2001

Adaptive Notch Filter for Single and Multiple Narrow-Band Interference

Choy Chun Sin
Edith Cowan University

Edith Cowan University

Copyright Warning

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

DECLARATION

I certify that this thesis does not, to the best of my knowledge and belief:

i incorporate without acknowledgment any material previously submitted for a degree or diploma in any institution of higher education;

ii contain any material previously published or written by another person except where due reference is made in the text; or

iii contain any defamatory material.

Signature: 

Date: 28/4/2011

EDITH COWAN UNIVERSITY
PERTH WESTERN AUSTRALIA

BACHELOR OF ENGINEERING IN COMMUNICATION SYSTEM

ENS4241 ENGINEERING PROJECT 2

Adaptive Notch Filter for Single and Multiple Narrow-band Interference

SUBMITTED BY : CHOY CHUN SIN
STUDENT NO : 3992139

SUPERVISOR : Dr Dr Ganesh Kothapalli
Dr Amine Bermak
LOCAL SUPERVISOR : Dr Ma Zongming

ABSTRACT

In this project, the adaptive notch filter for single and Multiple narrow-band interference is implemented using simplified LMS algorithm.

Performances of the LMS adaptive algorithms is evaluated and analysed through simulation on the computer using MATLAB. The algorithm are then written in C programme and implemented using Texas Instrument Tool which consist of TMS320C54x EMV board and Code Composer Studio.

ACKNOWLEDGEMENT

The students would like to take this opportunity to express their gratitude and appreciation to the following persons who have helped in the project:

Dr Ganesh Kothapalli & Dr Amine Bermak, the project supervisors, for giving their advise and support through out the project.

Dr Ma Zhongming, the local project supervisor, for giving his advice and moral support.

Mr Goi Sio Peng and Mr Heng Kwee Tong of Centre for Wireless Communications for making efforts to share their knowledge. Their guidance and invaluable assistance were timely and much appreciated.

Finally, to everyone, who has in one way or another contributed to the project.

LIST OF FIGURES

TABLE OF CONTENTS PAGE

1 INTRODUCTION

1.1 BACKGROUND

Digital signal processing is one of the fastest growing fields in the last thirty years and it plays a major role in many application areas. The resulting digital signal processing systems are attractive due to their reliability, accuracy, small physical sizes, and flexibility.

One category of digital signal processing knows as adaptive signal processing which applications are increasing rapidly. Adaptive signal processing evolved from techniques developed to enable the adaptive control of time-varying system. It has gained a lot of popularity due to the advances in digital technology that have increased the computing capacities and broadened the scope of digital signal processing. The key difference between classical signal processing techniques and adaptive signal processing method is that the latter deal with time varying digital systems.

An adaptive system is a system whose coefficients could automatically adjust to changing environment or input signal. The transition from implementing digital filters with DSP microprocessors to implementing adaptive digital filters is a natural extension. Adaptive filters are used to their greatest advantage when there is an uncertainty about the characteristics of a signal or when the characteristics change during filter's characteristics no longer change.

Adaptive filter theory could be found applications in such field as biomedical engineering, seismology, astrology, navigation, communication, radar, sonar and control systems. They are used in the area of system identification, channel equalization, signal enhancement, and prediction.

The project will focus on the techniques of using adaptive notch filter in the single, and multiple interference noise cancellation application.

1.2 OBJECTIVES

In many systems, noise interference is a major problem that could cause the whole system to malfunction or inaccurate readings to be taken. Noise signals normally reside in higher frequencies in the frequency spectrum. So to remove the noise signal, the signal could be passed through a low pass filter so that the higher noise frequency could be removed. However, by doing this, the bandwidth of the signal would be effectively reduced. This is however not desirable as, if there are signals in the higher frequency range that is wanted, then they will be lost.

To overcome this problem, a notch filter could be implemented. The function of this notch filter is to cancel out only the noise frequency. That is, it is simply a band stop filter with a very narrow bandwidth with the notch frequency placed over the noise signal. This is illustrated in

Figure 1.1

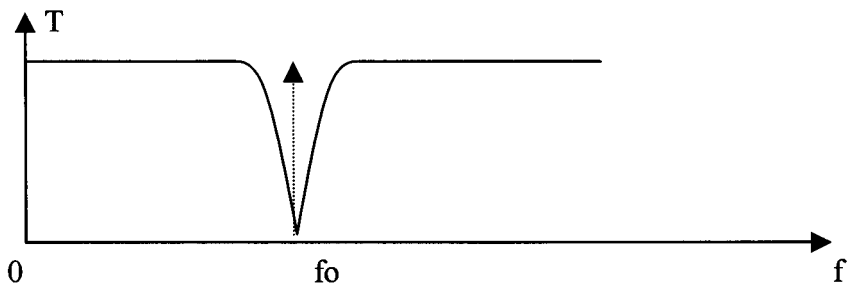

Figure 1.1 Frequency Response of a Notch Filter

As can be seen from the frequency response plot of the filter in Figure 1.1, the whole frequency spectrum of the signal is passed through except at the noise frequency. In this way, the bandwidth of the signal is preserved and only the noise component is being removed.

Also if the noise frequency shift up and down, the notch filter must be able to adapt to the shift. That is, the filter must be able to track the noise and cancel it out. This is important feature in noise cancellation as most noise signals are not stationary at a frequency.

The objective of this project is to implement this adaptive notch filter digitally to cancel out single and multiple narrow-band noise in a wide-band signal. It will be implemented on the TMS320C54X Evaluation Module board and demonstrated on audio signal.

1.3 PLANING

1.3.1 PHASE I

The schedule was planned base on weekly basis. The duration to complete a task will vary from one to four weeks depends on its complexity.

The following schedule was planned before the project was started.

		August				September				October				
ID	Task Name	wk 1	wk 2	wk 3	wk 4	wk 1	wk 2	wk 3	wk 4	wk 1	wk 1	wk 2	wk 3	wk 4
	PHASE I													
1	Project Proposal													
2	Pending for approval													
3	Received of approval													
4	Sourcing for notes, books and informations													
5	Theoertical Studies													
6	Design Approach													
7	Report writing													
8	Final report submission													

Figure 1.2 Project Phase I Schedule

1.3.2 PHASE II

In project II, the schedule was planned according to project I on weekly basis. Due to short time frame, the duration has slightly reduce from one to two weeks depend on the complexity.

		December				January				February
ID	TASKS	wk1	wk2	wk3	wk4	wk1	wk2	wk3	wk4	wk1
1	Studies the user guides of development tool									
2	Trying to explore the hardware & the software dubugger									
3	Disussion on the software approach & assign the task									
4	Hardware testing via PC									
5	Self study on the individual Algorithm									
6	Draft report & final simulation									
7	Final report									

Figure 1.3 Project Phase II Schedule

2 BASIC THEORY

2.1 ADAPTIVE FILTER

Adaptive filters are considered non-linear systems, therefore their behaviour analysis is more complicated than for fixed filters. On the other hand, because the adaptive filters are self designing filters, from the practitioner's point of view their deign can be considered less involved than in the case of digital filters with fixed coefficients.

The general set up of adaptive filtering environment is illustrated in Figure 2.1, where k is the iteration number, x(k) denotes the input signal, y(k) is the adaptive filter output signal, and d(k) defines the desired signal. The error signal e(k) is calculated as d(k) – y(k). The error signal is then used to form a performance function that is required by the adaptation algorithm in order to determine the appropriate updating of the filter coefficients. The minimisation of the objective function implies that the adaptive filter output signal is matching the desired signal in some sense.

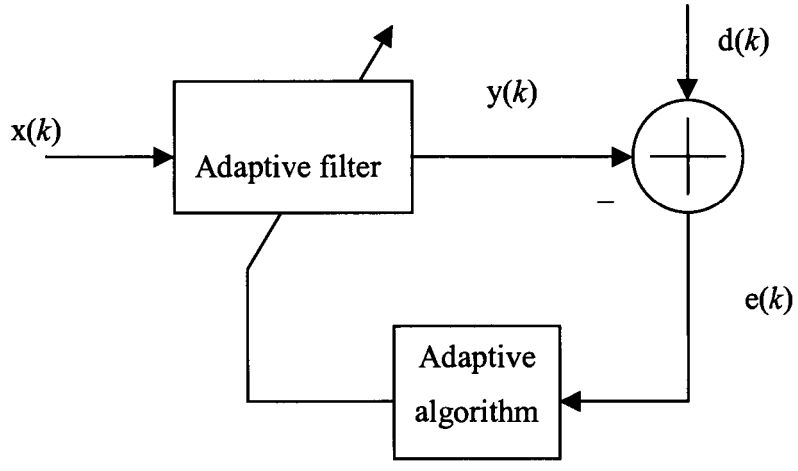

Figure 2.1 General adaptive filter configuration

The specifications of an adaptive system consists of the following items :

2.1.1 APPLICATION

The type of application is defined by the choice of the signals acquired from the environment to be the input and desired-output signals. Examples are echo cancellation, channels equalization, system identification, signal enhancement, adaptive beamforming, noise canceling and control.

2.1.2 ADAPTIVE FILTER STRUCTURE

The adaptive filter can be implemented in a number of different structures or realizations. The choice of structure can influence the computational complexity (amount of arithmetic operations per iteration) of the process and also the necessary number of iterations to achieve a desired performance level.

Basically, there are two major classes of adaptive digital realizations :

2.1.2.1 Finite-duration Impulse Response (FIR) filter

The most widely used adaptive FIR filter structure is the transversal filter which implements an all-zero transfer function with a canonic direct from realization without feedback. The output of the FIR filter can be described by the following equation :

$$y(n) = \sum_{k=0}^{N} b_k x(n-k) \quad \text{.....2.1}$$

The equation represents a weighted sum of the present and past input to the filter with b_k is the weight of the (N+1) order filter. The equation may also be represented by the network structure as shown in Figure 2.2

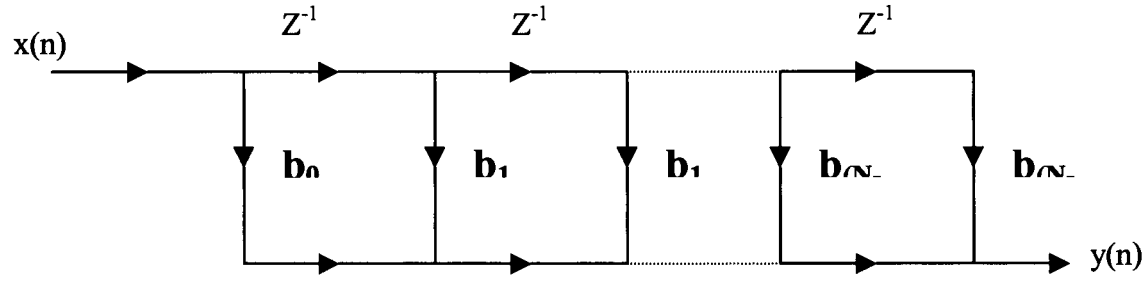

Figure 2.2Network structure representation of FIR filter

For this realization, the output signal is a linear combination of the filter coefficients, that yields a quadratic mean square error (MSE) function with a unique optimal solution. FIR typically required large filter orders to obtain satisfactory sharp cutoff characteristics and hence increase in computational complexity.

2.1.2.2 Infinite-duration Impulse Response (IIR) Filter

The most widely used adaptive IIR filter structure is the canonic direct-form realization, due to its simple implementation and analysis The output of the IIR filter can be described by the following equation :

$$y(n) = \sum_{k=1}^{N} a_k\, y(n-k) + \sum_{k=0}^{M} b_k\, x(n-k) \qquad \text{.....2.2}$$

Equation (2,2) shows that the output of the filter is a weighted sum of the present and past input of the filter, as well as past values of the output of the filter. The a_k and b_k are the weights of the filter and N & M is the order of the filter. A network structure representation of a second order IIR filter is illustrated in Figure 2.3 :

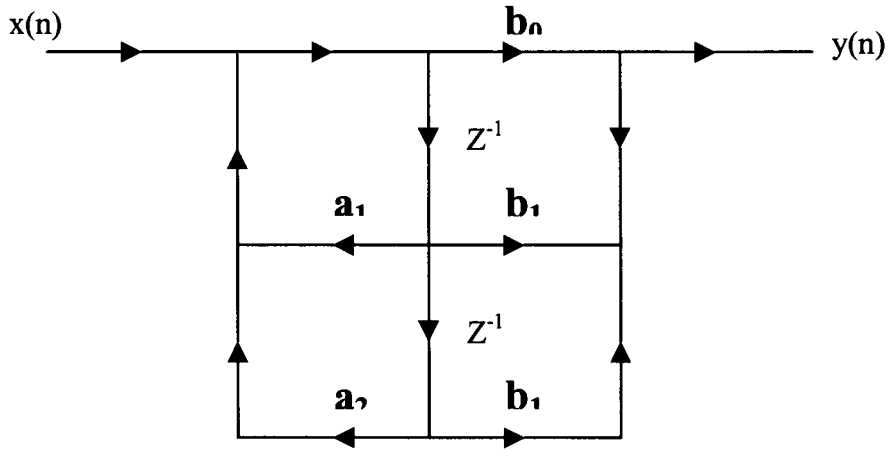

Figure 2.3 Network structure representation of a 2nd order IIR filter

Sharp cutoff and constant magnitude response can be obtained using second-order filter sections. However, there are some inherent problems related to recursive adaptive filters which are structured dependent, such as pole-stability monitoring requirement and slow speed of convergence.

2.1.3 ALGORITHM

The algorithm is the procedure used to adjust the adaptive filter coefficient in order to minimize a prescribed criterion. The algorithm is determined by defining the search method (or minimization algorithm), the objective function, and the error signal nature. The choice of algorithm determines several crucial aspects of the overall adaptive process, such as existing of sub-optimal solutions, biased optimal solution, and computational complexity.

2.2 ADAPTIVE NOISE CANCELLATION

The basic concept of adaptive noise cancellation reduces the level of undesired noise by using adaptive filtering techniques with an auxiliary reference signal. This technique takes advantage of the correlation between the noise contaminating the desired signal and a noise-alone reference signal. Figure 2.4 shows the block diagram of an adaptive noise cancellation system :

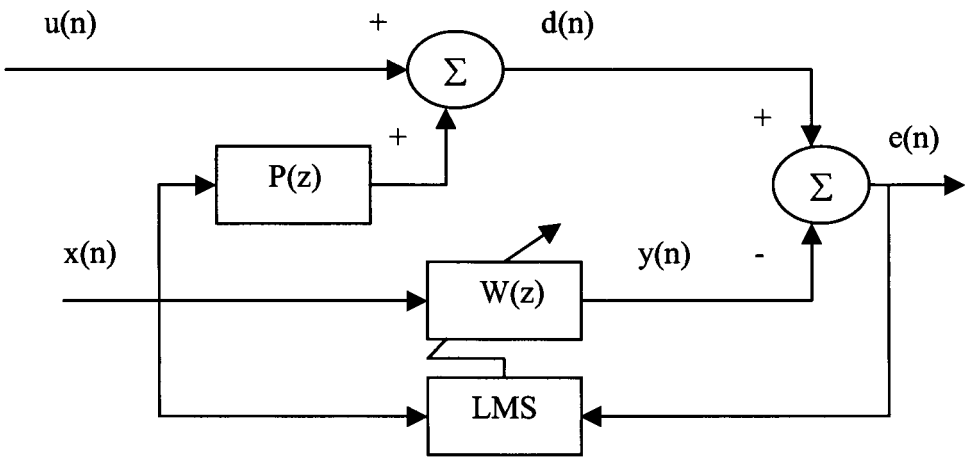

Figure 2.4 Block Diagram of adaptive noise chancellor

The chancellor has two inputs, the primary input d(n) and the reference input x(n). The primary input d(n) consists of desired signal u(n) plus noise, which is derived from x(n) through the linear filter P(z). The reference input consists of noise x(n) alone.

The objective of the adaptive filter is to use the reference input x(n) to estimate the noise component of d(n) using adaptive filter W(z). The adaptive filter output y(n) is

then subtracted from the primary channel signal d(n), producing e(n) as the desired signal plus a small amount of residual noise.

The area of focus for the project is to deduce a method on reducing sinusoidal interference of an audio signal. The conventional method of eliminating such sinusoidal interference is through the use of a notch filter tuned to the frequency of the interference. A very narrow notch is usually desired in order to filter out interference without seriously distorting the signal of interest.

An adaptive notch can be realised by using an adaptive noise canceler with a sinusoidal reference signal. The advantages of adaptive notch filter are that it offers easy control of bandwidth, an infinite null, and the capability to adaptively track the exact frequency of interference.

