.9 Test 3C results



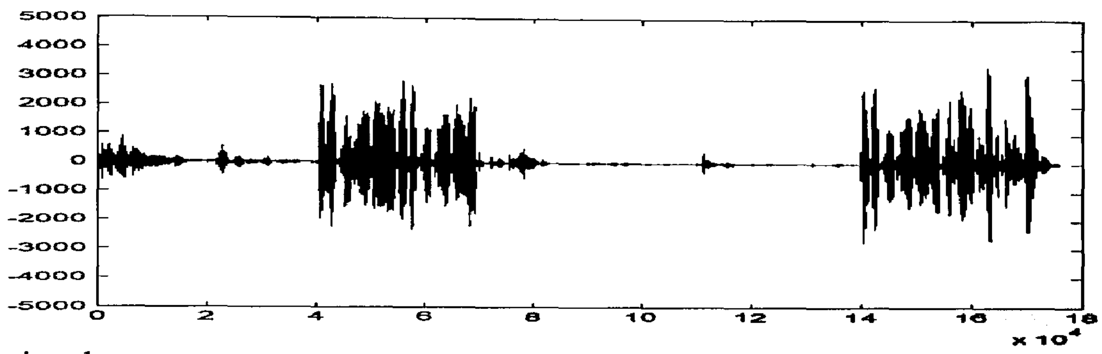
far end signal



far end echoed signal



near end signal



error signal

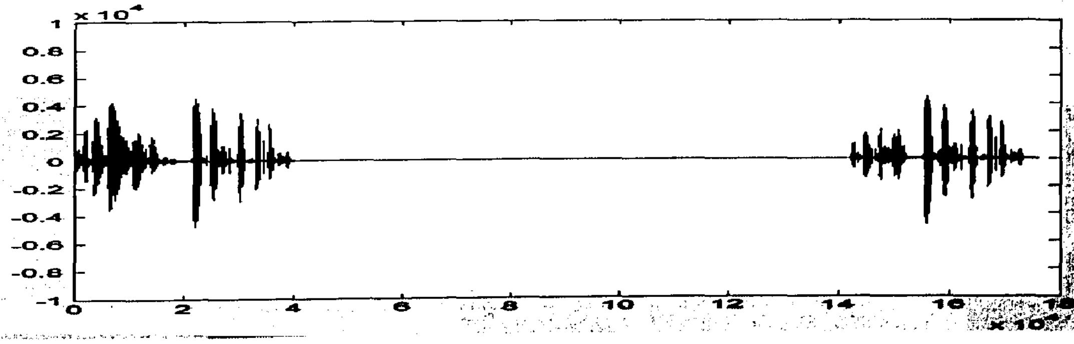
Figure 6.7 Test 3C

6.6 Test 4: Leak rate test

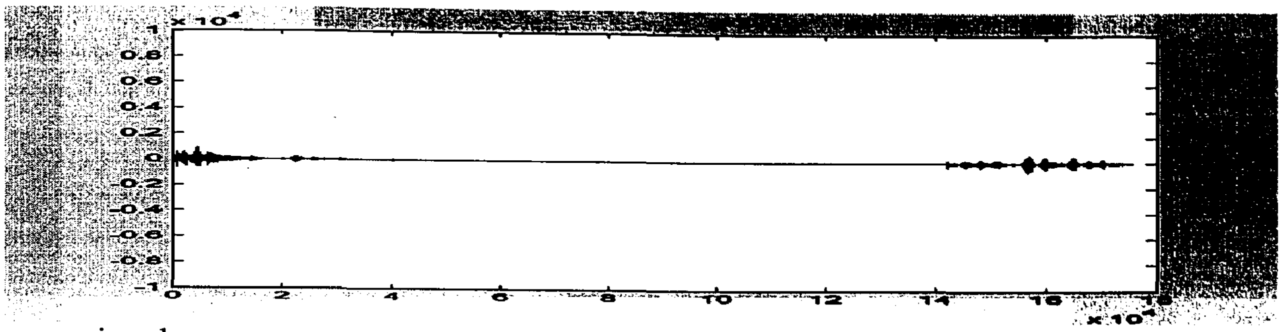
The G.168 Requirements for Test 4: the requirements for the residual echo level are relaxed by 10 dB from the requirements for steady state in Test 1 and 2.

L(Rin) Input	**L(Ret) Requirement (NLP Disabled)**	**Result**
-10 dBm0	-27 dBm0	-25 dBm0

Table 6.10 Test 4 results



far end signal



error signal

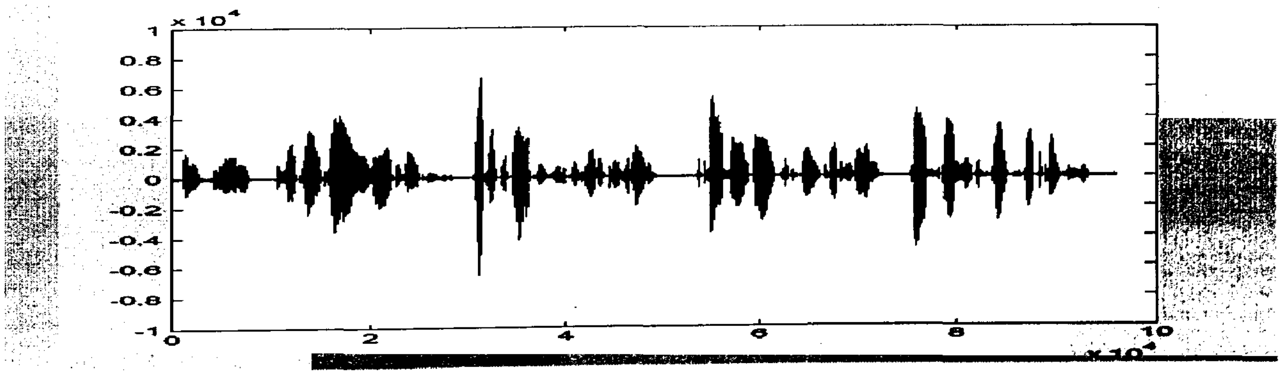
Figure 6.8 Test 4

6.7 Test 5: Infinite return loss convergence test

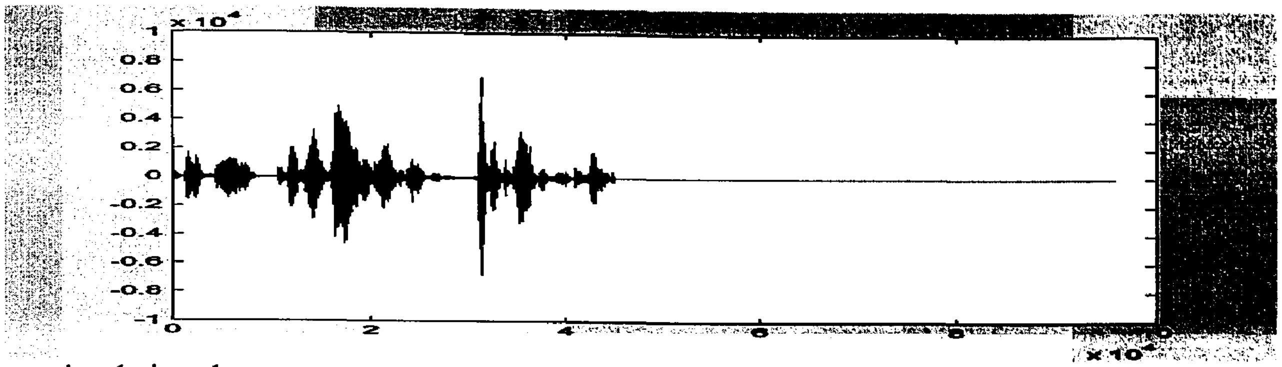
The G.168 Requirements for Test 5: the requirements are such that any spurious response vanishes to less than 20 dB below the input level by 1 sec after the echo path is cut off and to less than 27 or 25 dB below the input level by 10 sec after the echo path is cut off.

L(Rin) Input	**L(Rin) – L(Res) Requirement (NLP Disabled)**	**Sample**
-10 dBm0	>20 dB after 1 sec >27 dB after 10 sec	meet req.

Table 6.11 Test 5 results



far end signal



received signal

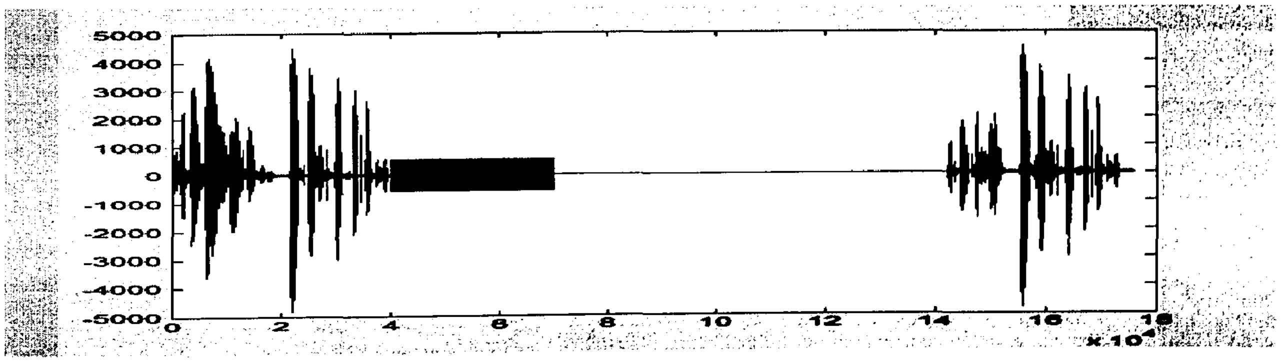
Figure 6.9 Test 5

6.8 Test 6: Non-divergence on narrow-band signals

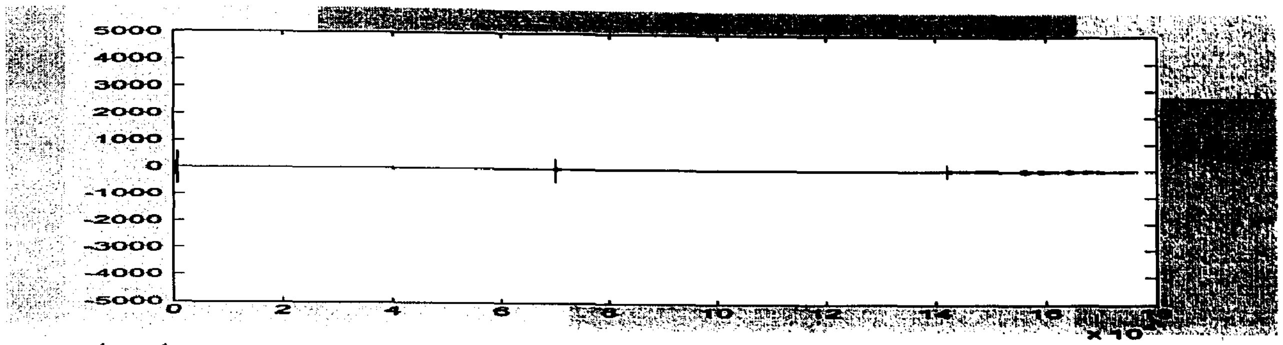
The G.168 Requirements for Test 6: the requirements for this test for the residual echo are relaxed by 10 dB versus the steady state residual echo requirements for Tests 1 and 2.