2.3 ADAPTIVE ALGORITHMS

Adaptive algorithms are used to adjust the coefficients of the digital filter such that the error signal, e(k), is minimised according to some criterion. There are varies of algorithms used in digital filter applications, examples, Least Mean Square Algorithm (LMS), Normalised LMS Algorithm (NLMS), Variable Step Size LMS Algorithm (VSLMS) , Normalised Variable Step Size Algorithm (VSNLMS) ,Recursive Least Square Algorithm (RLS) and RLS-QR Algorithm (RLS-QR) etc. The most common algorithms that have found widespread application are the Least Mean Square Algorithm (LMS) and Recursive Least Square Algorithm (RLS). In terms of computation and storage requirements, the LMS algorithm is the most efficient. Further, it does not suffer from the numerical instability problem inherent in the other two

algorithms. For these reasons, the LMS algorithm has become the first choice in many applications. A more detail information of the basic LMS adaptive algorithm was introduced.

2.3.1 BASIC LEAST MEAN SQUARE (LMS) ADAPTIVE ALGORITHM

In LMS algorithm, the coefficients are adjusted from sample to sample in such a way as to minimise the Mean Square Error (MSE).

The LMS is based on the steepest descent algorithm where the weight vector is updated from sample to sample as follows:

$$W_{k+1} = W_k - \mu * \Delta_k \quad \text{---------- (2.3)}$$

where W_k and Δ_k are the weight and the true gradient vectors, respectively, at the kth sampling instant. μ controls the stability and rate convergence.

The LMS algorithm is a practical method of obtaining estimates of the filter weights W_k in real time. The widrow-hopf LMS algorithm for updating the weights from sample to sample is given by

$$W_{k+1} = W_k + 2*\mu * e_k * \Delta_k \quad \text{---------- (2.4)}$$

Where:

$$e_k = yk - \Sigma w_k(i)x_{k-I}$$

Clearly, the LMS algorithm above does not require prior knowledge of the signal statistics, but instead uses their instantaneous estimates. The weights obtained by the LMS algorithm are only estimates, but these estimates improve gradually with time as the weights are adjusted and the filter learns the characteristics of the signals. Eventually, the weights converge.

The condition for convergence is:

$$0 < \mu > 1/\lambda_{max}$$

where λ_{max} is the maximum eigen value of the input data covariance matrix. In practical, W_k never reaches the theoretical optimum $W_{optimum}$, but fluctuates about it as shown on Figure 2.5

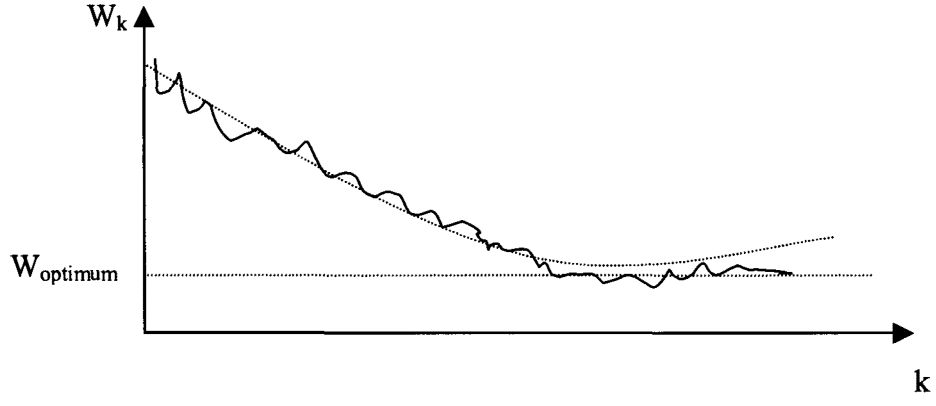

Figure 2.5 Weight adaptation, W_k vs k

3 PROJECT APPROACH

3.1 NOTCH FILTER TRANSFER FUNCTION

We have decided to use IIR filter structure and LMS algorithm to design and implement the adaptive notch filter after going through the literature studies. The initial task is to obtain the transfer function of the notch filter.

With the constraints and characteristics of a notch filter lead to a direct realisation of transfer function from the poles and zeros diagram. From the pole-zero plot of notch filter shown in Figure 3.1, it converges to a polynomial transfer function with zeros on the unit circle whose angular locations correspond to the signal frequencies(w). r is a constant which is slightly less than 1 so that the poles lie in the stability domain. It also determines the relationship between the distance of the pole from the unit circle and the normalised bandwidth.

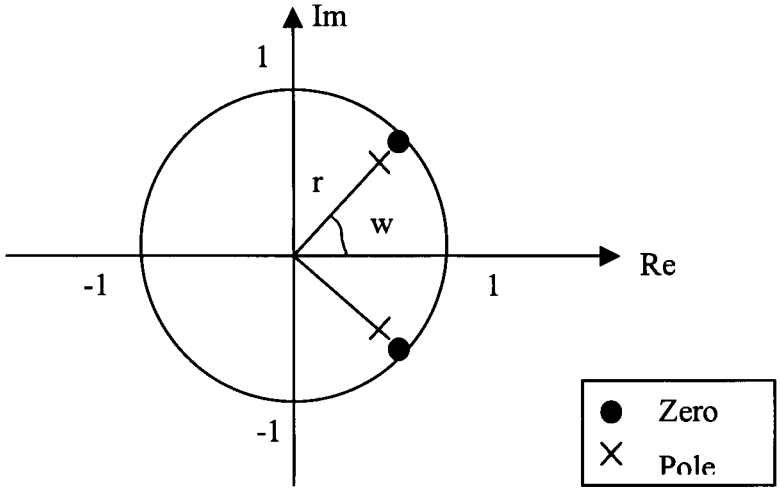

Figure 3.1 Poles and zeros plot of notch filter

With all the condition stated as above we could derive the transfer function of the adaptive notch filter :

$$H(z) = Y(z) / X(z)$$

$$= (1-e^{jw}z^{-1})^* (1-e^{-jw}z^{-1}) / (1-re^{jw}z^{-1})^* (1-re^{-jw}z^{-1}) \qquad |r| < 1$$

$$= 1 - z^{-1}(e^{jw} + e^{-jw}) + z^{-2} / 1 - r z^{-1}(e^{jw} + e^{-jw}) + r^2 z^{-2} \quad |r| < 1$$

$$= 1 - z^{-1}(2\cos w) + z^{-2} / 1 - r z^{-1}(2\cos w) + r^2 z^{-2} \qquad |r| < 1$$

$$= 1 - az^{-1} + z^{-2} / 1 - arz^{-1} + r^2 z^{-2} \qquad |r| < 1 \quad \text{----------(3.1)}$$

where

$$a = 2\cos w$$

In measuring the performance of the filter, the stability of the filter is very important. It is critical that the position of the poles of the filter remains within the circle to ensure stability of the output of the filter. If however the position of the poles were allowed to drift out of the unit circle radius in the pole-zero plot, then the output of the filter will become unstable.

By taking the inverse Z-transform of the transfer function, it is easy to show that the output of the filter is given by:

$$y(n) = x(n) - a(n)x(n-1) + x(n-2) + a(n)ry(n-1) - r^2y(n-2) \text{ ---------- } (3.2)$$

Comparing equation (3.2) with equation (2.2) mentioned earlier, it could be noted that they are similar. That is, it is a summation of the weighted pasted and present input values as well as weighted past output values. Thus it has the same structure as IIR filter.

3.2 LMS ADAPTIVE ALOGRITHM

The Least Mean Square algorithm is chosen due to its simplicity and ease of computation. In addition, it does not require repetition of data. Thus it is generally the best for many different application of adaptive signal processing.

Generally, the LMS algorithm is applied in transversal tapped (FIR) filters with the structure shown in Figure 2.2. But it can be extended to IIR filters with the structure shown in Figure 2.3.

3.2.1 ADAPTATION OF PARAMETER *a*

In this project, we are only concern with second order section with one weight. Thus this will simplify the understanding of the algorithm. The algorithm is based on the steepest-descent gradient search. The equation used here *will be modified* from the one introduced by Widrow –hopf. Here we are trying to minimise the mean-squared output energy level instead of minimizing the mean-squared error.

This is given by:

$$a(n+1) = a(n) - \mu [dy(n)^2/da(n)] \quad \text{---------- (3.3)}$$

Differentiating the mean-squared output w.r.t a(n)

$$a(n+1) = a(n) - 2 \mu y(n) [dy(n)/da(n)] \quad \text{---------- (3.4)}$$

from equation (2)

$$[dy(n)/da(n)] = -x(n-1) + ry(n-1)$$

Substituting into equation (3)

$$a(n+1) = a(n) - 2 \mu y(n) [-x(n-1) + ry(n-1)] \quad \text{---------- (3.5)}$$

The above equation (3.5) is the LMS algorithm for this adaptive notch filter. Here μ is the gain constant that regulates the speed and stability of adaptation.

Since we are dealing with real time processing, we need a relatively fast way of reaching the minimum. The way that the search is conducted is to set the weight of the

filter, that is a(n) in equation (3.2), to some initial arbitrary value. And it will be adjusted in a stepwise fashion until the minimum is reached. This is shown in Figure 3.2

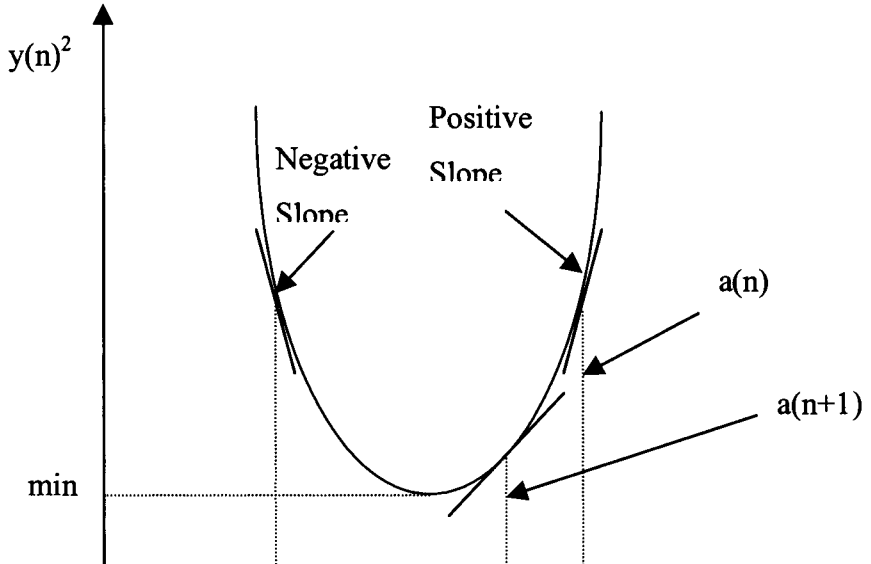

Figure 3.2 Search for minimum with stepwise fashion

When making a step, the size and direction must be considered. Each step will consist of an increment to the weight. If the current value of the weight is to the right of the minimum, then the step will be negative. If the current weight is to the left of the minimum, then the step size will be positive.

Thus the negation of the derivation indicated the proper direction of increment. Since the derivative vanishes at the minimum, it can be used to adjust the step size. And it can be observed that the step size and direction is proportional to the negative of the derivative as in equation (3.3). And repeated iteration of equation (3.3) will cause a(n) to move by steps from its initial value until it reaches the minimum.

And since the weight changes at each iteration are based on imperfect gradient estimates, it is expected that the adaptive process to be noisy, i.e it would not follow the true line of steepest descent on the performance surface.

With the equation obtained in equation (3.5), we can see the simplicity and ease of computation of LMS algorithm. It allows the weight of the filter to be updated averaging, squaring or differentiating.

3.2.2 ADAPTATION OF PARAMETER *r*

The stability of the filter is maintained by constraining the poles of the filter within the unit circle. And to achieve a narrow notch, the poles are kept as close to the unit circle as possible. However, by doing so, because the LMS algorithm adapt by searching the gradient of squared output, it would be less sensitive to noise signals that are far away from the notch.

To be able to track noise signal that is far away from the notch, the parameter, r , has to be adapted as well. The idea to start off with a wide notch so that the noise signal can be detected and then to reduce the width of the notch to the narrow notch as before. This is depicted in Figure 3.3 and Figure 3.4.

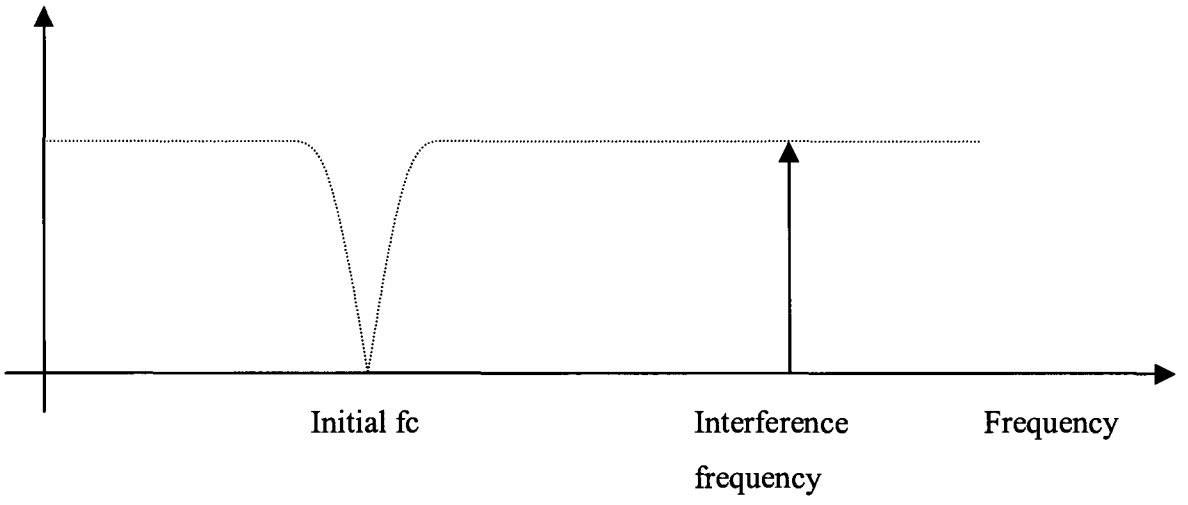

Figure 3.3 Initial state of adaptation

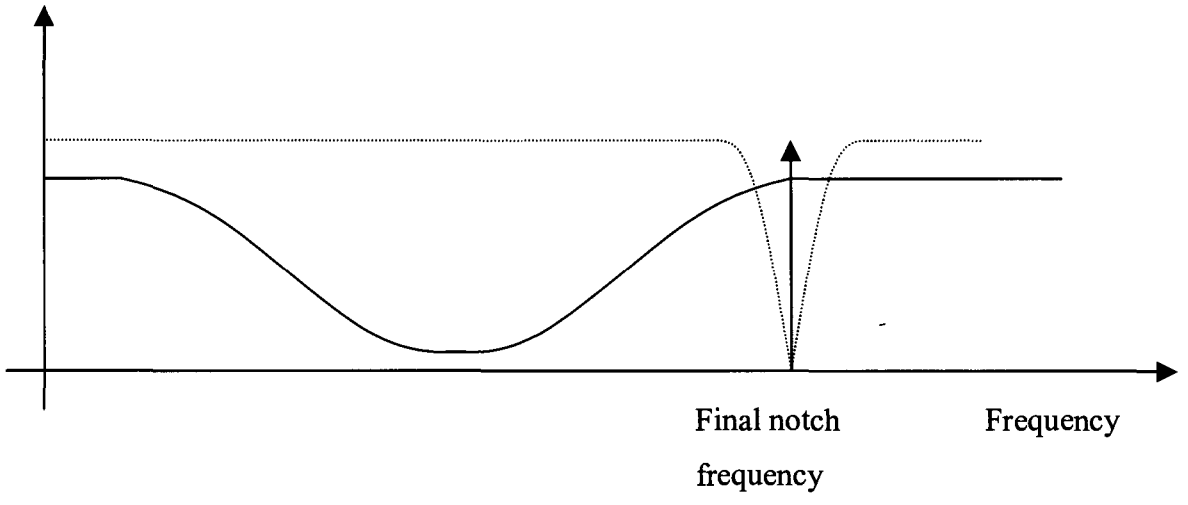

Figure 3.4 Final adaptation notch with parameter a & r

3.3 CASCADE ADAPTIVE NOTCH FILTER

The cascade section of the adaptive IIR notch filter is implemented by having two section of the basic notch filter as shown in the following figures. The adaptation is done at the end of the section.

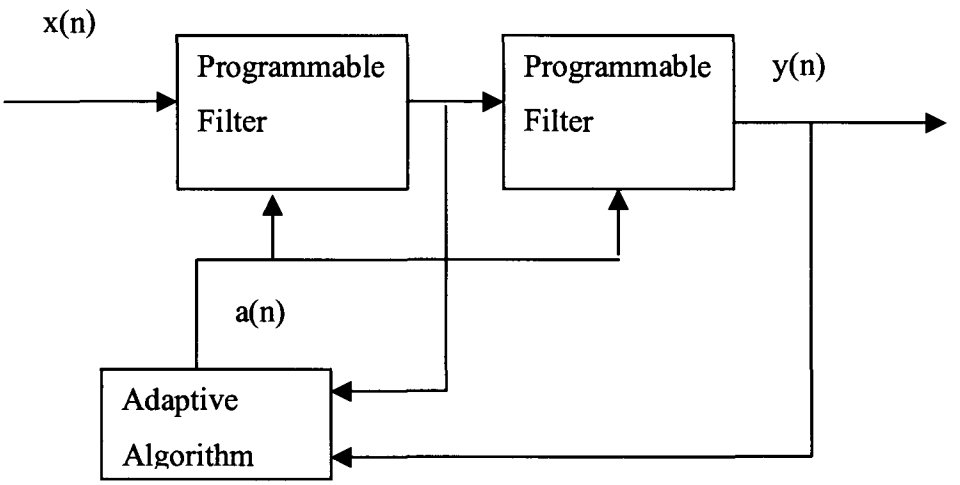

Figure 3.5 Cascade section of the adaptive filter

This cascading of two sections together allows the filter to have a narrower notch and yet retain its stability by constraining the poles in both sections well within the unit circle.

3.4 MULTIPLE CHANNEL ADAPTIVE NOTCH FILTER

The multiple adaptive IIR Notch filter will be extended to cancel out multiple sinusoidal interference. It is similar to cascade filter but with individual adaptive algorithm as shown in the figure.

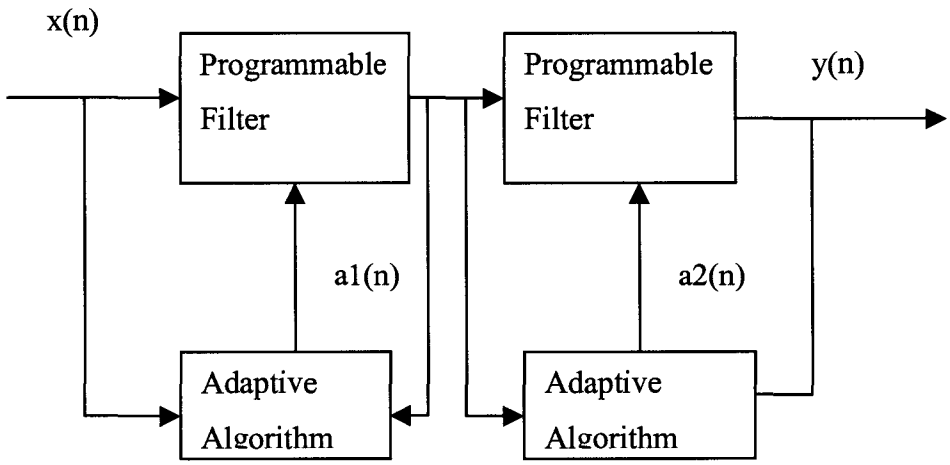

Figure 3.6 Multiple interference adaptive Notch filter

The frequency response of the multiple notch has the form shown in Figure 3.7

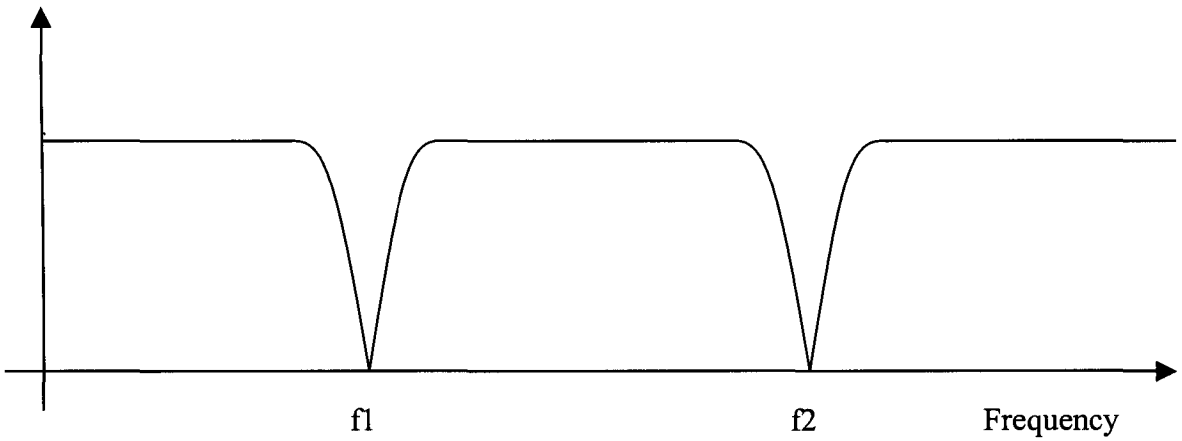

Figure 3.7 Frequency response of multiple interference adaptive notch filter

3.5 IMPLEMENTATION APPROACH

The derived simulation will be simulated using Matlab for windows Version 5.3.1 for testing the feasibility of the adaptive algorithm. In the initial approach, TMS320C5x DSK board was used as the development tool but it was replaced by TMS320C54x EMV board with Code Composer Studio in the later phase. The TMS320C54x EMV board and the Code Composer Studio will be discussed in Chapter 5.

An analogue signal will be injected to the analogue input connect of TMS320C54x EMV board and the analogue output will be monitored.

4 SIMULATION AND DISCUSSION

In this chapter, the theoretical approach of the adaptive notch filters, which had been derived from previous chapter, will be investigated using MATLAB software. Both of the basic and adaptive notch filter will be analysed and discussed. The MATLAB Programs for all the simulations are attached in Appendix A.

4.1 BASIC IIR NOTCH FILTER

The transfer function of the notch filter has been derived in the previous chapter as shown in equation (3.1).

$$H(z) = 1-az^{-1} + z^{-2}/1-arz^{-1} + r^2z^{-2}$$

Where $a=2\cos\omega$ and $|r| <1$

There are a few parameters namely the poles and zeros, parameter a, r and sampling frequency which will affect the performance of the filter. Figure 4.1 shows the frequency response of the transfer function with sampling frequency of 48KHz, notch frequency of 4KHz and r=0.95

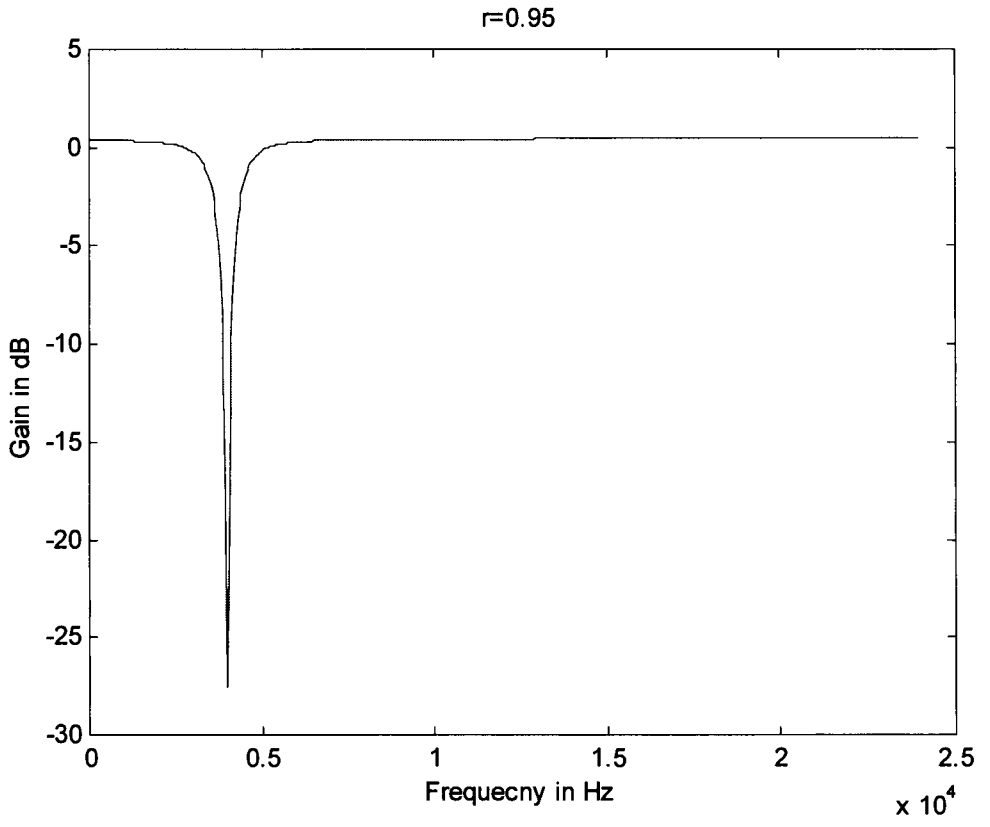

Figure 4.1 Frequency Response of the Basic Notch Filter

4.1.1 POLES AND ZEROS

In measuring the performance of the filter, the stability of the filter is very important. It is critical that the positions of the poles remain within the unit circle to ensure stability of the filter. If the position of the poles drift out of the circle radius, the output will be unstable. It is illustrated in the following figures.

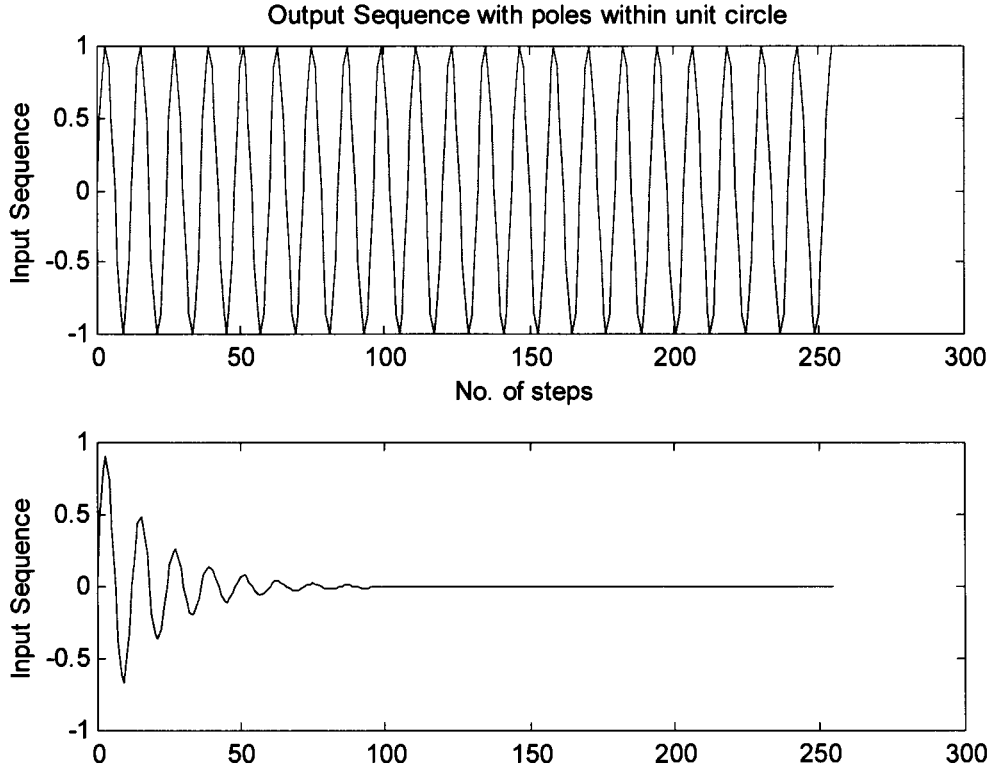

Figure 4.2 Output response of the notch filter with poles within the unit circle

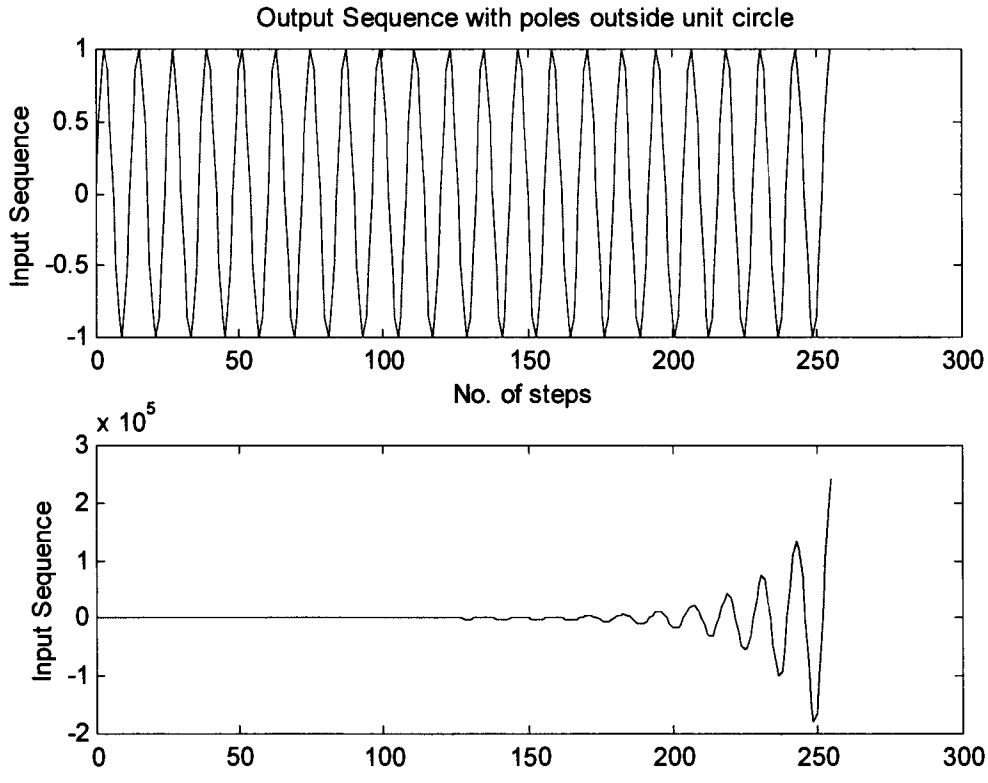

Figure 4.3 Output response of the notch filter with poles outside the unit circle

As shown in Figure 4.3, with the poles of the transfer function outside the unit circle, the output response of the filter to a sine wave input is a noisy signal.

4.1.2 RELATION BETWEEN A AND NORMALISED BANDWIDTH

The relation between the frequency varying parameter, a and the normalised bandwidth is given in Figure 4.4

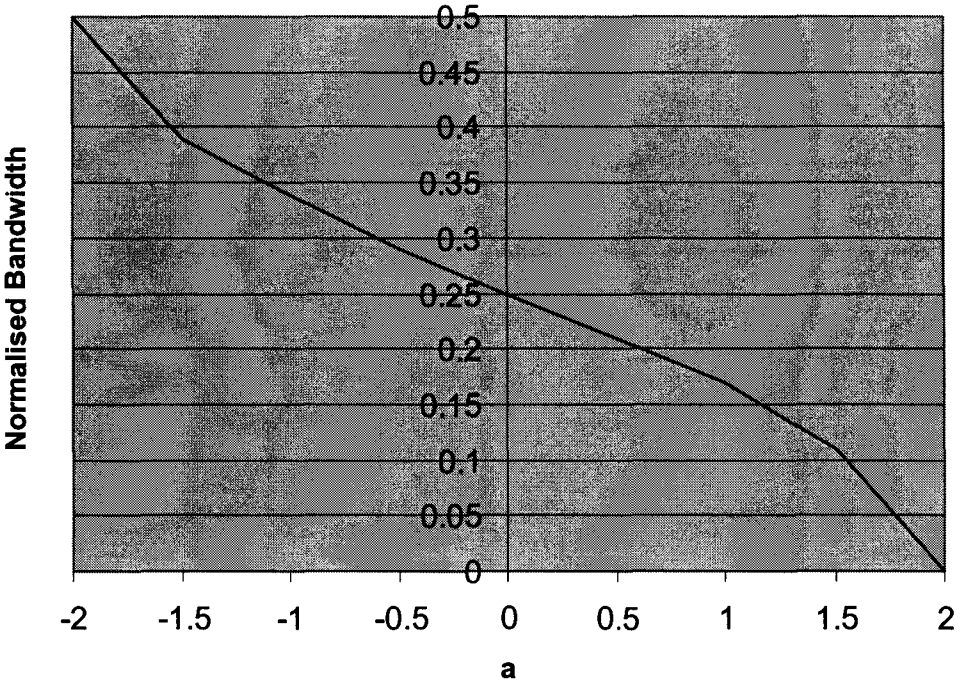

Figure 4.4 Relation between a and normalised bandwidth

4.1.3 RELATION BETWEEN R AND NORMALISED BANDWIDTH

The parameter, r determines the distance of the pole from the unit circle and the normalised bandwidth. The following figures are plots of frequency response of different values of r.