L(Rin)	**L(Ret)** **Final Requirement**	**Result** **(NLP Disabled)**
-10 dBm0	-27 dBm0	-32 dBm0

Table 6.12 Test 6 results



far end signal



error signal

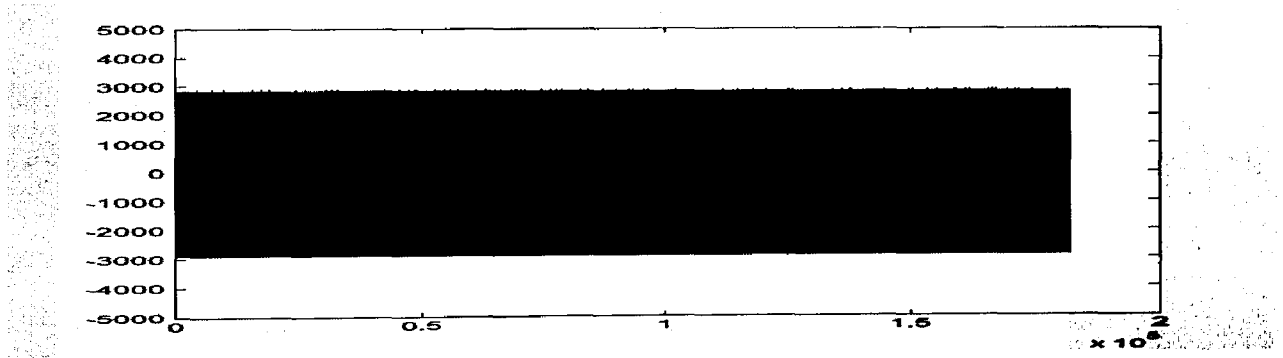
Figure 6.10 Test 6

6.9 Test 7: Stability test

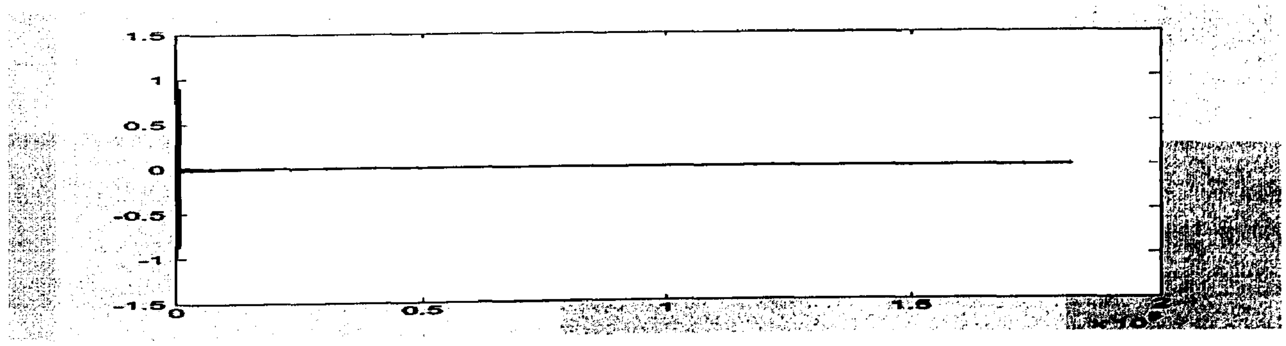
The G.168 Requirements for Test 7: note that the requirements for this test are actually more stringent than that for steady state.

L(Rin) Input	**L(Res), NLP dis Output Requirement**	**Result (NLP Disabled)**
-10 dBm0 (1 kHz)	-38 dBm0	-48 dBm0

Table 6.13 Test 7 results



far end signal



error signal

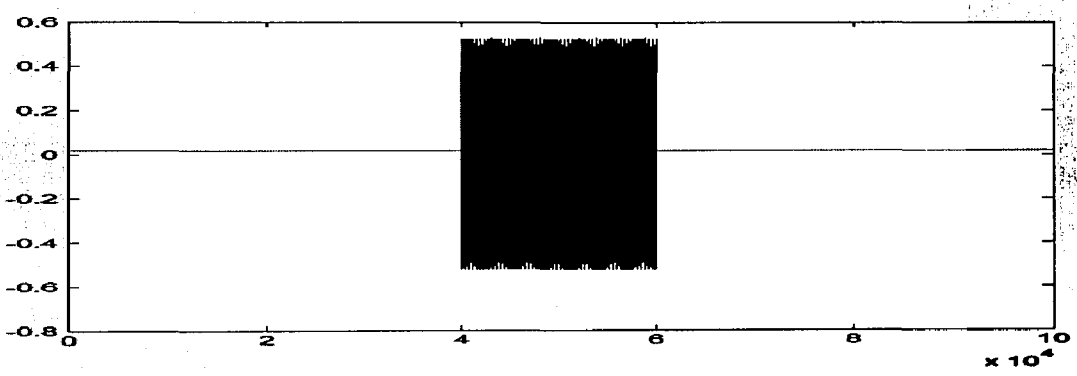
Figure 6.11 Test 7

6.10 Test 8: Non-convergence, in-band signaling, and continuity check tones

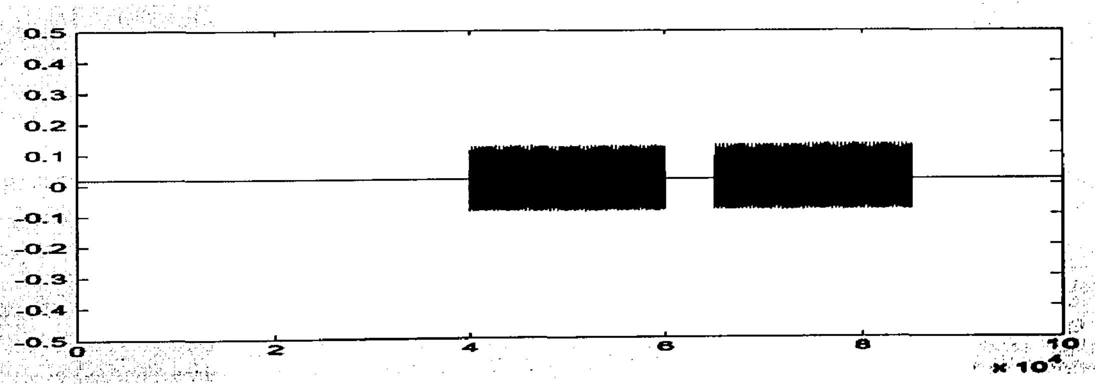
The G.168 Requirements for Test 8: in this test, the signal applied at near end signal must remain uncancelled, while the echo of far end signal should be cancelled. The variation at filter output as compared to input should be within ± 2 dB.

Far end/near end Input	Variation Requirement	Result (NLP Enabled)
2400/2400	±2 dB	meet req.

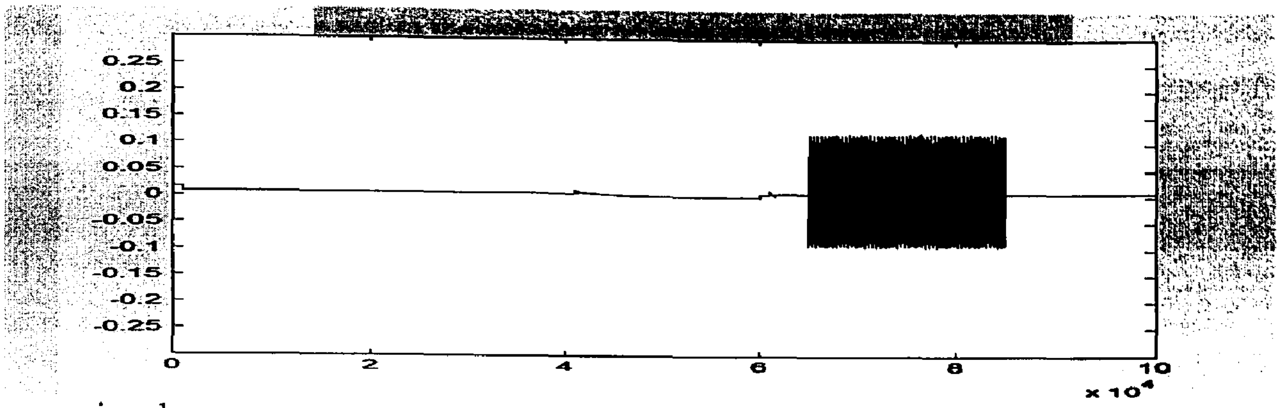
Table 6.14 Test 8 results



far end signal



received signal



error signal

Figure 6.12 Test 8

6.11 Summary

Test 1: Steady state residual and residual echo level test

L(Rin) Input	**L(Res), NLP Dis Output Req.**	**Result**	**L(Res), NLP enab Output Req.**	**Result (NLP Enabled)**
-10 dBm0	-37 dBm0	-46.15 dBm0	-65 dBm0	-66.15 dBm0

Test 2A and 2B: Convergence test with NLP enabled and disabled

Test 2A

Time (ms)	**L (Rin)-L (Ret) (NLP Enabled)**	**Result (NLP Enabled)**
0	6 dB	48 dB
50ms-1s	6 dB~20 dB	50 dB
1s+	@-10dBm0 55dB	57 dB

Test 2B

Time	**L (Rin)-L (Ret) (NLP Disabled)**	**Result (NLP Disabled)**
0	6 dB	8.8 dB
50ms-1s	6 dB~20 dB	9 dB ~20 dB
1s-10s	20 dB	40 dB
10s+	@-10dBm0 27dB	42dB

Test 2C: Convergence test in the presence of background noise

Test 2C(a)

L(Rin) Input	Noise Level L(Rin)-15	L(Ret) Requirement	Result (NLP Enabled)
-10 dBm0	-30 dBm0	-30 dBm0	-46 dBm0

Test 2C(b)

L(Rin) Input	Noise Level	L(Ret) Requirement	Result (NLP Enabled)
-10 dBm0	-55 dBm0	-38 dBm0	-46 dBm0

Test 2C(c)

L(Rin) Input	Noise Level L(Rin)-15	L(Ret) Requirement	Sample (NLP Disabled)
-10 dBm0	-30 dBm0	-30 dBm0	-37 dBm0

Test 3: Performance under conditions of double talk

Test 3A

L(Rin) Input	Low Near-End L(Rin)-15	Level L(Res) Requirement	Result (NLP Disabled)
-10 dBm0	-25 dBm0	-25 dBm0	-34 dBm0

Test 3B

L(Rin) Input	Doubletalk N≥L(Rin)	Level L(Ret) Requirement	Result (NLP Disabled)
-10 dBm0	-10 dBm0	-27 dBm0	-35 dBm0

Test 3C

L(Rin) Input	Doubletalk Level N Performance	t2(Req.)	Result
-10 dBm0	-10 dBm0	no peaks>N	meets req.

t3(Req.) Performance	Result	t4(Req.)	Result
-65 dBm0	-52 dBm0	no peaks>N+6 dBm0	meets req.