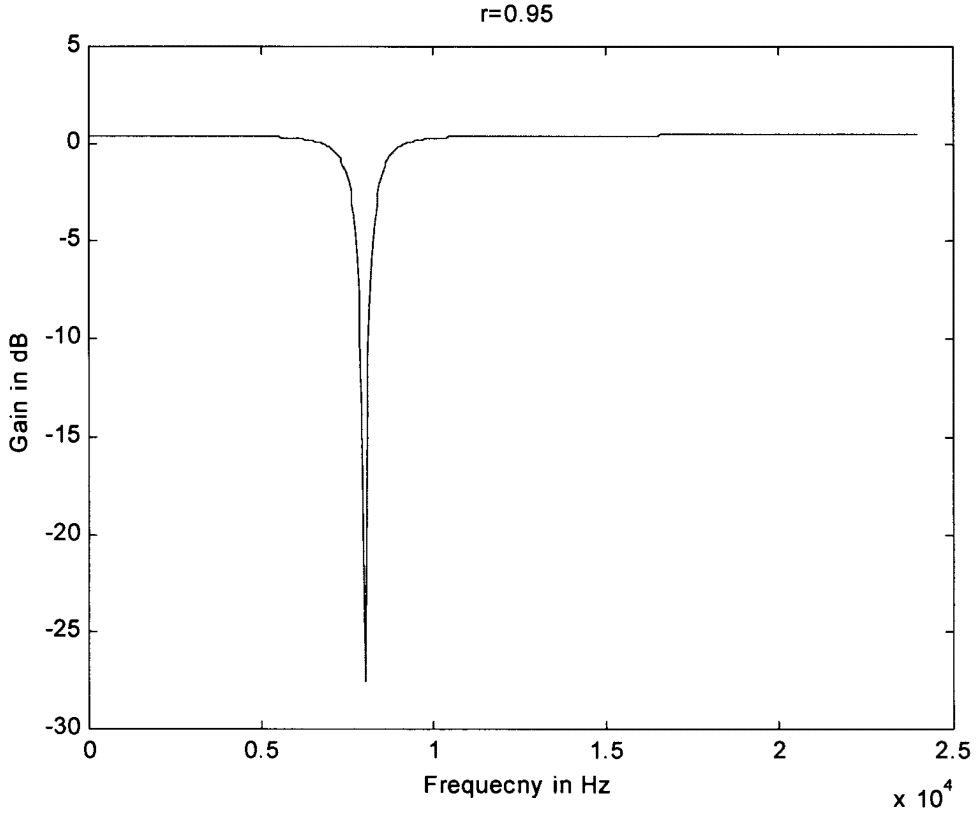

Figure 4.5 Frequency Response with r =0.95

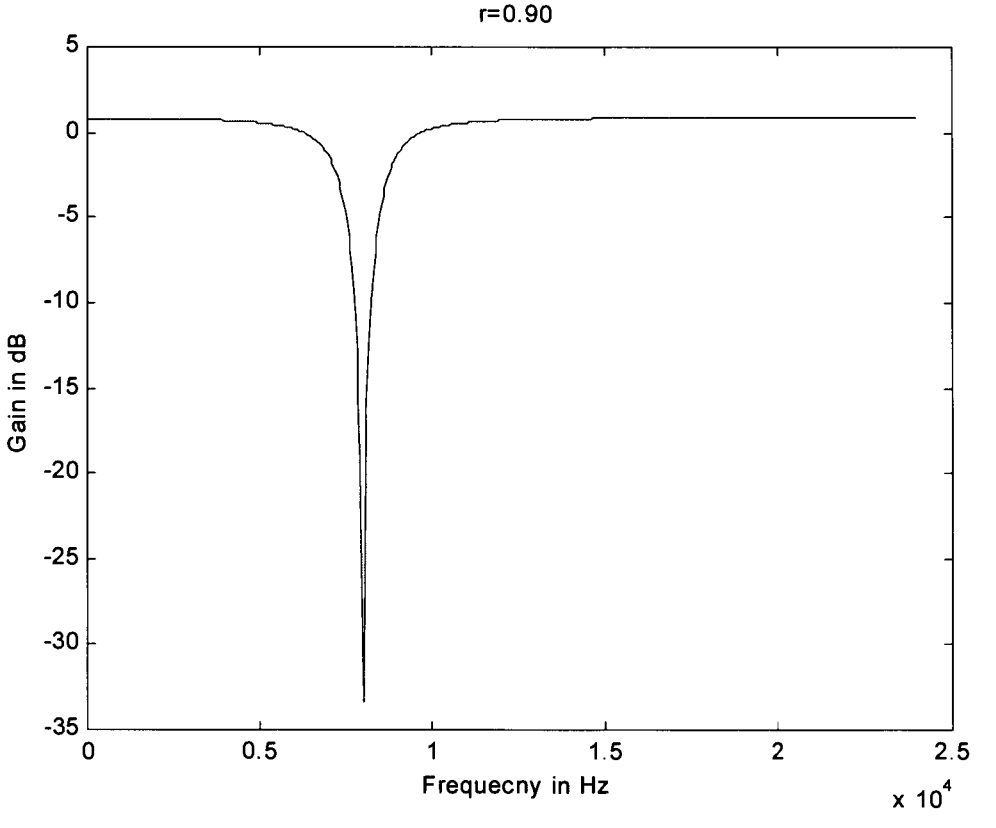

Figure 4.6 Frequency Response with r =0.90

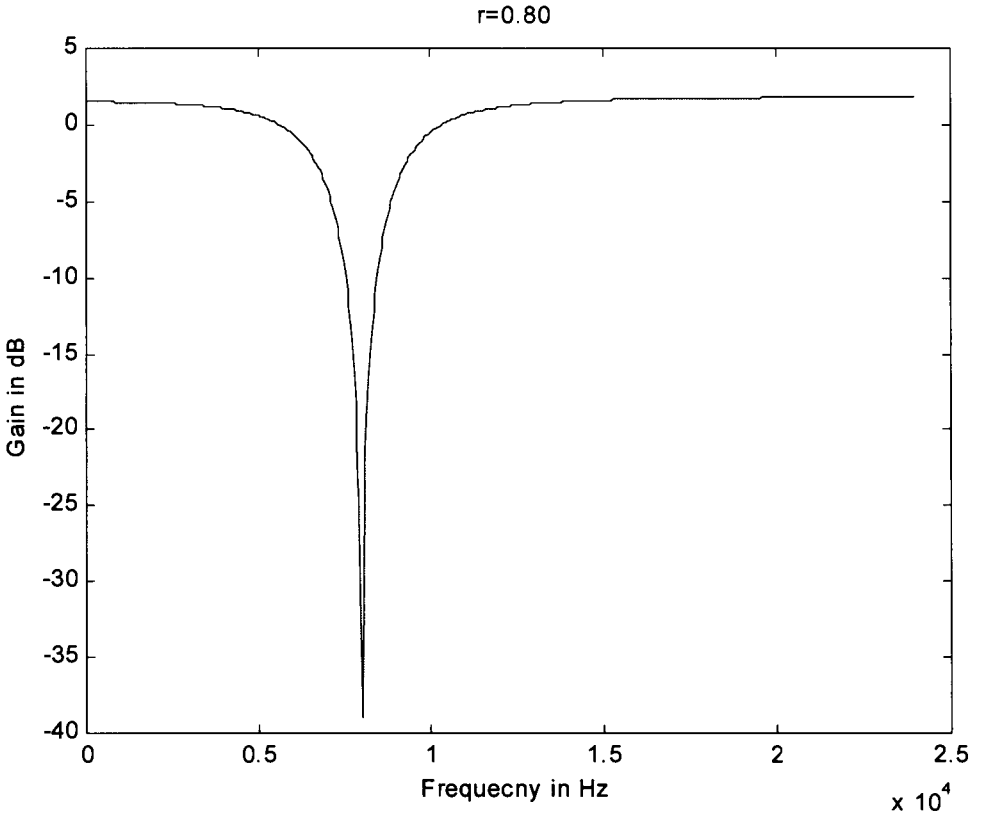

Figure 4.7 Frequency Response with r =0.80

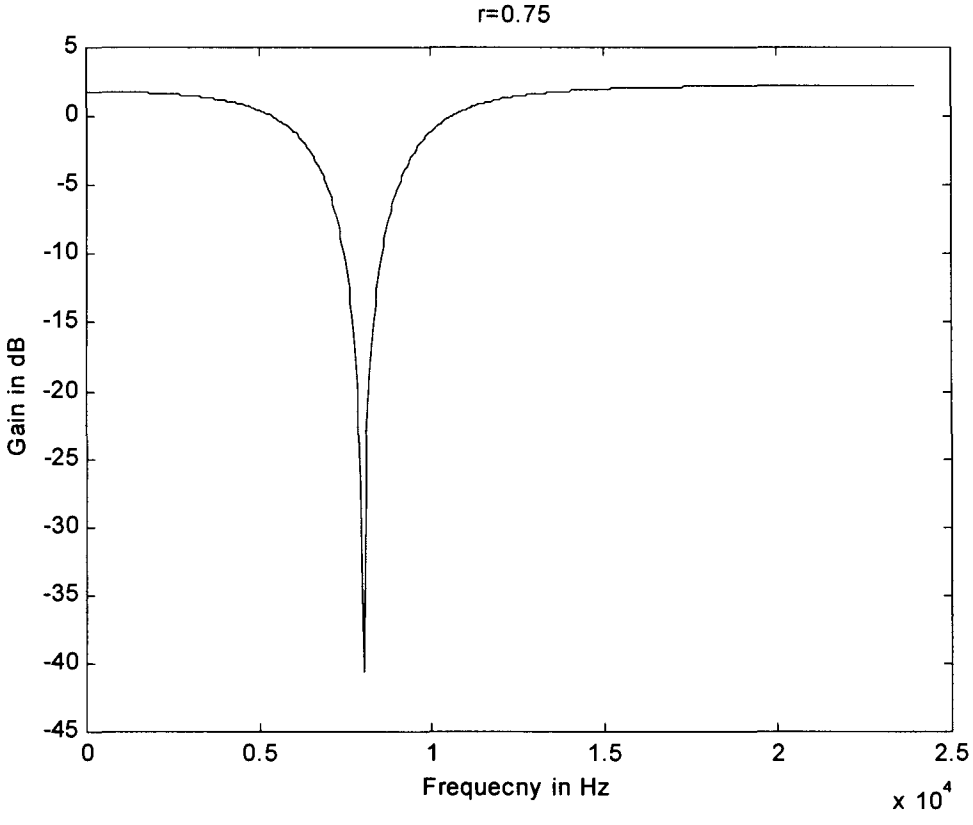

Figure 4.8 Frequency Response with r =0.75

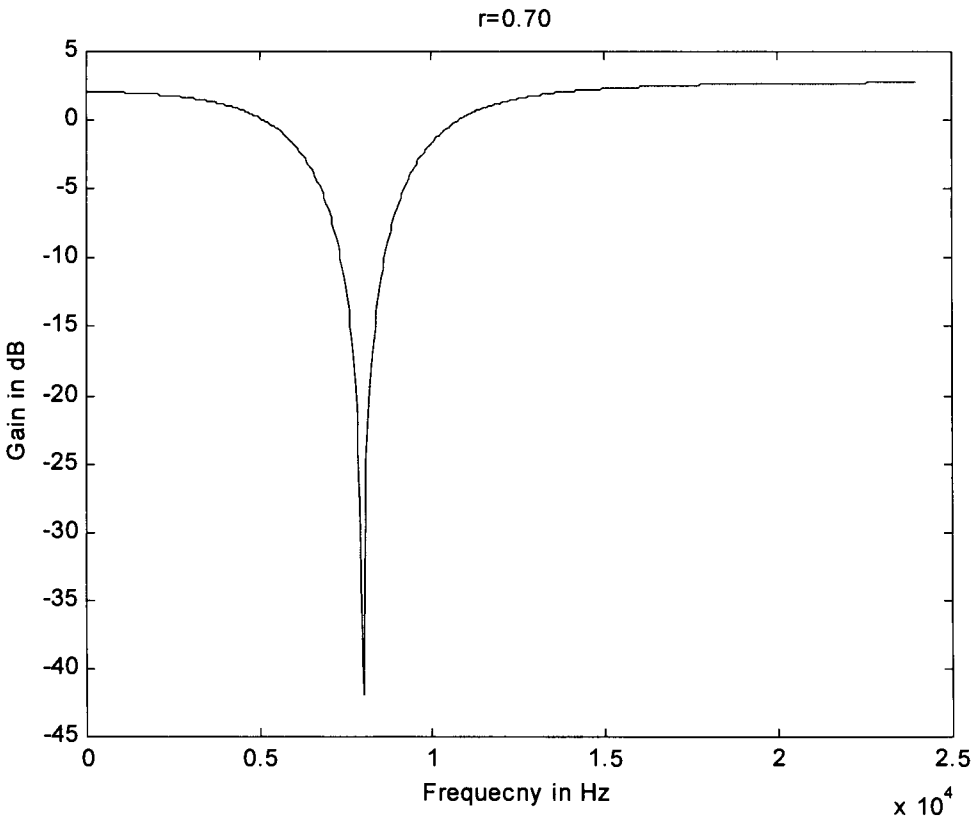

Figure 4.9 Frequency Response with r =0.70

Figure 4.10 show a relation between r and the normalised bandwidth. From the figures, it can be seen that the parameter, r is inversely proportional to the bandwidth of the notch filter.

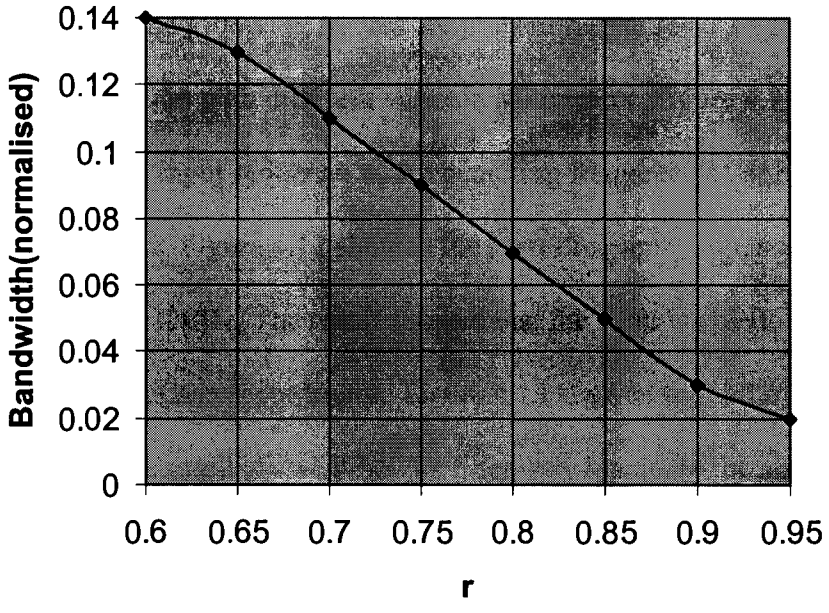

Figure 4.10 Relation between parameter r and normalised bandwidth

4.2 ADAPTIVE IIR NOTCH FILTER

The section discussed on the performance of the single channel adaptive notch filter. The theoretical adaptive equation were implemented using MATLAB and the results were simulated. The following figures are plots of output results by with the application of adaptive algorithm with different variable parameters. Figure 4.11 showed a single frequency (8KHz) input sequence with a sampling frequency of 48KHz and notch frequency of 8KHz. Figure 4.12 showed the same frequency input sequence with a frequency drift of 100Hz. The result showed that the filter is able to adapt the change of frequency and response accordingly. Figure 4.13 showed the same filter with a multiple channel input. The result showed that the filter only attenuate the notch frequency in the input signals.