Test 4: Leak rate test

L(Rin) Input	**L(Ret) Requirement (NLP Disabled)**	**Result**
-10 dBm0	-27 dBm0	-25 dBm0

Test 5: Infinite return loss convergence test

L(Rin) Input	**L(Rin) – L(Res) Requirement (NLP Disabled)**	**Sample**
-10 dBm0	>20 dB after 1 sec >27 dB after 10 sec	meet req.

Test 6: Non-divergence on narrow-band signals

L(Rin)	**L(Ret) Final Requirement**	**Result (NLP Disabled)**
-10 dBm0	-27 dBm0	-32 dBm0

Test 7: Stability test

L(Rin) Input	**L(Res), NLP dis Output Requirement**	**Result (NLP Disabled)**
-10 dBm0 (1 kHz)	-38 dBm0	-48 dBm0

Test 8: Non-convergence, in-band signaling, and continuity check tones

Far end/near end Input	**Variation Requirement**	**Result (NLP Enabled)**
2400/2400	±2 dB	meet req.

Table 6.15 Test result summary

Chapter 7

Real time simulation

7.1 TI TMS320C6711 DSK

The TI TMS320 floating-point family includes C3x, C4x, and C67x. Each generation of the TMS320 series has a unique central processing unit (CPU) with a variety of memory and peripheral configurations. In this project, the TMS320C6711 DSK is chosen to the real-time AEC simulations.

The TI TMS320C6711 DSK is a digital signal processing development kit used to prototype DSP applications targeted for the C6711 family of processors. It has a 3.5-mm audio IN jack and 3.5-mm audio OUT jack, with ADC and DAC executed onboard. The AD535 codec works at a fixed sample rate of 8kHz. The DSK also includes 16MB of synchronous dynamic RAM and 128 KB flash ROM and the 150-MHz C6711DSP is capable of executing 900 million floating-point operations per second (MFLOPS).

The DSK board is connected to a PC via a parallel port. The program files can be created in TI code composer studio on the PC, and then loaded onto the DSK [16].

7.2 C6711 DSK Simulation using Code Composer Studio

7.2.1 Simulation Procedure Introduction

CCS IDE from TI is easy to use development environment allows DSP designers of all experience levels to move quickly through each phase of the application development process including design, code and build, debug, analyze and optimize. The fully integrated development environment includes, real-time analysis capabilities, easy to use debugger, C/C++ Compiler, Assembler, linker, editor, visual project manager, simulators, XDS560 and XDS510 emulation drivers and DSP/BIOS support.

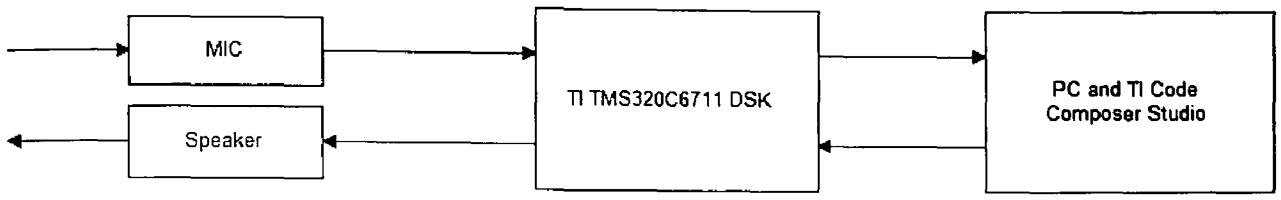
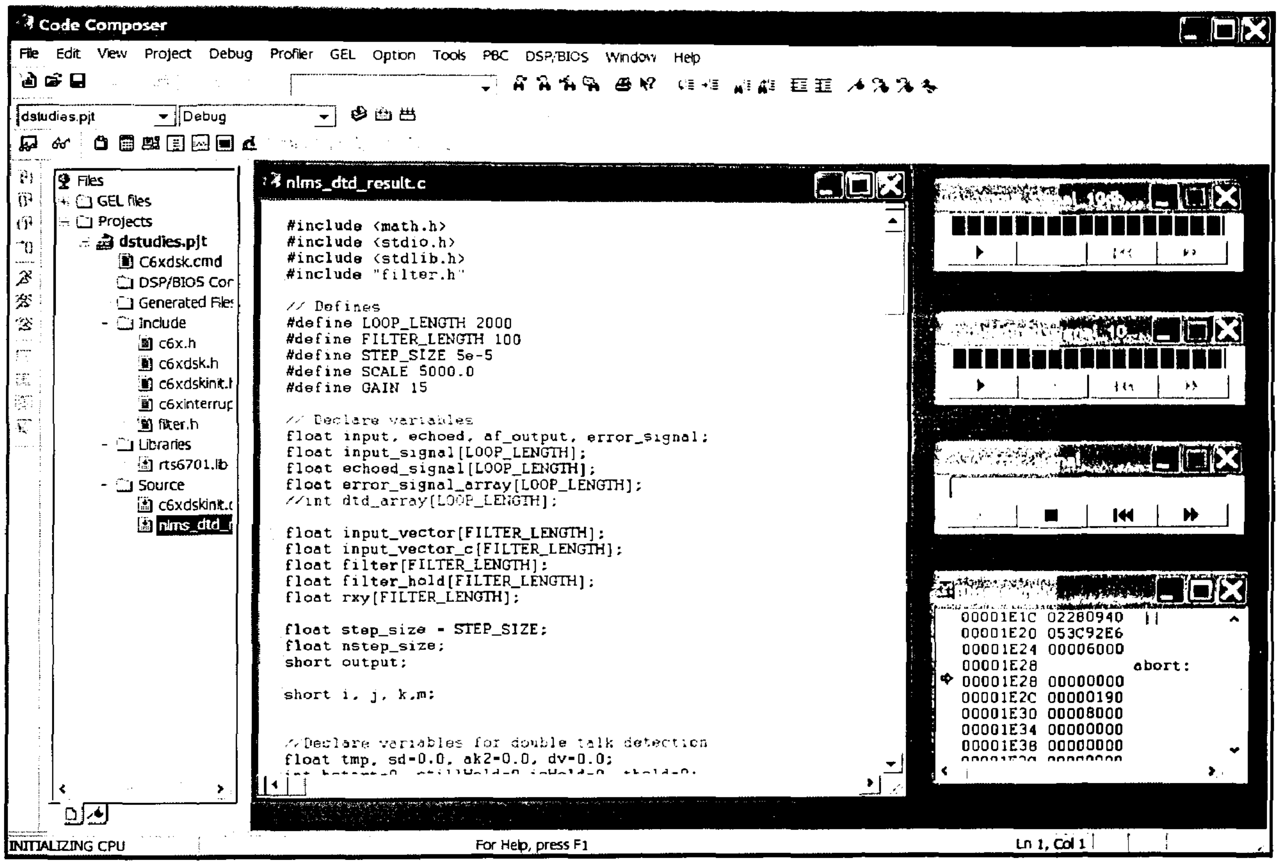


Figure 7.1 Adaptive echo cancellation implementation block diagram

The real-time acoustic echo cancellation system has been implemented as shown in figure 8.1. A microphone is used to input a voice signal from the user, this is then fed into the Texas Instrument TMS320C6711 development board. Based on human voice frequency range and adaptive echo cancellation system performance consideration, the sampling rate 8 KHz is used in the AD535 codec. The echo cancellation system use the normalized LMS algorithm and the normalized cross-correlation DTD algorithm detailed in previous sections. At each new sample time the input is sampled and input to memory and these memory audio data can be output to a file. In this project, there are two audio data files that contain these samples are loaded into memory as far end signal and received signal, and the CCS development workbench window is show in the Figure 8.2. The step size value for this iteration of the NLMS algorithm is then calculated by inverting the dot product of the input vector and a small constant should be included in the denominator to avoid zero divides. The output of the adaptive filter is then calculated by the dot product of the input vector and the current filter tap weight vector. This output is then subtracted from the desired signal to determine the estimation error value, e(n). This error value, the step size value and the current FIR tap weight vector are then input to the NLMS algorithm to calculate the filters tap weight to be used in the next iteration. If the filter is claimed to converge, the DTD detector will start monitor if there is a double talk situation. When double talk happens, the DTD detector inhibits the filter adaptation for a hold time to avoid divergence of the adaptive algorithm. Finally the AEC output error signal data is streamed out to a specified file using File I/O function provided by CCS to do further analysis [13][16][17].



```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "filter.h"

// Defines
#define LOOP_LENGTH 2000
#define FILTER_LENGTH 100
#define STEP_SIZE 5e-5
#define SCALE 5000.0
#define GAIN 15

// Declare variables
float input, echoed, af_output, error_signal;
float input_signal[LOOP_LENGTH];
float echoed_signal[LOOP_LENGTH];
float error_signal_array[LOOP_LENGTH];
//int dtd_array[LOOP_LENGTH];

float input_vector[FILTER_LENGTH];
float input_vector_c[FILTER_LENGTH];
float filter[FILTER_LENGTH];
float filter_hold[FILTER_LENGTH];
float rxy[FILTER_LENGTH];

float step_size = STEP_SIZE;
float nstep_size;
short output;

short i, j, k,m;


//Declare variables for double talk detection
float tmp, sd=0.0, ak2=0.0, dv=0.0;
```

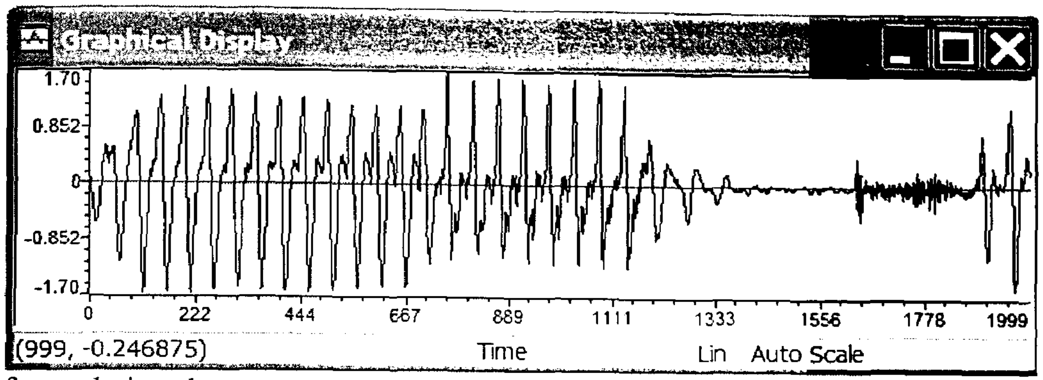
```
  00001E1C 02280940  ||
  00001E20 053C92E6
  00001E24 00006000
  00001E28                abort:
  00001E28 00000000
  00001E2C 00000190
  00001E30 00008000
  00001E34 00000000
  00001E38 00000000
```