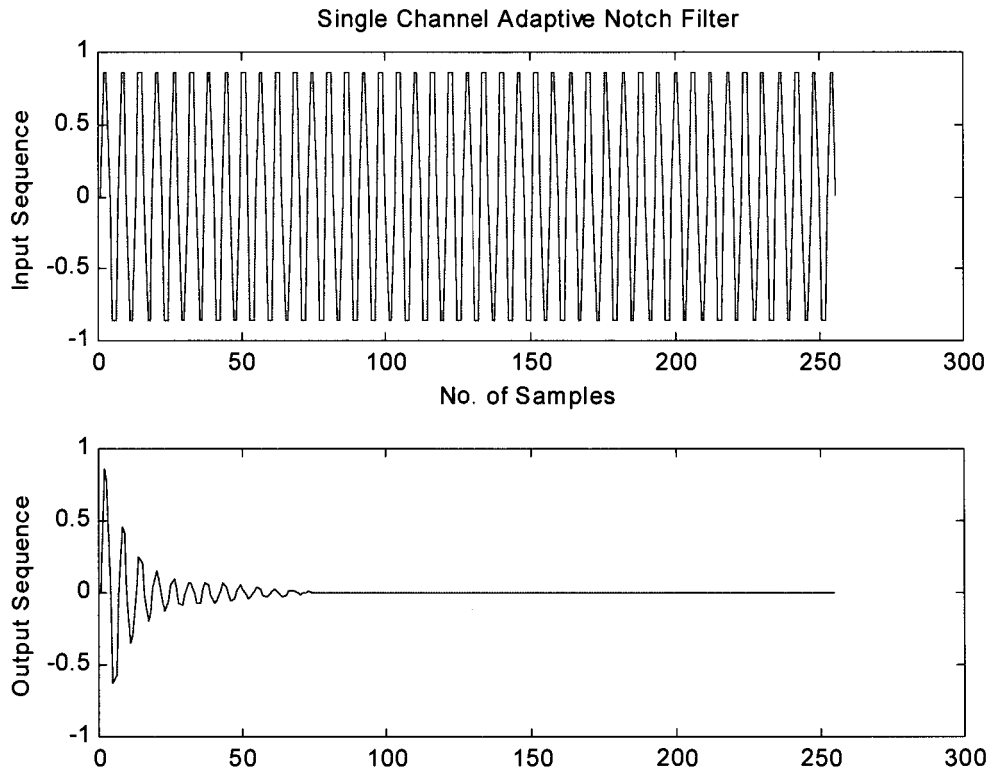

Figure 4.11 Output Response of Adaptive Notch Filter

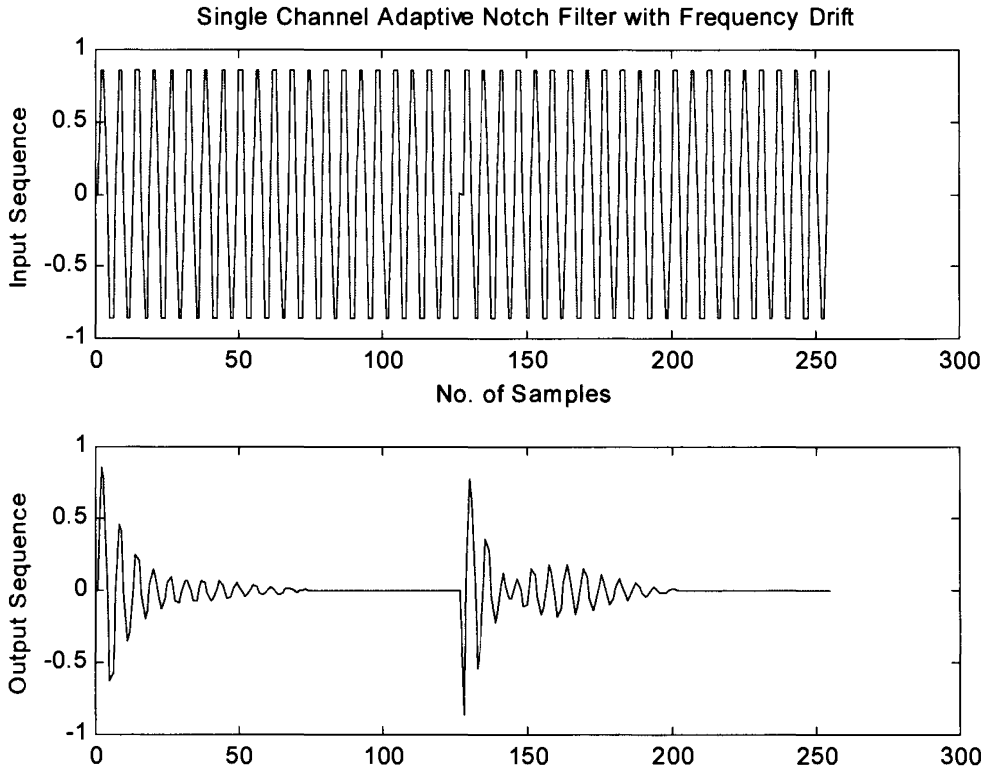

Figure 4.12 Output Response of Adaptive Notch Filter with Frequency Drift

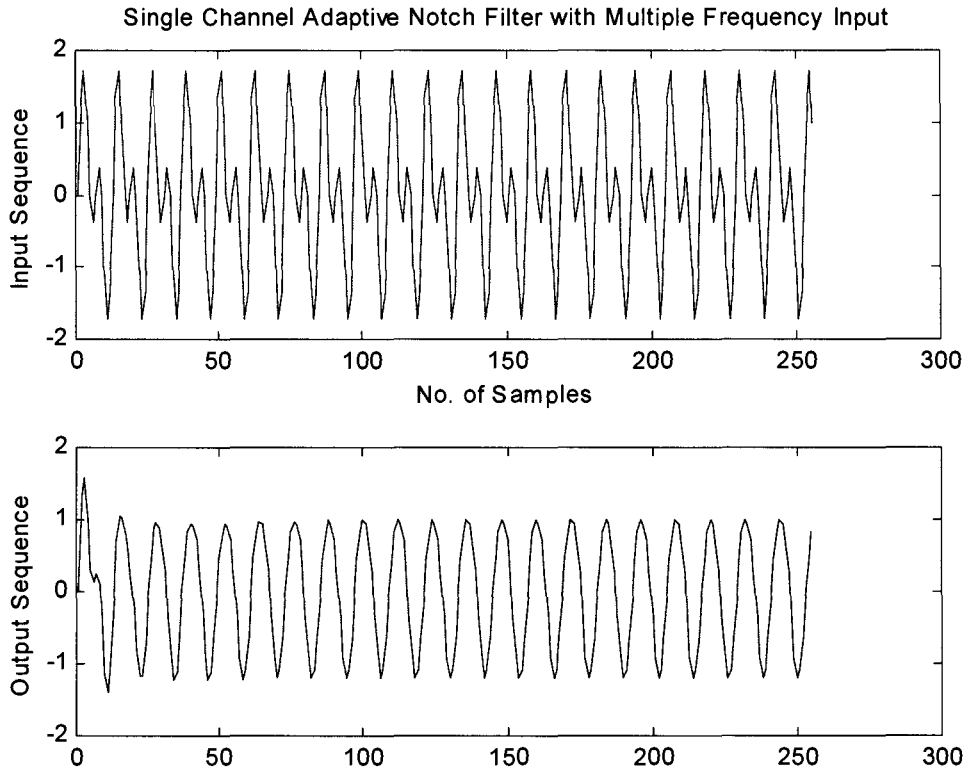

Figure 4.13 Output Response of Adaptive Notch Filter with Multiple Frequency Input

4.2.1 THE PARAMETER u

The parameter u as shown in the adaptive algorithm equation is the step size which determined the speed and stability of the adaptation. The following few figures are the plots of output sequence with same notch frequency and different step size. The input is a 8KHz sine wave and shift to 8.1KHz after 256 samples.

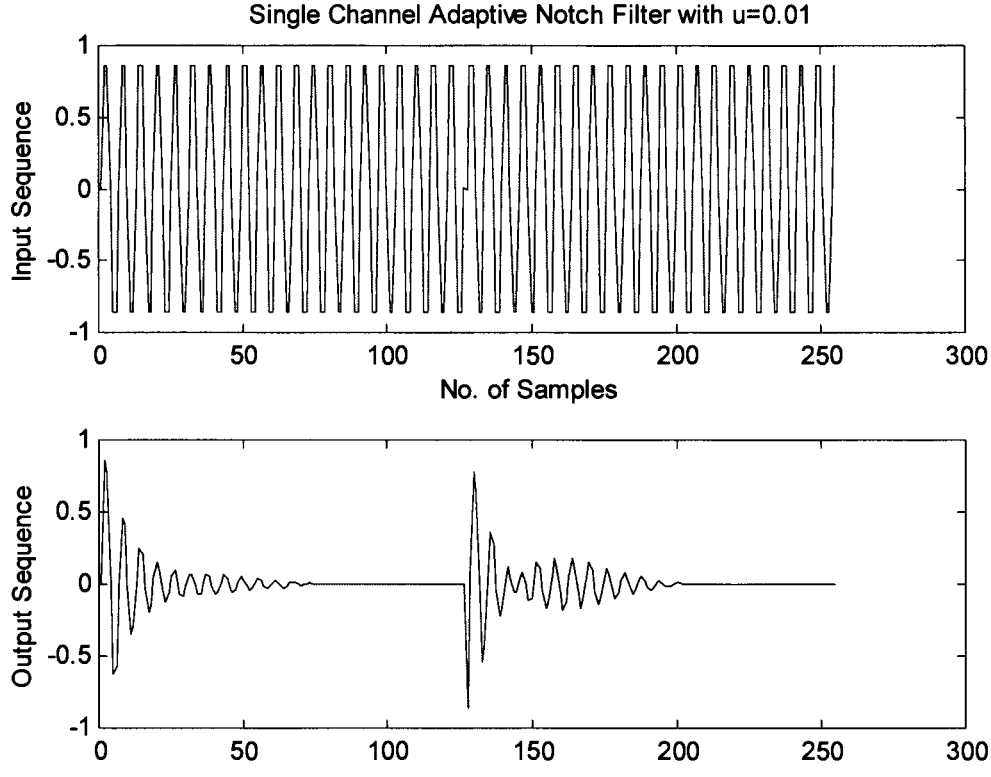

Figure 4.14 Output Response of Adaptive Notch Filter with u=0.01

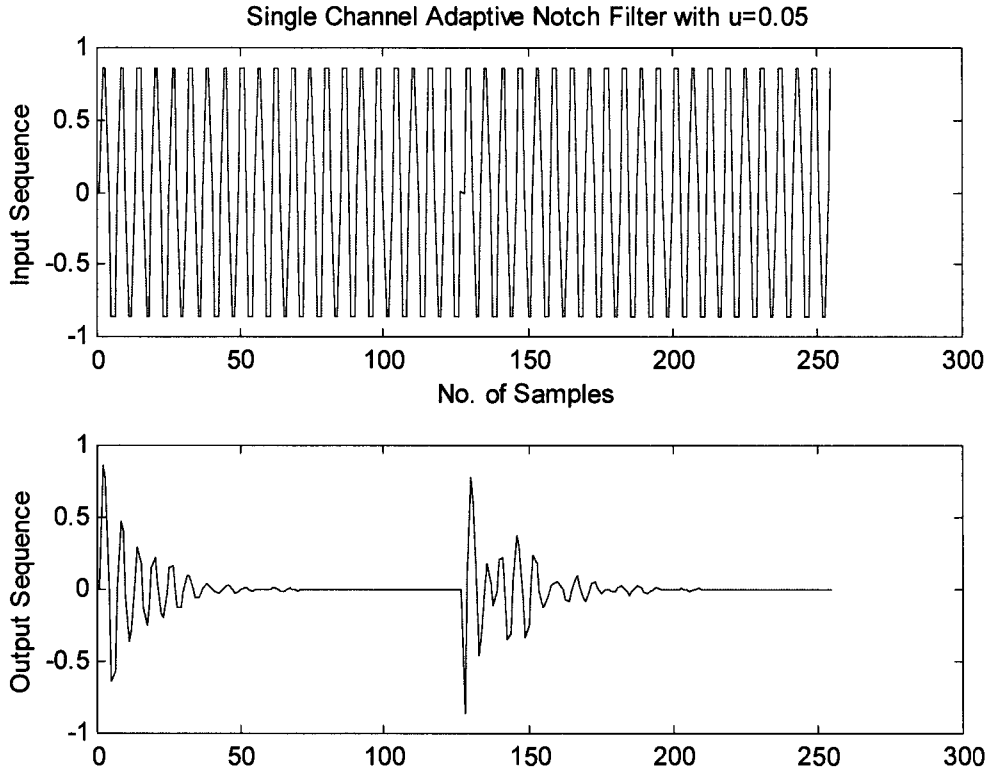

Figure 4.15 Output Response of Adaptive Notch Filter with u=0.05

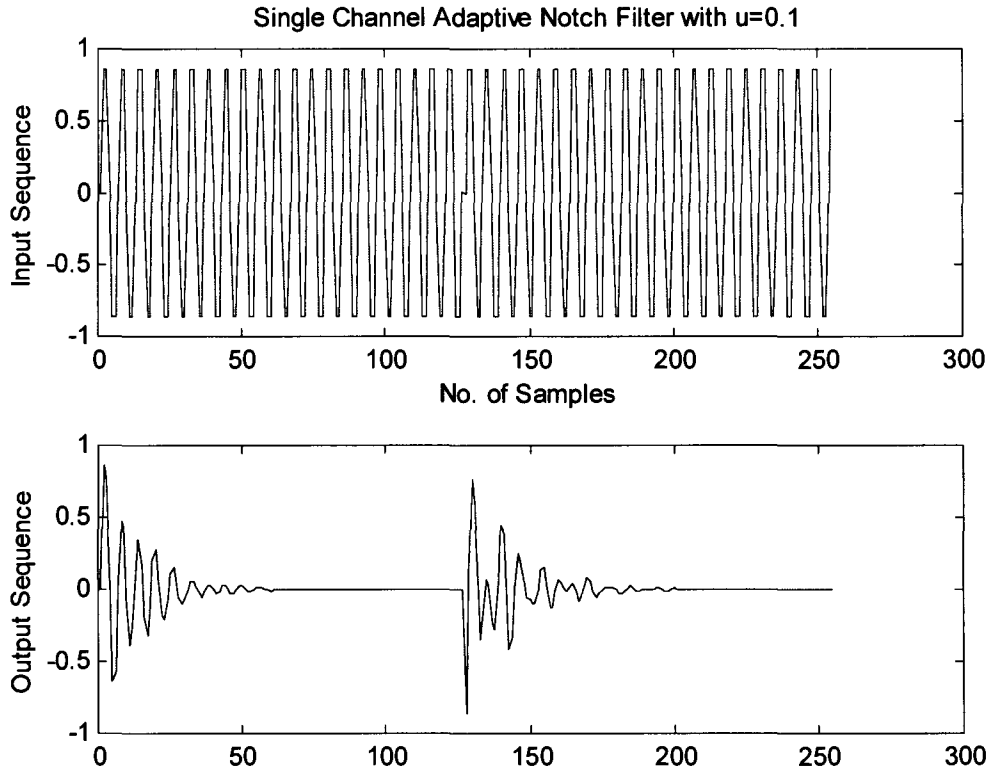

Figure 4.16 Output Response of Adaptive Notch Filter with u=0.1

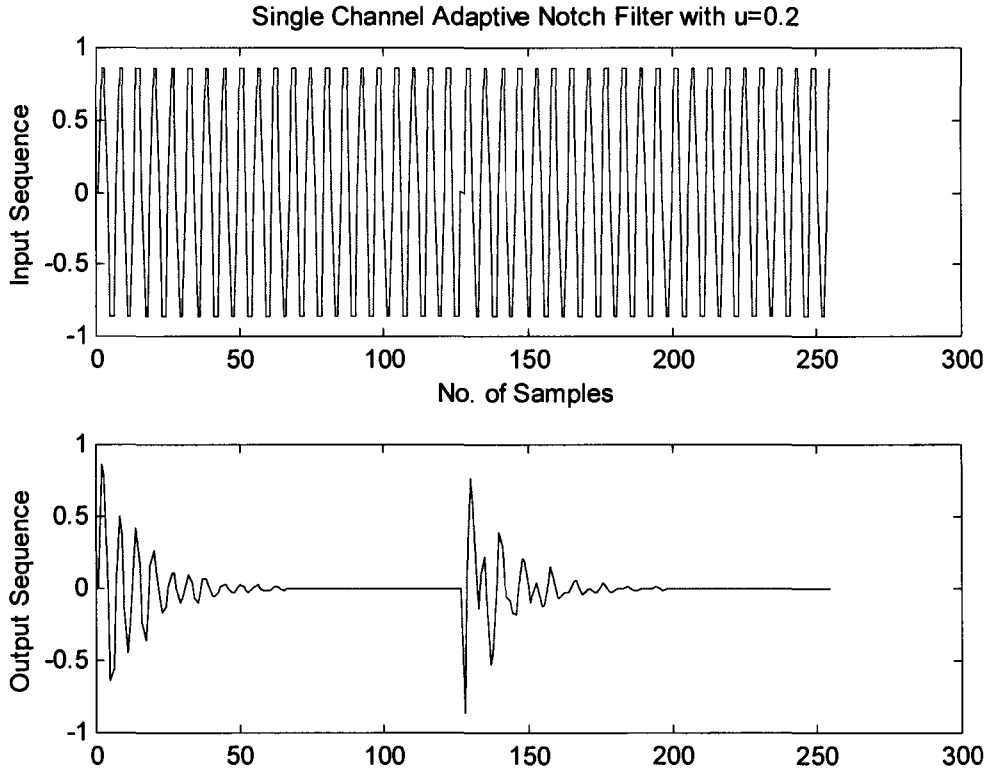

Figure 4.17 Output Response of Adaptive Notch Filter with u=0.2

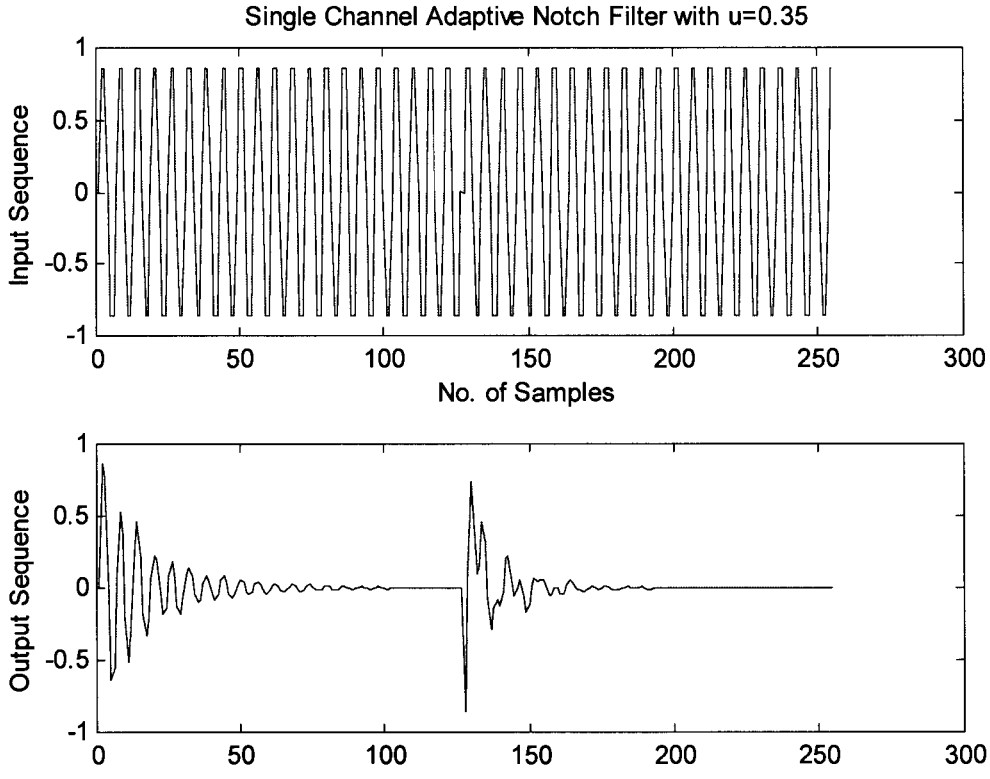

Figure 4.18 Output Response of Adaptive Notch Filter with u=0.35

4.3 CASCADE SECTION ADAPTIVE IIR NOTCH FILTER

The following figures showed the simulation results of the cascade filter with single frequency, frequency drift and multiple frequency input sequence. From the simulation results, it is shown that the filter able to track on the frequency drift and attenuate the frequency component in the region of notch frequency.

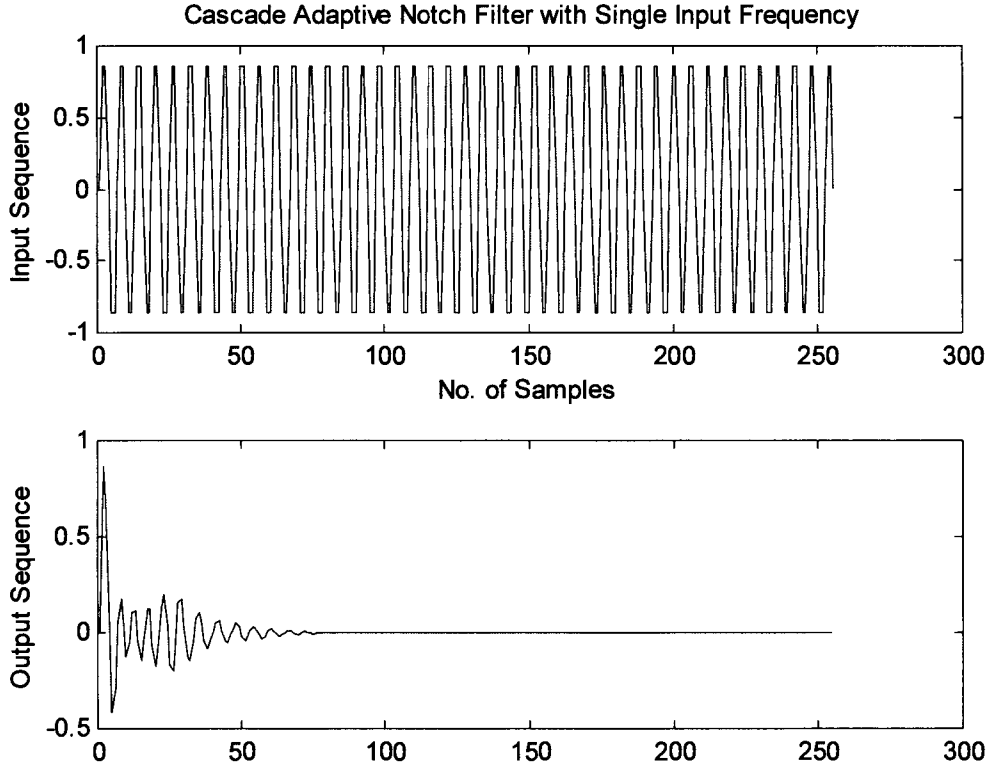

Figure 4.19 Cascade Notch Filter with Single Input Frequency

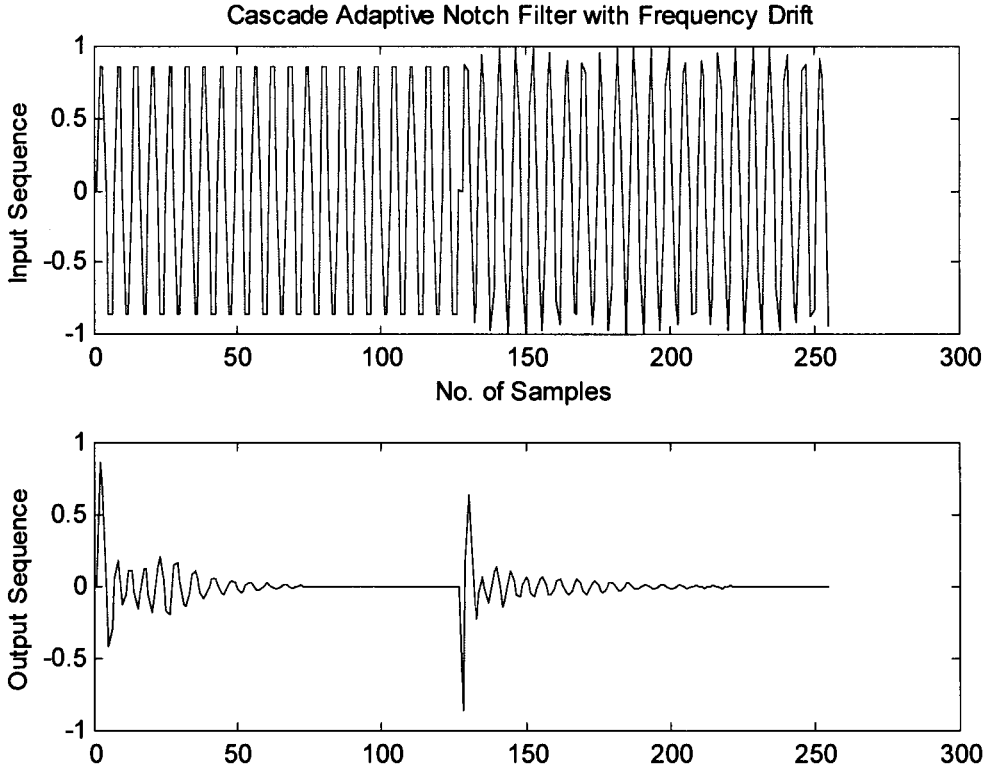

Figure 4.20 Cascade Notch Filter with Frequency Drift

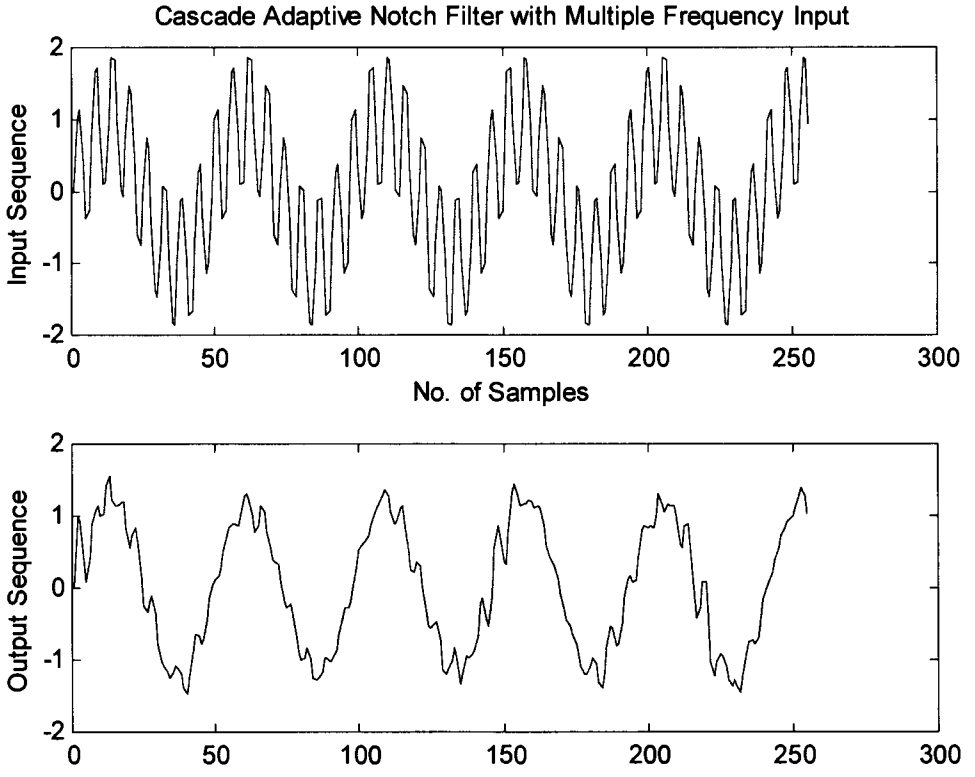

Figure 4.21 Cascade Notch Filter with Multiple Input Frequency

4.4 MULTIPLE ADAPTIVE IIR NOTCH FILTER

The following figures showed the simulation results of the multiple notch adaptive filter with multiple frequencies and frequency drift input sequence. From the simulation results, it is shown that the filter able to track on the frequency drift and attenuate the frequency component in the region of notch frequency.

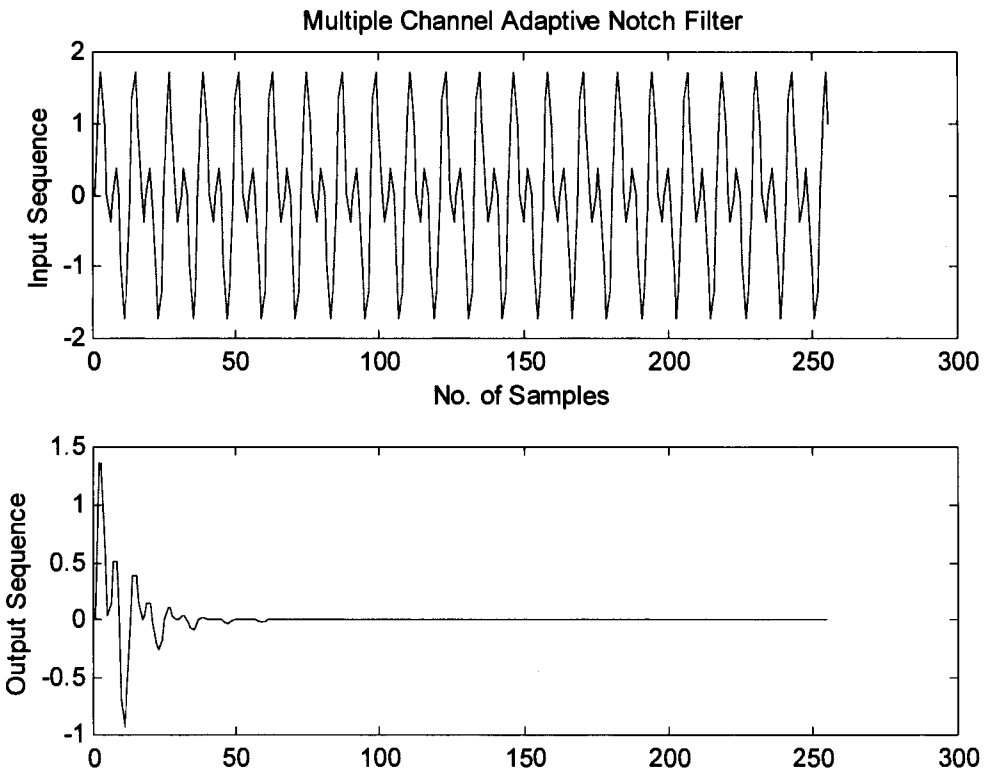

Figure 4.22 Multiple Notch Adaptive Filter

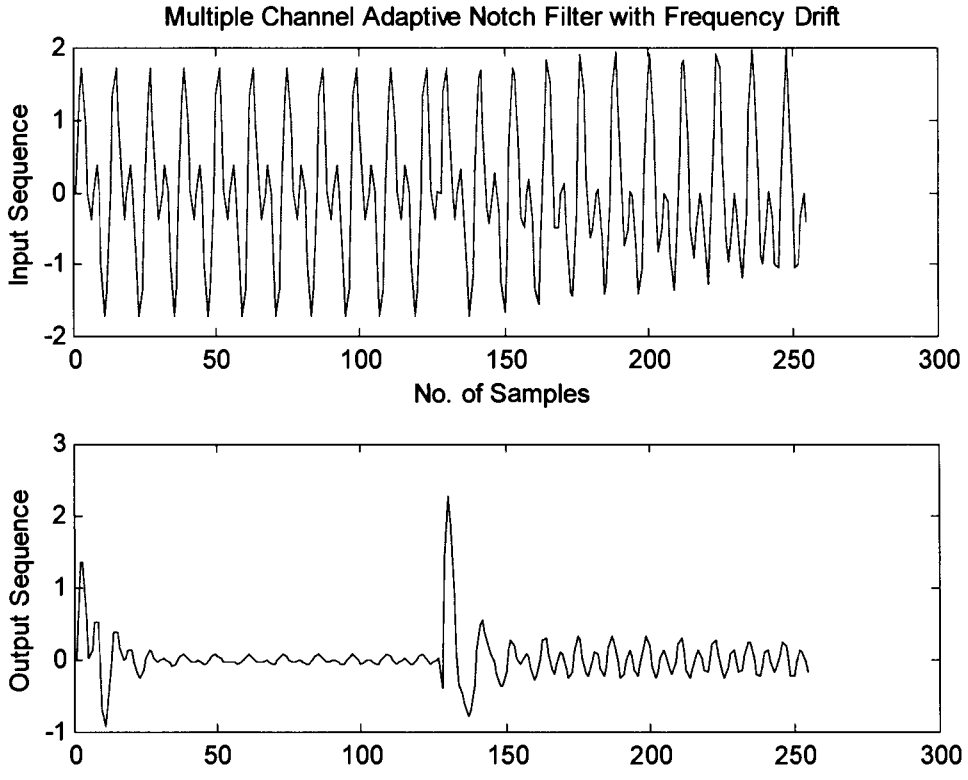

Figure 4.23 Multiple Notch Adaptive Filter with frequency drift

5 DEVELOPMENT TOOL

TEXUS INSTRUMENTS products, the TMS320C54X Evaluation Module (EVM) and the Code Composer Studio™ are used to develop the notch filters. TMS320C5X DSK was initially chosen for hardware implementation. In order to reduce the development cycle, software simulation is essential and it is accomplished by using Code Composer Studio. As the Code Composer Studio dose not support the TMS320C5X DSP chip for simulation, the hardware have been replaced by TMS320C54X based Evaluation Module (EVM).