Figure 7.2 CCS Workbech window

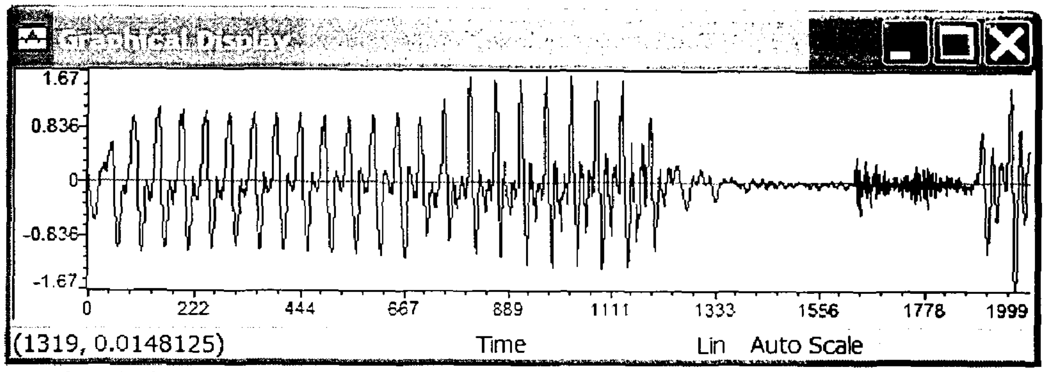
7.2.2 C6711 DSK Simulation Results

Using the CCS graphing function provided by CSS integration development environment, the far end signal, received signal and the error signal of the echo cancellation system can be viewed and shown in the following figures. As can be seen that the real time adaptive echo cancellation system is successfully developed with the NLMS and the normalized cross-correlation DTD algorithm.

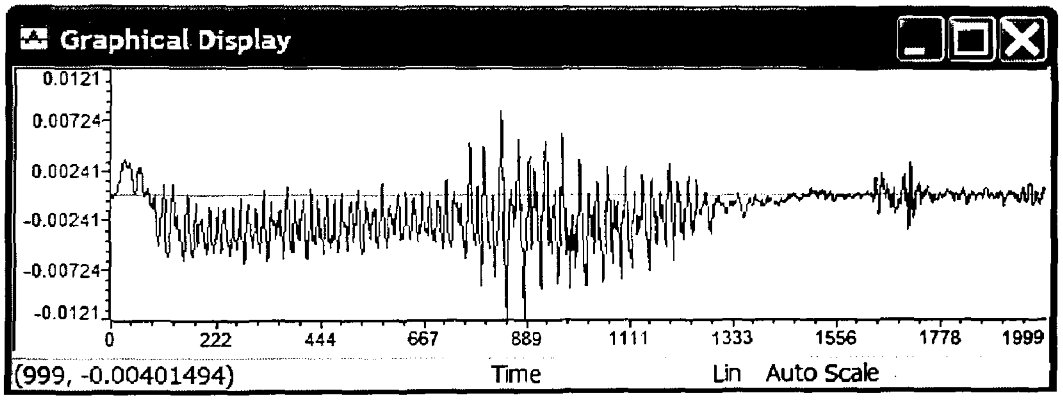
(1) Steady state residual and residual echo level test



far end signal



received signal



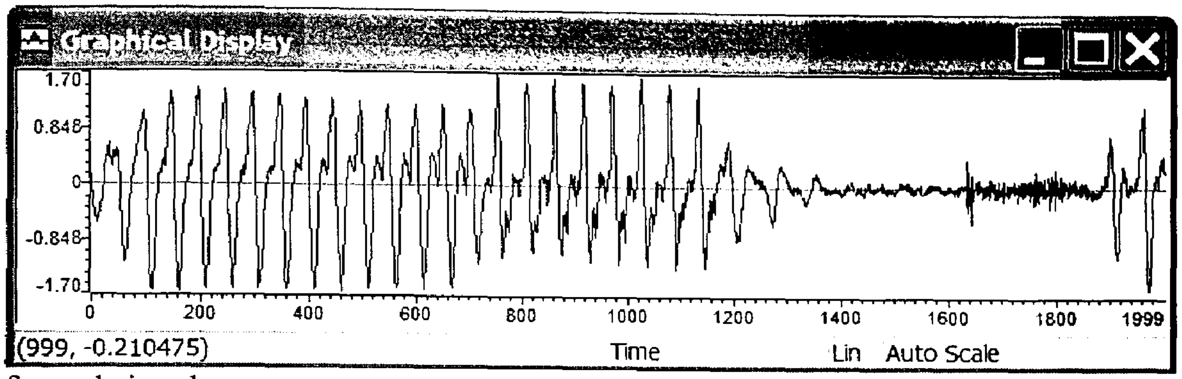
error signal

Figure 7.3 Steady state residual and residual echo level test

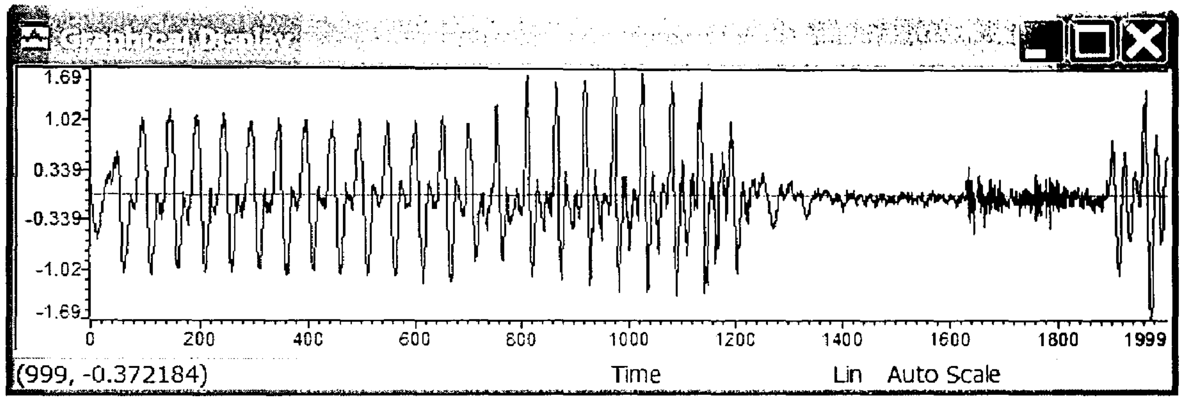
Input	**Output Req.**	**Result**	**Matlab Results**
-10 dBm0	-37 dBm0	-44 dBm0	-46 dBm0

Table 7.1 Steady state residual and residual echo level test result

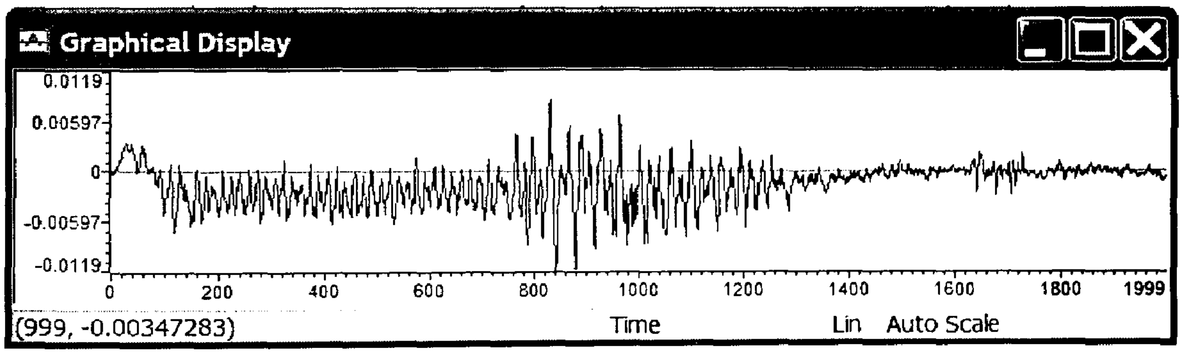
(2) Convergence test in the presence of background noise



far end signal



received signal



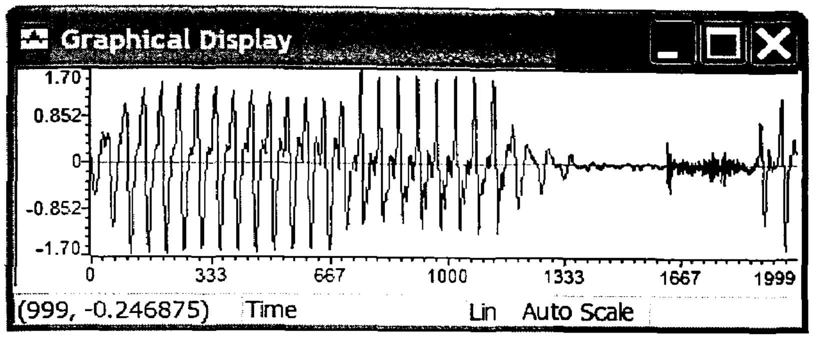
error signal

Figure 7.4 Convergence test in the presence of background noise

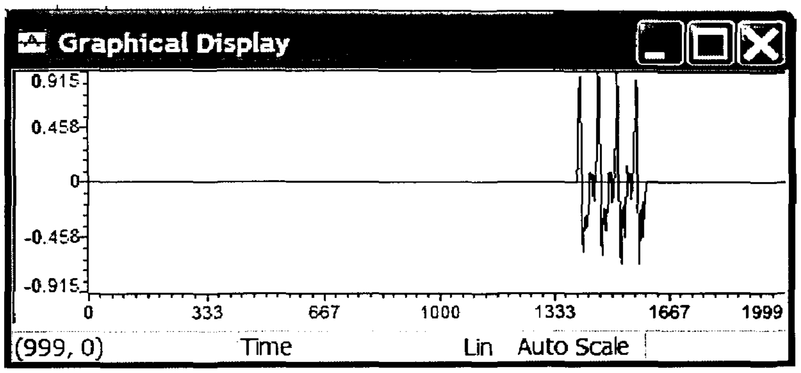
Input	**Noise**	**Output Req.**	**Matlab Result**	**Result**
-10 dBm0	-30 dBm0	-30 dBm0	-46 dBm0	-44 dBm0

Table 7.2 Convergence test in the presence of background noise test results

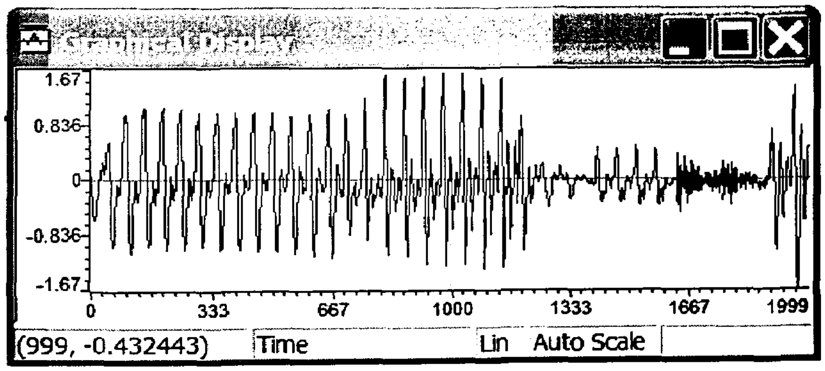
(3) Double talk test with low near end levels



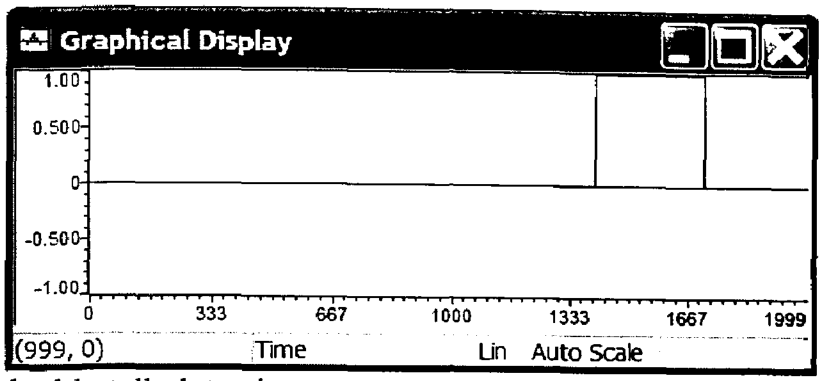
far end signal



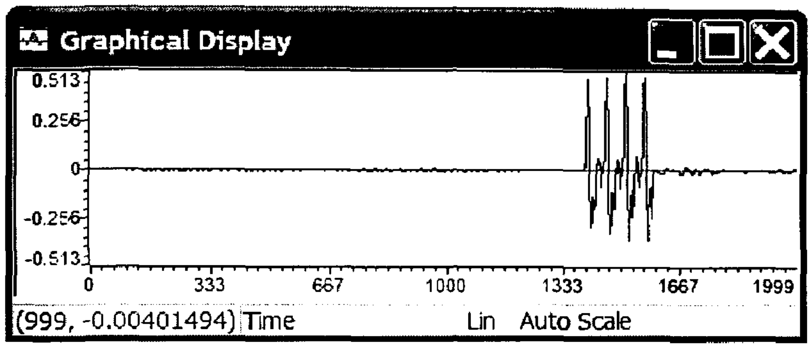
near signal



received signal



double talk detection



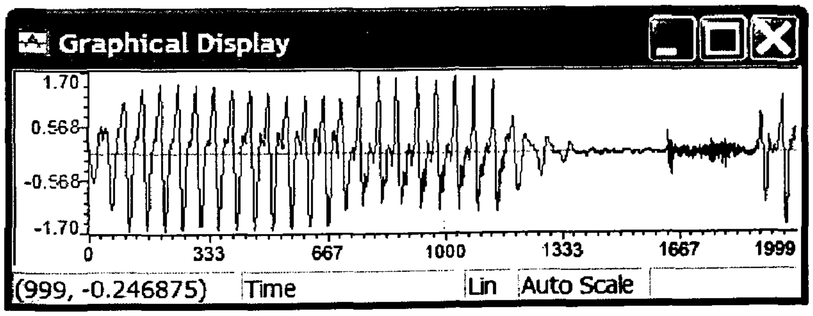
error signal

Figure 7.5 Double talk test with low near end levels

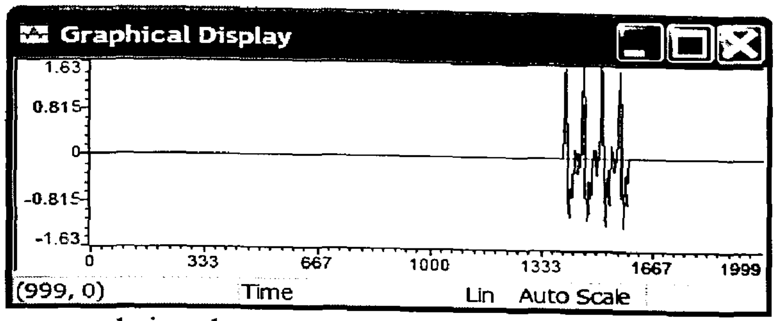
Input	**Near end**	**Output Req.**	**Matlab Result**	**Result**
-10 dBm0	-25 dBm0	-25 dBm0	-34dBm0	-33dBm0

Table 7.3 Double talk test with low near end levels test results

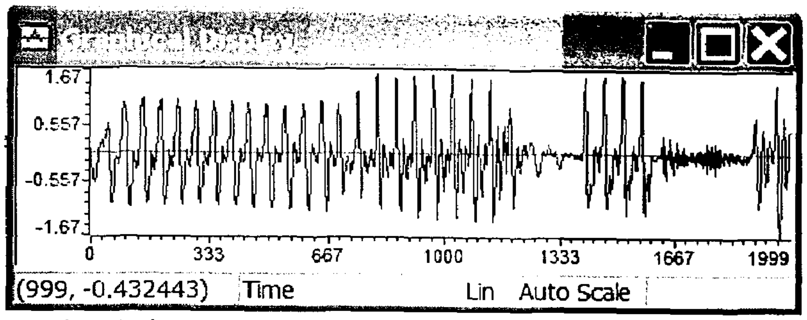
(4) Double talk test with high near end levels



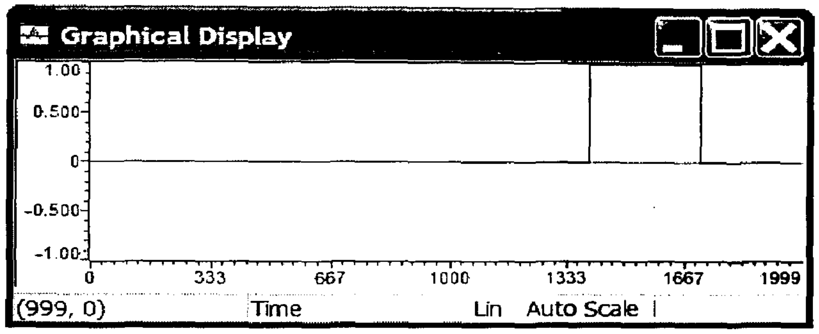
far end signal



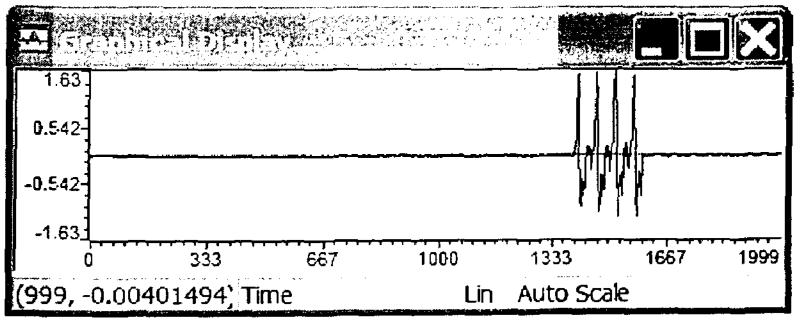
near end signal



received signal



double talk detecion



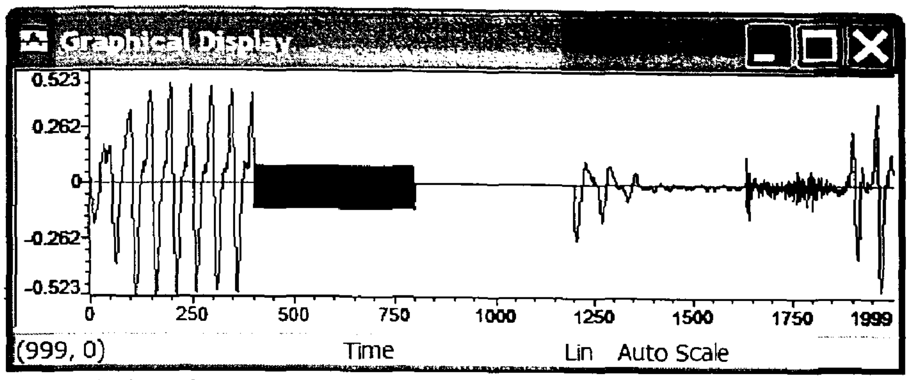
error signal

Figure 7.6 Double talk test with high near end levels

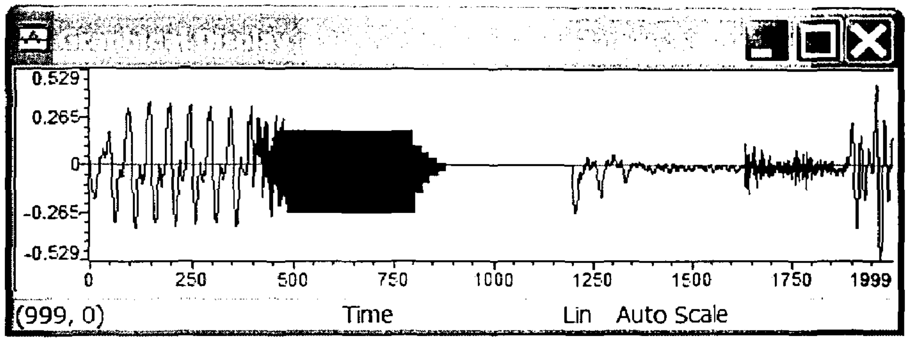
Input	**Near end**	**Output Req.**	**Matlab Result**	**Result**
-10 dBm0	-10 dBm0	-27 dBm0	-35 dBm0	-30 dBm0