5.1 TMS320C54X EVALUATION MODULE

5.1.1 OVERVIEW

The TMS320C54x evaluation module (EVM) is a PC/AT plug-in card that allows user to evaluate the performance of the DSP algorithm using the 'C54x digital signal processor (DSP). The software can be created and run on board or expand the system in a variety of ways.

The 'C54x EVM carries a 'C541 DSP on board to allow full-speed verification of 'C54x code. The 'C541 has 5K bytes of on-chip program/data RAM, 28K bytes of on-chip ROM, two serial ports, a timer, access to 64K bytes each of external program and data RAM, and an external analog interface for evaluation of the 'C54x family of devices for a given application.

5.1.2 KEY FEATURES

The 'C54x EVM has the following features:

- 'C541 operating at 40 MIPS with 128K words of zero wait-state memory
- Voice-quality analog interface to line I/O or speaker/microphone (user selectable) via standard RCA connectors
- External serial port
- Parallel I/O expansion bus
- Two 16-bit bidirectional host/target communication channels; one channel contains 64 words of buffering
- Embedded emulation support based on the IEEE 1149.1 standard
- A single 16-bit ISA half card, mappable to one of four I/O locations

5.1.3 FUNCTIONAL BLOCKS DESCRIPTION

Figure 5.1 shows the basic configuration and interconnects of the 'C54x EVM, including the host interface, target memory, analog interface, and emulation interface.

The 'C54x EVM supports a 16-bit PC/AT-host interface. The PC/AT interface provides the buffering, host I/O decode, and access control. The PC/AT host communicates to the 'C54x EVM via 38 16-bit I/O locations. Each I/O location is defined by the I/O page 0 address described by an offset. The first 32 I/O locations support emulation. The remaining six locations support host/target communications and control.

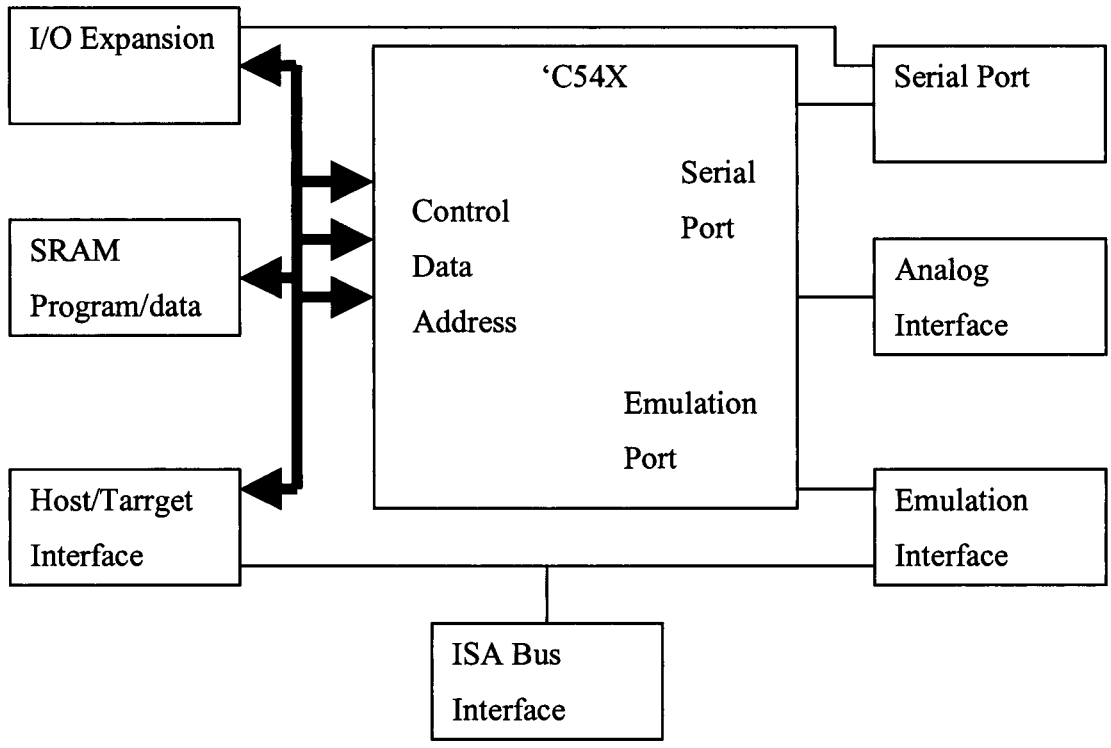

Figure 5.1 Configuration and Interconnection of TMS320C54X EVM

The 'C54x interfaces to 64K words of zero wait-state program memory and 64K words of zero wait-state data memory, in the total of 128K. Host target interface communicates through two independent channels, Channel A and Channel B. An external I/O interface supports 16 parallel I/O ports, a serial port, and other I/O features. A single single channel of voice-band analog input and out-put was provided by the EVM with programmable filtering, scaling, and sampling based on the TLC320AC01 analog interface circuit. Two RCA connectors provide analog input and output.

5.2 TMS32C5000 PC CODE COMPOSER STUDIO

5.2.1 INTRODUCTION

Code Composer Studio is a fully integrated suite of easy-to-use DSP software development tools, incorporating TI's efficient 'C5000 C compiler with the Code Composer Integrated Development Environment (IDE), DSP/BIOS™ and Real-Time Data Exchange(RTDX™) technologies. Code Composer Studio's real-time analysis and data visualisation capabilities, open architecture and advanced code generation tools greatly reduce the complexity of DSP development.

5.2.2 FEATURES AND BENEFITS

Code Composer Studio has two kind of tool features, compiler tools and debug tools.

Compiler tools includes C Compiler, Linker and IDE Simulator and debug tools includes Code Composer IDE C54x/C55x Emulation Debug Software, RTDX and DSP BIOS II.

In this project, the Code Composer IDE Simulator is used during our software development. The Code Composer Simulator could be used to simulate the actual execution of DSP code without the presence of a DSP chip, in parallel with hardware development. The advantages are the developers can also use the simulator for concurrent and field development and the simulator reduces costs by providing a team of developers access to the development environment through a software interface rather than through hardware.

5.2.3 PROJECT DEVELOPMENT CYCLE

The project development consists of initial set up and steps involved in the development cylcle.

5.2.3.1 Initial Set up

After installing the TMS320C5000 Code Composer Studio, an initial import configuration will prompt to add available system to the code composer setup. The C54x simulator will be chosen and added to the system.

The Code Composer Setup allow you to chose the available board or simulator types to add onto your system. In this case, the C54xx simulator type was chosen for our software simulation. In the later stage, the C54xx parallel port will be chosen to communicate with C54xx EMV board.

5.2.3.2 Development cycle

The developments cycles include the following steps :

Step 1. Project Management system - The C54x simulator provides the Visual Project Management system which is a fast way of visualising, accessing, and manipulating all project files from the same window. In the project view, files are organised into functional categories such as source files, include files, libraries, configuration and GEL script files. All compile options are saved with the project for easy retrieval.

Step 2. Build and Compile - Once a project file is created, used the compiler and builder to compile and build the programme. If the programme is successfully build, a .out file will be created which can be used to run the programme.

Step 3. Edit within the IDE - Code Composer Studio's™ integrated development environment (IDE) is similar to MS-Visual C++. The source code editor is tuned specifically for writing C and DSP assembly code to let you work more efficiently. The editor is integrated with all other tools in Code Composer Studio™ like the background build facility, the debugger, etc. This allows user to easily edit the code and see source and disassembly while coding and editing.

Step 4. Debug within IDE - Code Composer Studio's integrated debugger has DSP-specific capabilities and advanced breakpoints to simplify development. Conditional or hardware breakpoints are based on full C-expressions, local variables or CPU register symbols. A Probe Point™, a unique feature in Code Composer Studio™, is a sophisticated form of a breakpoint. It allows user to define a point in the algorithm where can perform an oscilloscope-type function. Unlike a breakpoint, program execution resumes after hitting a Probe Point™ and performs the connected activity (e.g. inject or extract signal data, observe signals, execute GEL script). Probe Points™ may be easily set, removed, or moved to other areas in the algorithm simply by clicking on the icon in the toolbar.

Step 5. I/O Data signal – Using Probe Points Execution reaches 1st Probe Point™: Inject known signal values from a file into DSP target to simulate conditions. Execution reaches 2nd Probe Point™: Execute GEL function (e.g. diagnostic test cases) and send a set of memory locations to disk. Execution reaches Third Probe Point™: Take snap shot of memory and display graphically.

Step 6. Analyze – Profile interactive is a quick measurement of code performance and how well the DSP target's resources are being used during your debugging and development session. Profile Points are similar to breakpoints but instead of halting the target processor, they accumulate hits and collect statistics on the number of instruction cycles or other events that have elapsed since the previous profile point was hit. This will allow high usage areas in optimisation to produce finely tuned code. Tightly integrated data visualisation also allows user to view data and signals in their native format for easy interpretation and analysis with many display types. As the program animates, a snapshot of the signal is taken at each probe point connected to a data visualisation window allowing user to see the signal as it progresses through an algorithm during debugging.

Step 7. Real time analysis using DSP/BIOS - Probe, trace, and monitor a DSP application while it runs! These utilities are based on a real-time link and awareness between the Code Composer Studio^{TM} host environment and the target. Even after the program has been halted, information already captured through the real-time analysis tools can provide invaluable insight into the sequence of events that led up to the current point of execution. Real-time analysis tools are used later in the development cycle when transitioning from the debug phase to the runtime phase of development.

6 IMPLEMENTATION

6.1 SOFTWARE IMPLEMENTATION

The simulation was done using Matlab for windows Version 5.3.1 for testing the feasibility of the adaptive algorithm and it had been discussed in Chapter 4. This will enable verification before actual implementation on the TMS320C54xx Digital Signal Processing Chip.

In this chapter, the software implementation on TMS320C54xx Digital Signal Processing will be discussed. The sequence, description of the programs and the graphical results of the simulation will also be discussed in the following section. The programme listings are attached in Appendix B.

6.1.1 OVERVIEW

Code Composer Studio is used as part of the software development system of TMS320C54xx DSP. It basically consists of six modules. Figure 6.1 shows the software development flow diagram. The **C compiler** accepts C source code and produces assembly language source code. The **assembler** translates assembly language source files into machine language object files. The machine language is based on common object file format (COFF). The **linker** combines object files into a single executable object module. As it creates the executable module, it performs relocation and resolves external references. The linker accepts relocatable COFF object files and

object libraries as input. While in the simulator, the simulations of the programmes were successfully generated. However, the hardware interface is yet to be performed.

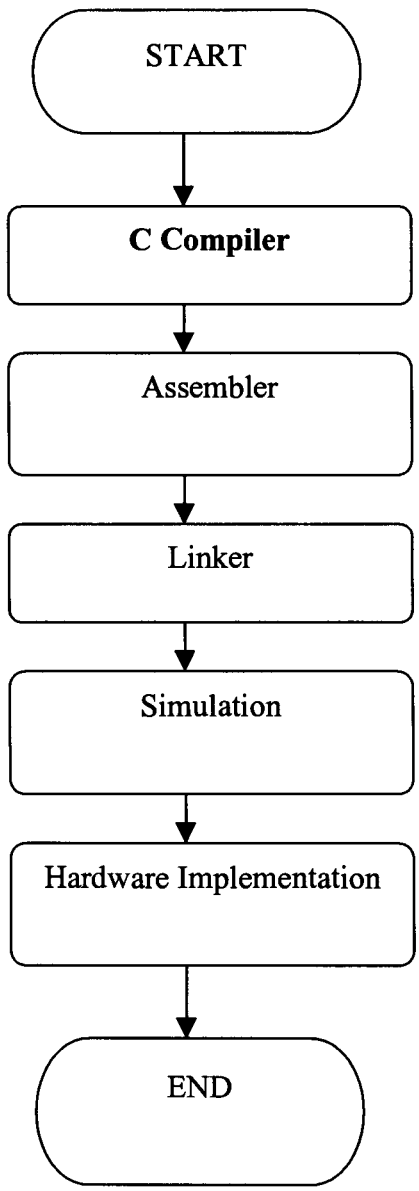

Figure 6.1 Software Development Flow

6.1.2 DATA REPRESENTATION

The data representation is in the Q15 format, dynamic range from – 1 to +0.999 corresponds to hexadecimal range of 8000 to 7FFF. The hardware limits the input range of the noise signals and the output range of the cancelling signals to a certain range, therefore, a format conversion is needed to comply with the Q15 format. This involves shifting the data by 8 bits to the left and, then exclusive OR it with hexadecimal number 8000.

6.1.3 PROGRAMME STRUCTURE

The programme was developed using the same sequence of flow as the Matlab simulation. The sampling frequency is set to 48KHz (using audio bandwidth = 22.1Khz). The input signal frequency is fixed at 8KHz and the number of sampling is limited to 200.

There are 4 different of programs being generated according to the different signal input, step size and coefficient. The first programme is a Basic Notch Filter where the coefficient "a" is fixed at a frequency. The second programme is an Adaptive Notch Filter where the coefficient "a" is adapted according to drift frequency. The programme also depends on the step size to generate a fast or slow adaptation. The third programme is a Cascaded Notch Filter where it gives a better performance than the basic notch filter. The fourth programme is a Multiple Notch filter where the capability is to notch two frequency with the input of two signal frequency. The programme flow chart of the 4 programs is shown at Figure 6.2

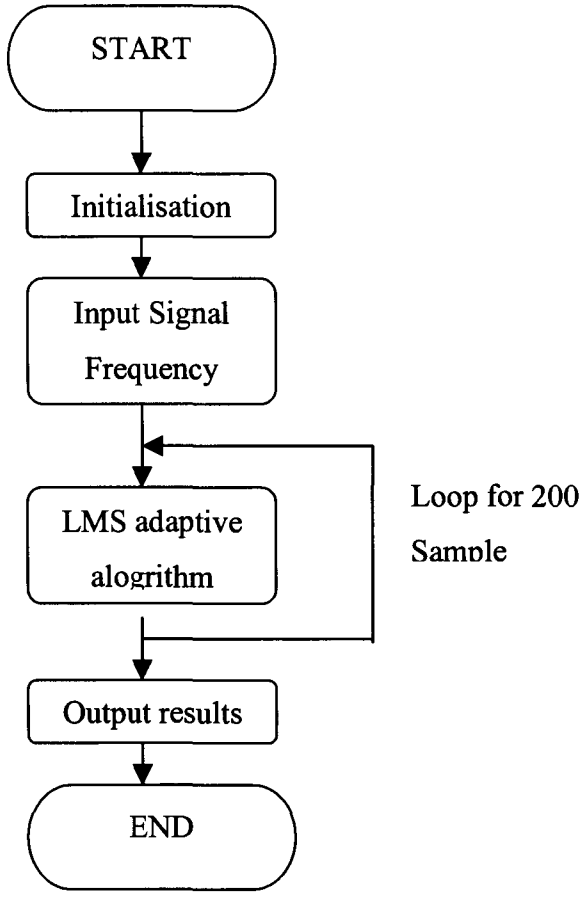

Figure 6.2 Programme Flow

The programme started with the initialisation of relevant parameters and variables. The input signal was added to the program through the file I/O function. Both of the input data file and output results are assigned to a probe point in order to monitor the simulation results in a graphical diagram. The programme will enter a loop which perform 200 steps of adaptive algorithm equation. The algorithms vary with types of filters. An output array with the length of 200 sample will be generated and displayed through the probe point setting.