Table 7.4 Double talk test with high near end levels test results

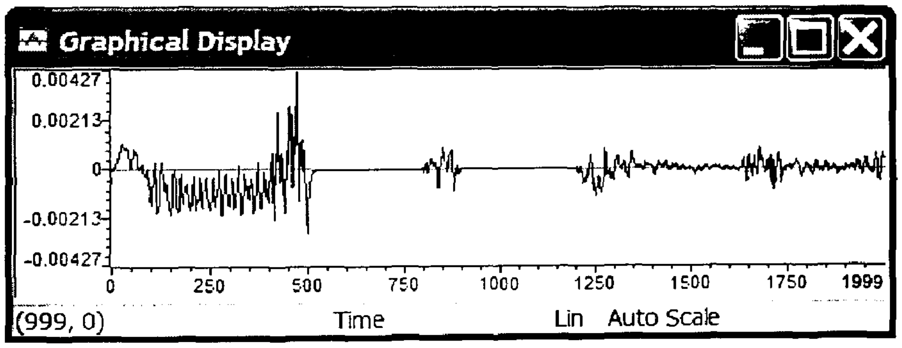
(5) Non-divergence on narrow-band signals



far end signal



received signal



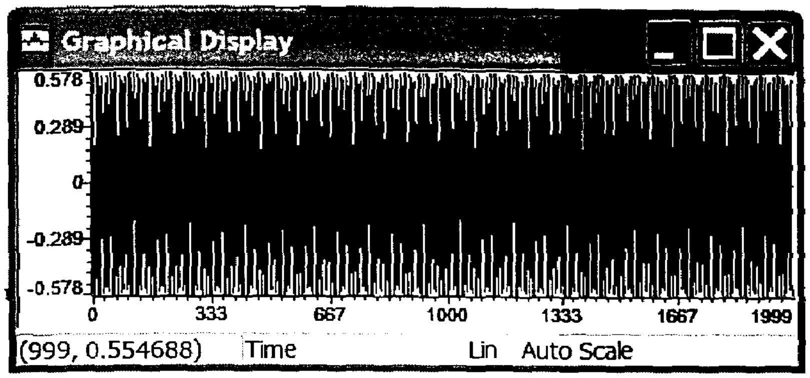
error signal

Figure 7.7 Non-divergence on narrow-band signals

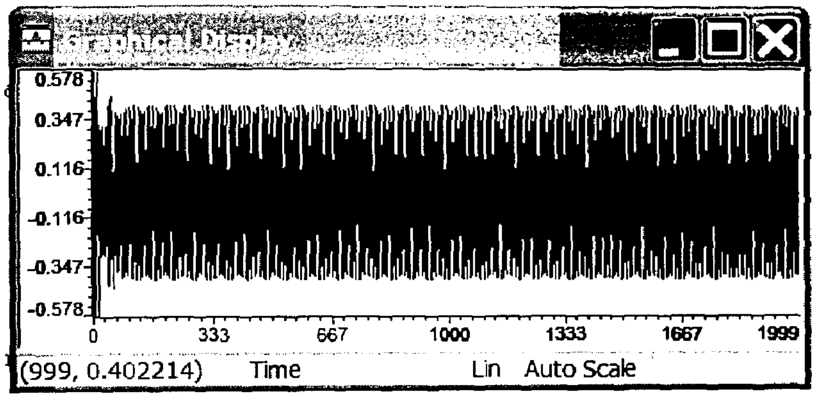
Input	**Output Req.**	**Matlab Result**	**Result**
-10 dBm0	-27 dBm0	-32 dBm0	-34 dBm0

Table 7.5 Non-divergence on narrow-band signals test result

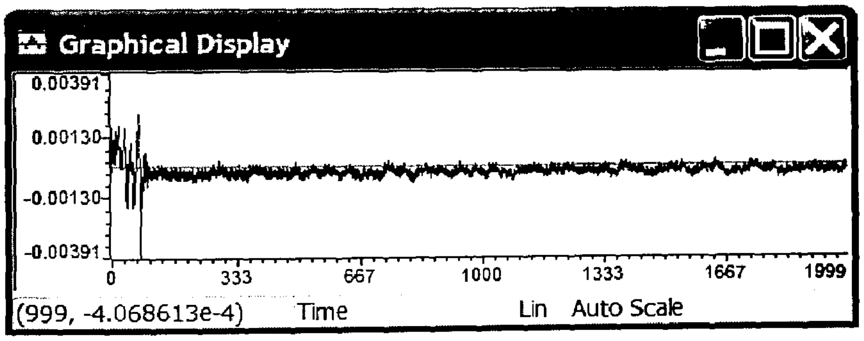
(6) Stability test



far end signal



received signal



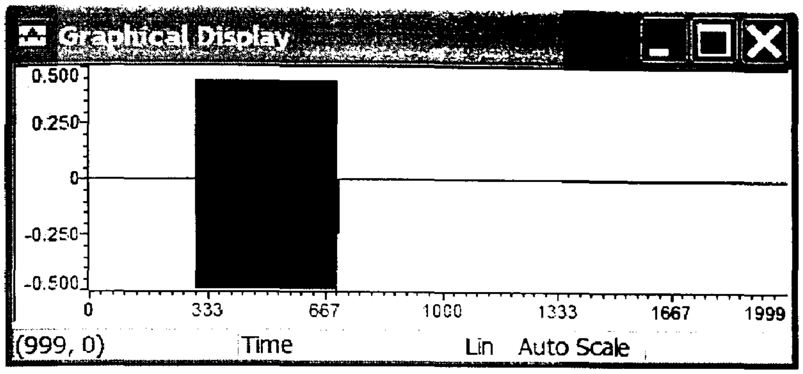
error signal

Figure 7.8 Stability test

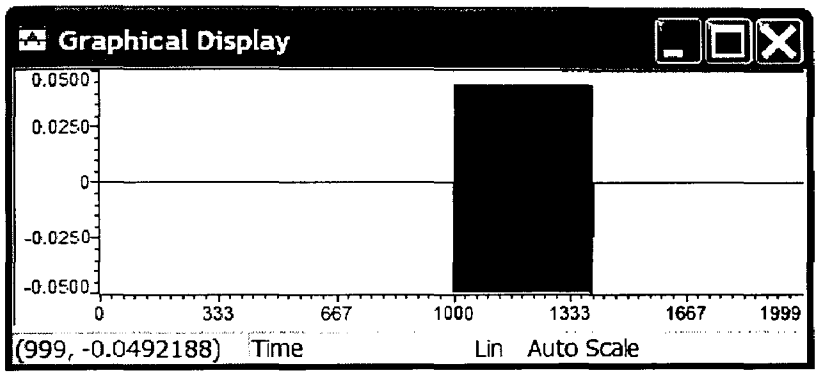
Input	Output Req.	Matlab Result	Result
-20 dBm0	-47 dBm0	-48 dBm0 @ -10 dBm0 input	-58 dBm0

Table 7.6 Stability test results

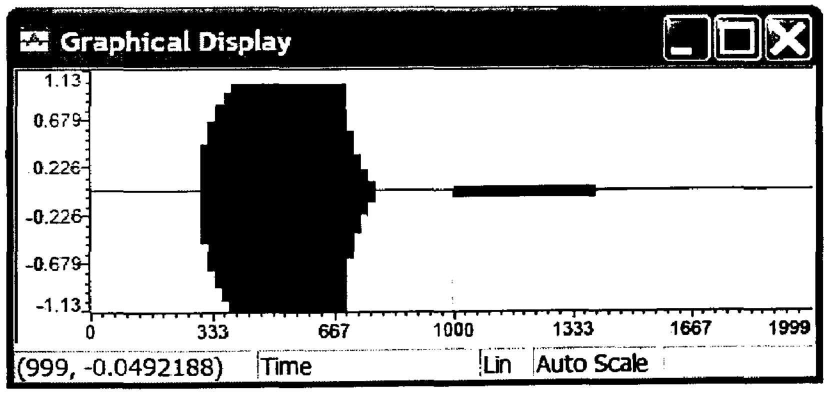
(7) Non-convergence, in-band signaling, and continuity check tones



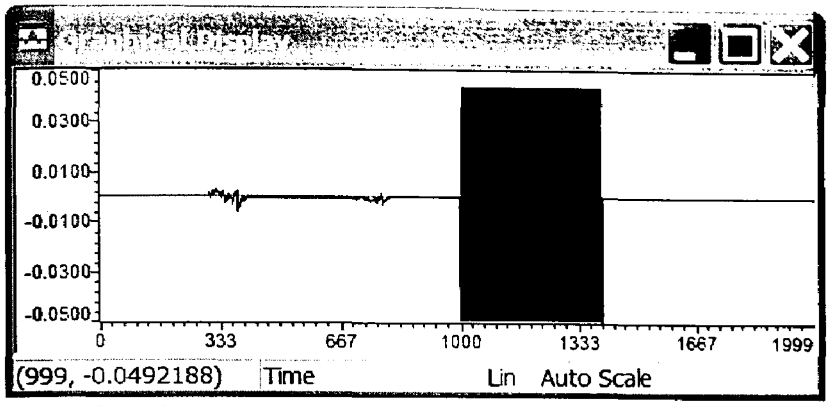
far end signal



near end signal



received signal



error signal

Figure 7.9 Non-convergence, in-band signaling, and continuity check tones

Far end/near end Input	Variation Requirement	Result (NLP Enabled)
2400/2400 Hz	±2 dB	meet req.

Table 7.7 Non-convergence, in-band signaling, and continuity check tones test results

7.3 Real time simulation summary

Test 1: Steady state residual and residual echo level test

Input	Output Req.	Result	Matlab Result
-10 dBm0	-37 dBm0	-44 dBm0	-46 dBm0

Test 2C: Convergence test in the presence of background noise

Input	Noise	Output Req.	Matlab Result	Result
-10 dBm0	-30 dBm0	-30 dBm0	-46 dBm0	-44 dBm0

Test 3: Performance under conditions of double talk

Test 3A

Input	Near end	Output Req.	Matlab Result	Result
-10 dBm0	-25 dBm0	-25 dBm0	-34dBm0	-33dBm0

Test 3B

Input	Near end	Output Req.	Matlab Result	Result
-10 dBm0	-10 dBm0	-27 dBm0	-35 dBm0	-30 dBm0

Test 6: Non-divergence on narrow-band signals

Input	**Output Req.**	**Matlab Result**	**Result**
-10 dBm0	-27 dBm0	-32 dBm0	-34 dBm0

Test 7: Stability test

Input	**Output Req.**	**Matlab Result**	**Result**
-20 dBm0	-47 dBm0	-48 dBm0 @ -10 dBm0 input	-58 dBm0

Test 8: Non-convergence, in-band signaling, and continuity check tones

Far end/near end Input	**Variation Requirement**	**Result (NLP Enabled)**
2400/2400	±2 dB	meet req.

Table 7.8 Real time simulation summary

Chapter 8

Conclusion

In this project, four adaptive algorithm LMS, NLMS, Fast Block LMS and RLS are investigated and simulated using AEC Analysis and Design program in Matlab for echo cancellation. Comparing these algorithms, NLMS is finally selected as the best algorithm for the real time echo cancellation systems. In double talk detection, the Geigel algorithm and the normalized cross-correlation DTD algorithm are separately integrated with NLMS FIR filter against double talk and are studied in NLMS & DTD Analysis and Design program in Matlab. The normalized cross-correlation DTD algorithm is selected for double talk detection, since the results are much better. Through testing with ITU G.168 standard, the performance of the adaptive echo cancellation system conforms with the standard. In addition, the real time adaptive echo cancellation simulation by TI TMS320C6711 also proves the selected algorithms and their settings are correct.

Chapter 9

Further Work

There are some possibilities for further development in this project, some of these are as follows.

- The double talk detection algorithm still can be done further studies in the simplified smoothing estimation algorithm area to improve the alarm accuracy and reduce the computation complexity.
- The real time echo cancellation system is successfully developed using the TI TMS320C6711 DSK. However, since this system is code in C language, if it is in assembler, the system performance will be much improved and If the TI C5000 DSP using Fixed-Point Digital Signal Processor, the same performance can be achieved with the lower cost.
- Instead of the DSK board, the adaptive echo cancellation system can use the TMS320C6711 digital signal processor in a custom designed circuit by which will also improve the whole system performance.

References

[1] C.F.N. Cowan, P.M. Grant, *Adaptive Filters*, Prentice-Hall, New Jersey, 1985.

[2] Paulo S.R. Diniz, *Adaptive Filtering, Algorithms and Practical implementation*, Kluwer Academic Publishers, Boston, 1997.

[3] D. P. Mandic, "A Generalized Normalized Gradient Descent Algorithm," IEEE Signal Processing Letters, vol. 11, pp. 115-118, February 2004.

[4] D. L. Duttweiler, "A twelve-channel digital echo canceler," IEEE Trans. Communication, vol. 26, pp. 647-653, May 1978.

[5] J. Benesty, D. R. Morgan, and J. H. Cho, "A new class of doubletalk detectors based on cross-correlation," IEEE Trans. Speech Audio Processing, vol. 8, pp. 168-172, March 2000.

[6] J. H. Cho, D. R. Morgan, and J. Benesty, "An objective technique for evaluating doubletalk detectors in acoustic echo cancelers," IEEE Trans. Speech Audio Processing, vol. 7, pp. 718–724, Nov. 1999.

[7] P. Ahgren, "A new doubletalk detection algorithm with a very low computational complexity," Submitted to IEEE Trans. Speech Audio Proc., December 2003.

[8] J. Benesty, D. R. Morgan, and J. H. Cho, "A new class of doubletalk detectors based on cross-correlation," IEEE Trans. Speech Audio Processing, March 2000.

[9] H. Ye and B.-X. Wu, "A new double-talk detection algorithm based on the orthogonality theorem," IEEE Trans. Commun., vol. 39, pp.1542–1545, Nov. 1991.

[10] T. G¨ansler, M. Hansson, C.-J. Invarsson, and G. Salomonsson, "A double-talk detector based on coherence," IEEE Trans. Commun., vol.44, pp. 1421–1427, Nov. 1996.

[11] Michael L. Honig, David G. Messerschmitt, *Adaptive Filters, Structures, Algorithms and Applications*, Kluwer Academic Publishers, Boston, 1984.

[12]C. Antweiler and M. Dörbecker, "Perfect sequence excitation of the NLMS algorithm and its application to acoustic echo control," Annales des Telecommunications, no. 7–8, pp. 386–397, July–August 1994.

[13] Chassaing, Rulph, *DSP applications using C and theTMS320C6x DSK*, John Wiley and Sons, New York, 2002.

[14] P. Heitkamper, "An adaptation control for acoustic echo cancellers," IEEE Signal Processing Lett., vol. , pp. 170–172, June 1997.

[15] B. Farhang-Boroujeny, *Adaptive Filters, Theory and Applications*, John Wiley & Sons, 1998.

[16] Texas Instruments: Code Composer Studio/ TMS320C6711, Documentation CD accompanying theTMS320C6711 DSP Starter Kit, 2002.

[17] Dave Bell, "How to Begin Development With the TMS320C6711 DSP," Texas Instruments Application Notes, Available: http://focus.ti.com/lit/an/spra522/spra522.pdf, March 1999.

APPENDIX A: MATLAB CODE

lms_function.m

```
function  [error_signal,desired_signal,filter_output,filter_current,mse,db]
                                                =lms_function(input_signal,filter_size,step_size,impulse)

desired_signal = conv(input_signal, impulse);

% initialise adaptive filter
filter_current = zeros(filter_size,1);
input_vector = zeros(filter_size, 1);
iterations=length(input_signal);

q = waitbar(0,'LMS Filtering...');
for i=1:iterations

   input_vector(1)=input_signal(i);
   filter_output(i)=dot(filter_current, input_vector);
   error= desired_signal(i)-filter_output(i)  ;
   filter_current = filter_current + 2*step_size*error*input_vector;

   for j=filter_size:-1:2
      input_vector(j)=input_vector(j-1);
   end

   error_signal(i)=error;
   cost(i)=error*error;

   waitbar( i/iterations, q);
end
close(q);

q1 = waitbar(0,'LMS Caculating MSE...');
for i=1:iterations-100
   mse(i)=mean(cost(i:i+100));
    waitbar( i/iterations, q1);
end
close(q1);

q2 = waitbar(0,'LMS Caculating attenuation in dB...');
for i=1:iterations-2500
   db(i)=-20*log10(mean(abs(desired_signal(i:i+2500)))'./mean(abs(error_signal(i:i+2500))));
    waitbar( i/iterations, q2);
end
close(q2);
```

nlms_function.m

```
function [error_signal,desired_signal,filter_output,filter_current,mse,db]
                                        =nlms_function(input_signal,filter_size,impulse)

iterations = length(input_signal);
desired_signal = conv(input_signal, impulse);

% initialise adaptive filter
filter_current = zeros(filter_size,1);
input_vector = zeros(filter_size, 1);
q = waitbar(0,'Filtering...');

for i=1:iterations

   input_vector(1)=input_signal(i);
   filter_output(i)=dot(filter_current, input_vector);
   error= desired_signal(i)-filter_output(i);
   step_size=1/(dot(input_vector, input_vector)+0.00001);
   filter_current = filter_current + step_size*error*input_vector;
   for j=filter_size:-1:2
      input_vector(j)=input_vector(j-1);
   end
   error_signal(i)=error;
   cost(i)=error*error;
   ss(i)=step_size;
   waitbar( i/iterations, q );
end
close(q);

q1 = waitbar(0,'Caculating MSE...');
for i=1:iterations-100
  mse(i)=mean(cost(i:i+100));
   waitbar( i/iterations, q1);
 end
close(q1);

q2 = waitbar(0,'Caculating attenuation in dB...');
for i=1:iterations-2500
   db(i)=-20*log10(mean(abs(desired_signal(i:i+2500)))'./mean(abs(error_signal(i:i+2500))));
    waitbar( i/iterations, q2);
end
close(q2);
```

flms_function.m

```
function[error_signal,desired_signal, filter_output,filter_coeff,mse,db]=
                        flms_function(input_signal,filter_size,step_size,estimated_power,impulse,lambda)

% initialization
FILTER_COEFF = zeros(2*filter_size,1);
input_length = length(input_signal);
desired_signal = conv(input_signal, impulse);

block_length = floor(input_length/filter_size)*filter_size;

input_signal = input_signal(1:block_length);
desired_signal = desired_signal(1:block_length);

input_signal = input_signal(:);
desired_signal = desired_signal(:);

error_signal = desired_signal;

Blocks = block_length/filter_size;

q = waitbar(0,'Fast LMS Filtering...');
% loop, FLMS
for k=1:Blocks-1


  INPUT_SIGNAL = fft([input_signal((k-1)*filter_size+1:(k+1)*filter_size)],2*filter_size);

  filter_output = ifft(INPUT_SIGNAL.*FILTER_COEFF);
  filter_output = filter_output(filter_size+1:2*filter_size,1);

  desired_vec = desired_signal(k*filter_size+1:(k+1)*filter_size);

  error_signal(k*filter_size+1:(k+1)*filter_size,1) = desired_vec-filter_output;

  ERROR_VEC = fft([zeros(filter_size,1);error_signal(k*filter_size+1:(k+1)*filter_size)],2*filter_size);

  estimated_power=lambda*estimated_power+(1-lambda)*abs(INPUT_SIGNAL).^2;

  DESIRED_VEC = 1./(1+estimated_power);

  phivec = ifft(DESIRED_VEC.*conj(INPUT_SIGNAL).*ERROR_VEC,2*filter_size);
  phivec = phivec(1:filter_size);

  FILTER_COEFF = FILTER_COEFF+step_size*fft([phivec;zeros(filter_size,1)],2*filter_size);

  error_signal = real(error_signal(:));

  filter_coeff = ifft(FILTER_COEFF);
  filter_coeff = real(filter_coeff(1:length(FILTER_COEFF)/2));
  filter_output=real(filter_output(:));

  waitbar(k/(Blocks-1), q);
end
close(q);
```

```
cost=error_signal.*error_signal;
q1 = waitbar(0,'Fast LMS Caculating MSE...');
iterations=length(cost);
for i=1:iterations-100
   mse(i)=mean(cost(i:i+100));
    waitbar( i/iterations, q1);
end
close(q1);

q2 = waitbar(0,'Fast LMS Caculating attenuation in dB...');
iterations=length(desired_signal);
for i=1:iterations-2500
   db(i)=-20*log10(mean(abs(desired_signal(i:i+2500)))'./mean(abs(error_signal(i:i+2500))));
    waitbar( i/iterations, q2);
end
close(q2);
```

rls_function.m

```
function [error_signal,desired_signal,filter_output,filter_current,mse,db]
                    =rls_function(input_signal,filter_size,lambda,impulse)

desired_signal = conv(input_signal, impulse);
iterations=length(input_signal);

% initialise adaptive filter
filter_prev = zeros(filter_size,1);
input_vector = zeros(filter_size, 1);
psi_inv_prev = eye(filter_size);
intermediate= zeros(filter_size, 1);
gain = zeros(filter_size, 1);

q = waitbar(0,'RLS Filtering...');
for i=1:iterations
   input_vector(1)=input_signal(i);
   intermediate = psi_inv_prev*input_vector;
   gain = (1/(lambda+dot(input_vector, intermediate)))*intermediate;
   filter_output(i)=dot(filter_prev, input_vector);
   error= desired_signal(i)-filter_output(i);
   filter_prev = filter_prev + gain*error;
   psi_inv_prev = (1/lambda)*(psi_inv_prev - gain*((input_vector')*psi_inv_prev));

   for j=filter_size:-1:2
      input_vector(j)=input_vector(j-1);
   end
   error_signal(i)=error;
   cost(i)=error*error;

  waitbar( i/iterations, q);

end
close(q);

q1 = waitbar(0,'RLS Caculating MSE...');
for i=1:iterations-100
   mse(i)=mean(cost(i:i+100));
    waitbar( i/iterations, q1);
end
close(q1);

q2 = waitbar(0,'RLS Caculating attenuation in dB...');
for i=1:iterations-2500
  db(i)=-20*log10(mean(abs(desired_signal(i:i+2500)))'./mean(abs(error_signal(i:i+2500))));
 waitbar( i/iterations, q2);
end
close(q2);

db_avg=mean(db)

end
```

nlmsGeigel_function.m

```
function [e,xF,xE,v,y,s,th,db,holdRec]=nlmsGeigel_function(L,TG,geigLen,hold_time)

farendThres = 3.5e6;

hstart=0;
thold=0;
isHold=0;
stillHold=0;


% Load data files.
xF = readData( 'Far.pcm' );
xE = readData( 'FarEcho.pcm' );
v  = readData( 'Near.pcm' );
y = xE + v;
xF=xF(10e+4:end);
y=y(10e+4:end);

% Initialize adaptive filter
e = zeros( size(xF) );     % Error signal.
s = zeros( size(xF) );     % Estimated echo signal.
state = eps*ones(L,1);
th    = state;
th0   = state;
dG    = zeros( size(xF) );

% show if last sample was detected as DT.
wasDT   = 0 ;
noDTcounter = 0;

loopLen = length(xF);
holdRec = zeros( size(xF) );

q = waitbar(0,'NLMS & Giegel DTD Filtering...');
for k=1:loopLen,
   % Update the filter state.
   if k>L,
      state(1:end) = flipud(xF(k-L+1:k));
   end
   s(k) = state' * th0;            % Estimated echo value.
   e(k) = y(k) - s(k);             % Prediction error.

   if (k-20000) > L
      % Geigel DTD.
      dG(k) = abs( y(k) )/ max( abs(xF(k-geigLen:k) ) );
   end


   if dG(k) > TG
     hstart=1;
     stillHold=1;
   else
      stillHold=0;
   end
```

```
      if hstart==1
        thold=thold+1;
        if thold < hold_time
           isHold =1;

           if stillHold
           thold=0;
           end


        else
           hstart=0;
           thold=0;
           isHold=0;
           stillHold=0;

        end
     end

      if isHold ~=1
        % Update the step-size parameter using NLMS.
        normState = state'*state;
        mu = 1/( normState + 1e3 );
      end

     if( normState > farendThres & isHold ~=1)
        th = th + mu * e(k) * state;
     end

     if isHold
        th = th0;
     else
        th0 = th;
     end

     holdRec(k)=isHold;

     waitbar( k/loopLen, q );
  end
  close(q);

   q1 = waitbar(0,'NMLS & Giegel DTE calculating REL...');
  for i=1:loopLen-2500
     db(i)=-20*log10(mean(abs(y(i:i+2500)))'./mean(abs(e(i:i+2500))));
     waitbar( i/(loopLen-2500), q1 );
  end
  close(q1);

  end
```

nlmsNCC_function.m

```
function [e,xF,xE,v,y,s,th,db,holdRec]=nlmsNCC_function(L,dt_low,dt_high,cv_hold,dt_hold)
L  = 1024;
farendThres = 3.5e6;

% Load data files.
xF = readData( 'Far.pcm' );
xE = readData( 'FarEcho.pcm' );
v  = readData( 'Near.pcm' );

y = xE+v;

% Initialize adaptive filter
e = zeros( size(xF) );     % Error signal.
s = zeros( size(xF) );     % Estimated echo signal.
state = eps*ones(L,1);
th    = state;
th0   = state;
dG    = zeros( size(xF) );

% show if last sample was detected as DT.
wasDT   = 0 ;
noDTcounter = 0;

loopLen = length(xF);
dtm=zeros(size(xF));
rxy=zeros(L,1);
isDT=0;
dv=1;
convergeCounter=zeros(1000,1);
convergeFlag=0;
rxy=0;
i=0;
startFlag=0;
isCoverge=0;
wasCoverge=0;
hstart=0;
thold=0;
isHold=0;
holdRec = zeros( size(xF) );
stillHold=0;


q = waitbar(0,'NMLS & NCC: Filtering...');
for k=1:loopLen,
  % Update the filter state.
  if k>L,
    state(1:end) = flipud(xF(k-L+1:k));
  end

  s(k) = state' * th0;           % Estimated echo value.
  e(k) = y(k) - s(k);            % Prediction error.

  if k>2*L & rxy==0
    for m=0:L-1,
      rxy=rxy+flipud(xF(k-m-L+1:k-m)).*y(k);
   end

   ss=rxy'*th;


end

   if k > 2*L

   rxy=0.9*rxy+0.1*state(1:end).*y(k);
   ss=0.9*ss+0.1*y(k)*y(k);

           ak2=rxy'*th;
           ak1=ss;


          dv=ak2/ak1;

  dNCR(k)=dv;

 if convergeFlag ~=1

  if dv > dt_low & dv < dt_high
      isCoverge=1;
      startFlag=1;
  else
      isCoverge=0;

  end


  if startFlag & i==0
      i=i+1;
  end

  if startFlag & wasCoverge & isCoverge
      i=i+1;
      if i>cv_hold
         convergeFlag=1
         k
         for m=0:L-1,
            rxy=rxy+flipud(xF(k-m-L+1:k-m)).*y(k);
         end

         ss=rxy'*th;

      end
  else
     start=0;
     i=0;
  end

   wasCoverge=isCoverge;

end
```

```
if convergeFlag

   if (dv <(dt_low-0.15) | dv > (dt_high+0.25))
      hstart=1;
      stillHold=1;
   else
      stillHold=0;
    end

end

end

  % Update the step-size parameter using NLMS.


if hstart==1
   thold=thold+1;
   if thold < dt_hold
      isHold =1;

      if stillHold
      thold=0;
      end


   else
      hstart=0;
      thold=0;
      isHold=0;
      stillHold=0;

   end
end

if isHold ~=1
   normState = state'*state;
   mu = 1/( normState + 1e3 );
end

if( normState > farendThres & isHold ~=1)
   th = th + mu * e(k) * state;
end


if isHold
   th = th0;
else
   th0 = th;
end

holdRec(k)=isHold;


waitbar( k/loopLen, q );
```

```
end
close(q);

q1 = waitbar(0,'NLMS & NCC: calculating ERLE...');
for i=1:loopLen-2500
   db(i)=-20*log10(mean(abs(y(i:i+2500)))'./mean(abs(e(i:i+2500))));
   waitbar( i/(loopLen-2500), q1 );
end
close(q1);
```

APPENDIX B: TI CCS CODE

nlms_adfilt_result.c

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "filter.h"

// Defines
#define LOOP_LENGTH 2000
#define FILTER_LENGTH 100


// Declare variables
float input, echoed, af_output, error_signal;
float input_signal[LOOP_LENGTH];
float echoed_signal[LOOP_LENGTH];
float error_signal_array[LOOP_LENGTH];
//int dtd_array[LOOP_LENGTH];

float input_vector[FILTER_LENGTH];
float input_vector_c[FILTER_LENGTH];
float filter[FILTER_LENGTH];
float filter_hold[FILTER_LENGTH];
float rxy[FILTER_LENGTH];

float nstep_size;
short output;

short i, j, k,m;


//Declare variables for double talk detection
float tmp, sd=0.0, ak2=0.0, dv=0.0;
int hstart=0, stillHold=0,isHold=0, thold=0, isNLP=0;


// Procedure for determining dot product of two arrays
float dotp (float a[], float b[])
{
        float suml, sumh;
        suml=0;
        sumh=0;
        for(j=0; j<FILTER_LENGTH; j+=2)
        {
                suml += a[j] * b[j];
                sumh += a[j+1]*b[j+1];
        }
        return (suml+sumh);
}

//Procedure for determining dot product of one array and one value
void dotpv(float a[], float b)
```

```
{
          for(j=0; j<FILTER_LENGTH; j+=1)
          {
                    a[j]= a[j] * b;

          }


}

void dotVectorSum(float a[], float b[])
{

          for(j=0; j<FILTER_LENGTH; j+=1)
          {
                    a[j]= a[j] + b[j];

          }

}


void copyVector(float a[], float b[]){

for(j=0; j<FILTER_LENGTH; j+=1)
          {
                    a[j]= b[j];

          }


}


void decideDTD(){
          copyVector(input_vector_c,input_vector);
          dotpv(input_vector_c,echoed);
          dotpv(input_vector_c,0.1);
          dotpv(rxy, 0.9);
          dotVectorSum(rxy, input_vector_c);

          sd=0.9*sd+0.1*echoed*echoed;

   ak2=dotp(rxy, filter);
          dv=ak2/sd;

          if(dv<0.75 || dv>1.35){
                    hstart=1;
                    stillHold=1;
          }else{

                    stillHold=0;
          }

          if (hstart==1){
```

```
        thold=thold+1;

        if (thold < 80){
                isHold =1;
                if (stillHold==1){
                        thold=0;
                }
        }
        else{
                hstart=0;
                thold=0;
                isHold=0;
                stillHold=0;
        }
    }


}


doFiltering()
{

        for(k=0;k<LOOP_LENGTH;k++){


        input=input_signal[k];          // newest input cast to float
        input_vector[0] = input;                // put sample

        echoed=echoed_signal[k];

        //calculate output of adaptive filter
        af_output=dotp(filter, input_vector);

        // calculate error value
        error_signal = echoed-af_output;

        if(isNLP==1){
            error_signal= error_signal*0.01;
        }

        error_signal_array[k]=error_signal;

  decideDTD();

 //dtd_array[k]=isHold;

        if(isHold!=1){
                // calculate variable step size
                nstep_size=1/(dotp(input_vector, input_vector)+0.0001);

                //update tap weights
                for (i=0; i<FILTER_LENGTH; i++)
                {
                        filter[i] = filter[i] + nstep_size*error_signal*input_vector[i]; //calculate taps
                }
        }

        for (i=FILTER_LENGTH-1; i>=1; i--)
        {
                input_vector[i]=input_vector[i-1]; //shift vector
        }


        if(isHold==1){
                for(m=0; m<FILTER_LENGTH;m++){
                        filter[m]=filter_hold[m];
                }

        }else{
                for(m=0; m<FILTER_LENGTH;m++){
                        filter_hold[m]=filter[m];
                }
        }

                }//for loop end

}


// This is main procedure executed on start up of program
main()
{
         // Initialise variables
        error_signal=0.0;
        echoed=0.0;
        af_output=0.0;
        nstep_size=0;
        isNLP=0;

        for (i=0; i<FILTER_LENGTH; i++) // initialise filter, input vector
        {
                input_vector[i]=0.0;
                rxy[i]=0.0;
        }



        for (i=0; i<FILTER_LENGTH; i++) // initialise filter, input vector
        {
                filter[i]=filter_current[i];

        }


        doFiltering();


}
```