6.1.4 RESULTS

The development results presented here are generated using the simulator program, C54x Code Composer Studio (Simulator/CPU).

6.1.4.1 Basic IIR Notch Filter

The basic notch filter is generated using equation (3.2). The input signal frequency is at 8KHz and the parameter, r varies between 0.9 or 0.75. The coefficient, a is a constant value and the results are shown in Figure 6.3 and Figure 6.4 .

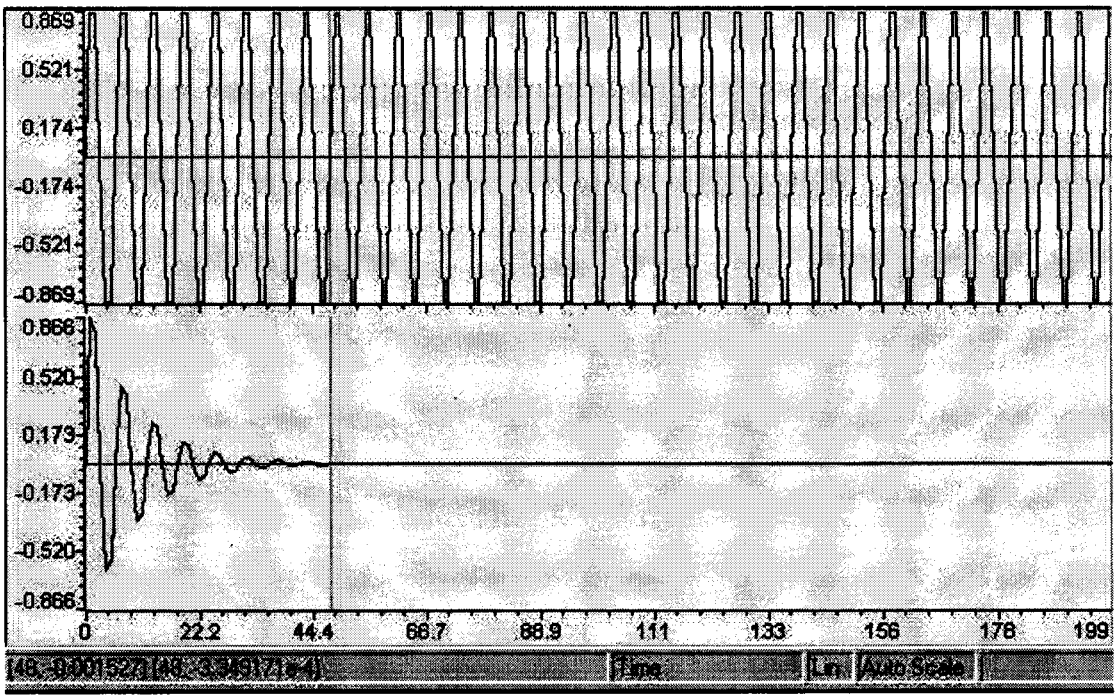

Figure 6.3 Output Response with r = 0.9

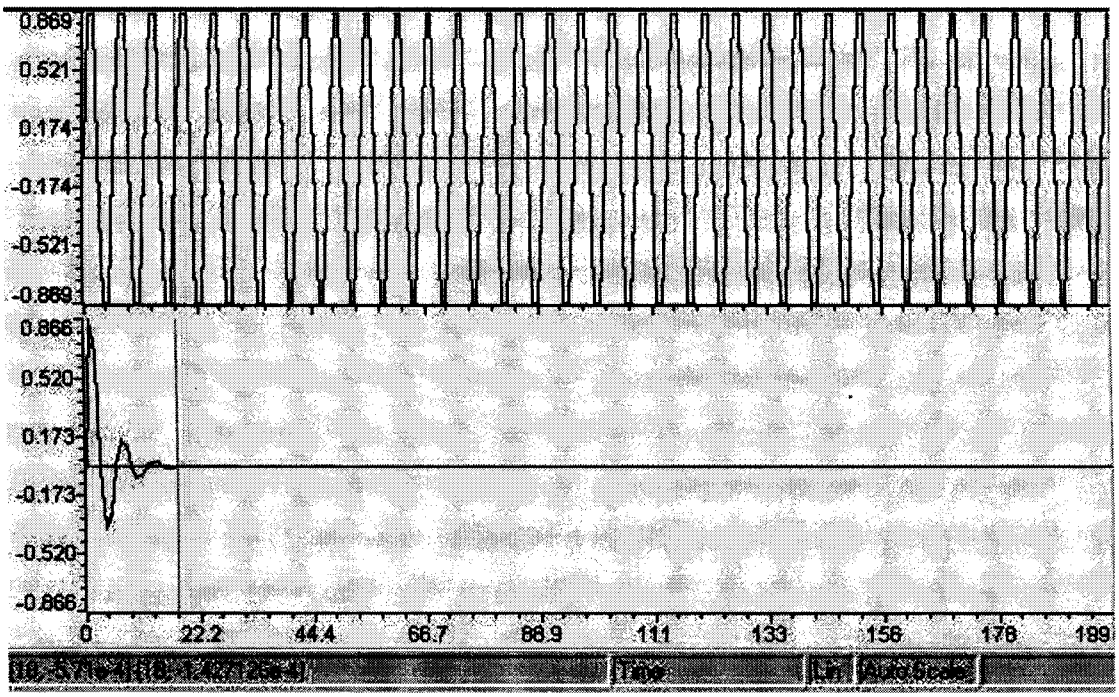

Figure 6.4 Output Response with r = 0.75

The following figures (Figure 6.5 and Figure 6.6) showed the simulation results with multiple sinusoidal input noise signal frequency of 4Khz and 8Khz with the parameter maintained at 0.9 and 0.75. It is shown that the notch filter has cancelled one of the noise input signal at 8KHz and generated only 4KHz output.

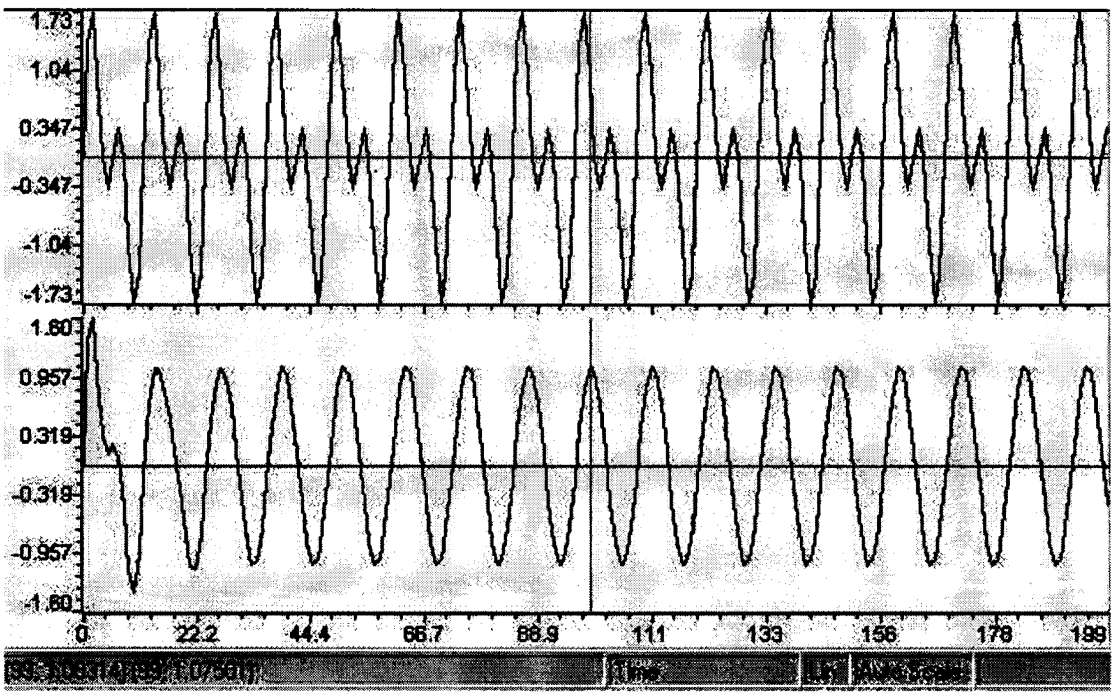

Figure 6.5 Output Response with multiple frequency input, r = 0.9

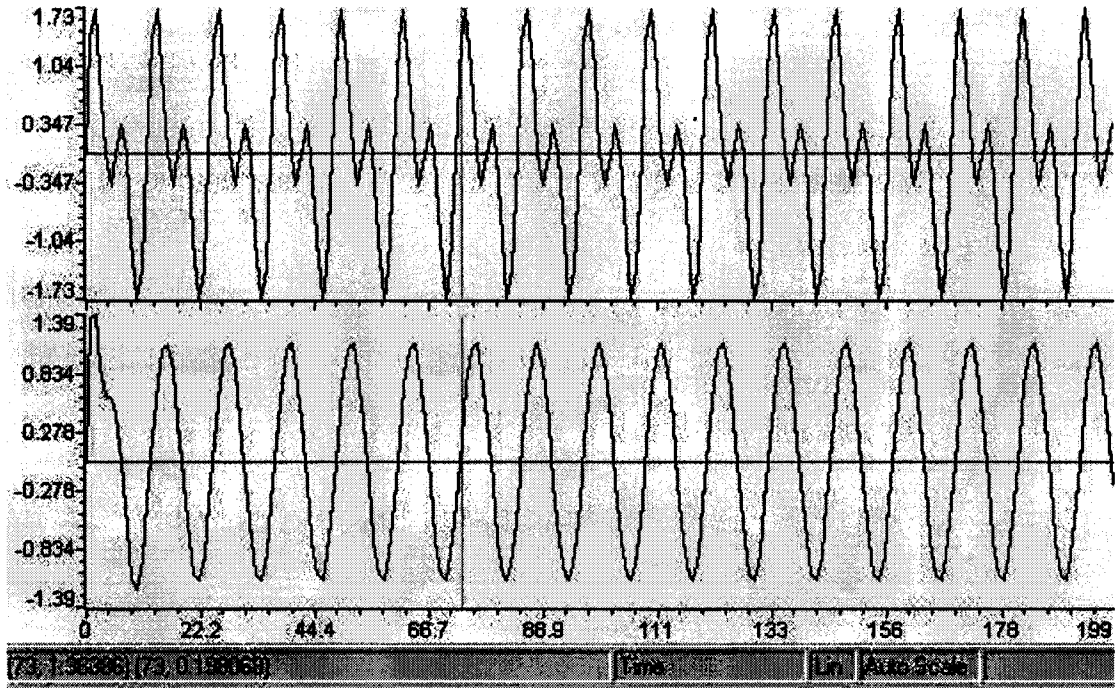

Figure 6.6 Output Response with multiple frequency input, r = 0.75

6.1.4.2 Adaptive IIR Notch Filter

The adaptive notch filter is implemented using equation (3.2) and equation (3.3). The following figures showed simulation results with different step size. The input signal is 8KhZ and drift to 8.1KHz after 100 samples. The parameter r is set at 0.9.
From the figures, it shows that, the smaller the step size the more accurate the notch frequency but at the expense of a large number of samples. That is, it takes a longer time to adapt.

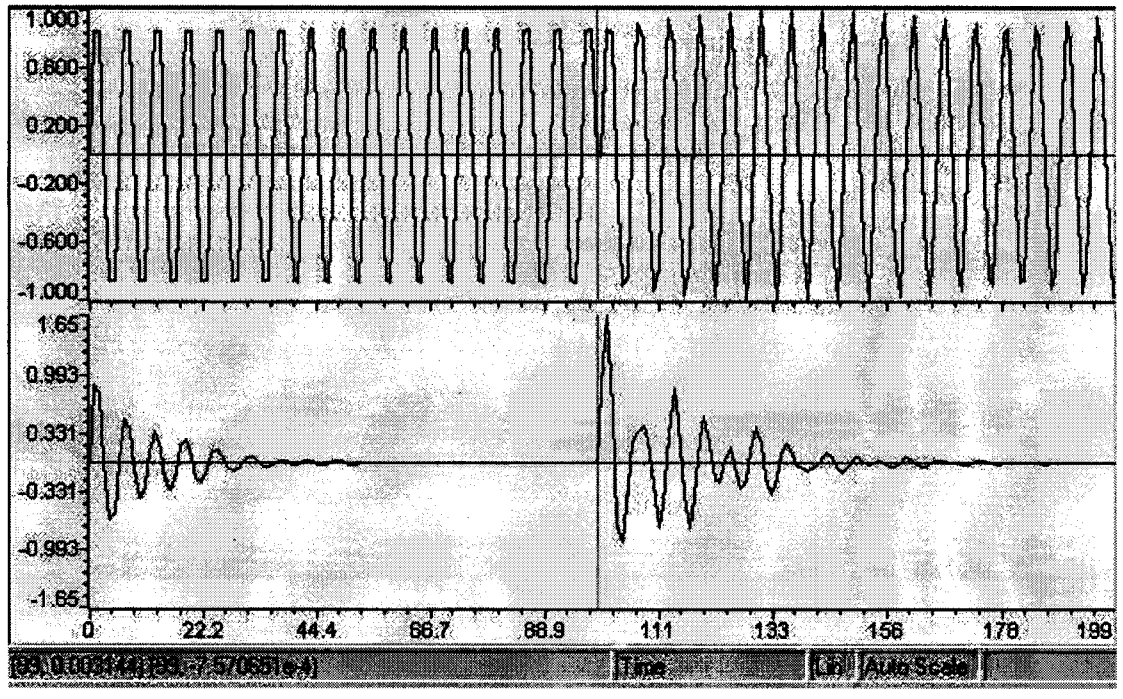

Figure 6.7Output Response with single input frequency and u = 0.1

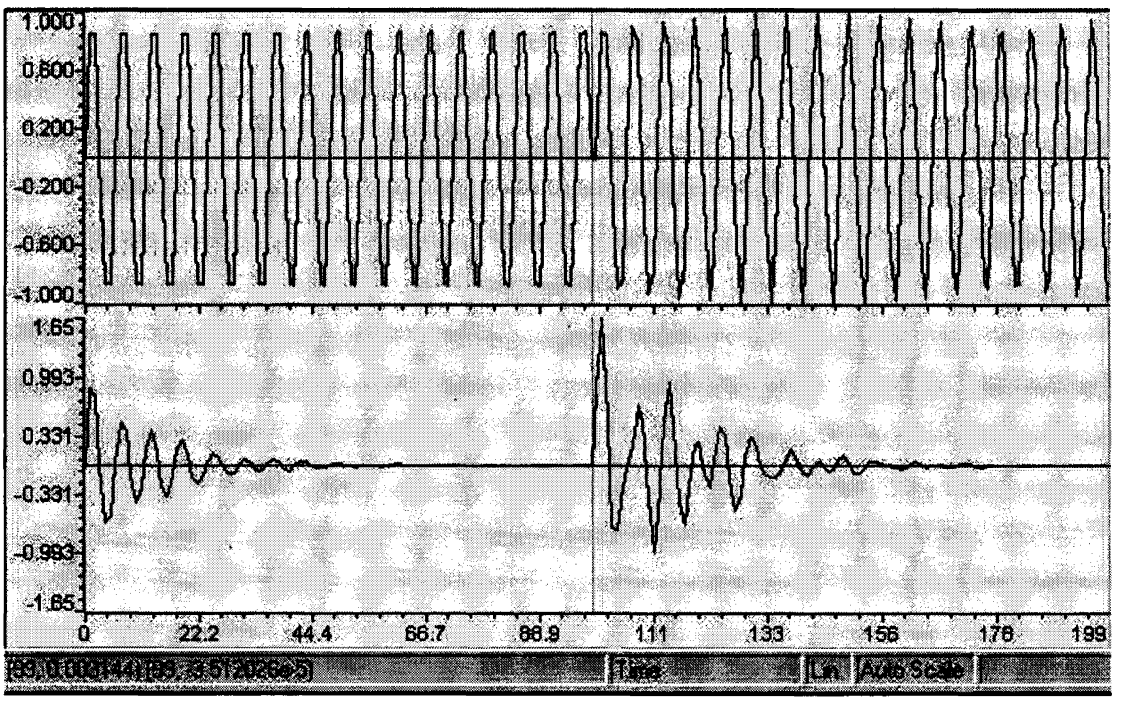

Figure 6.8Output Response with single input frequency and u = 0.15

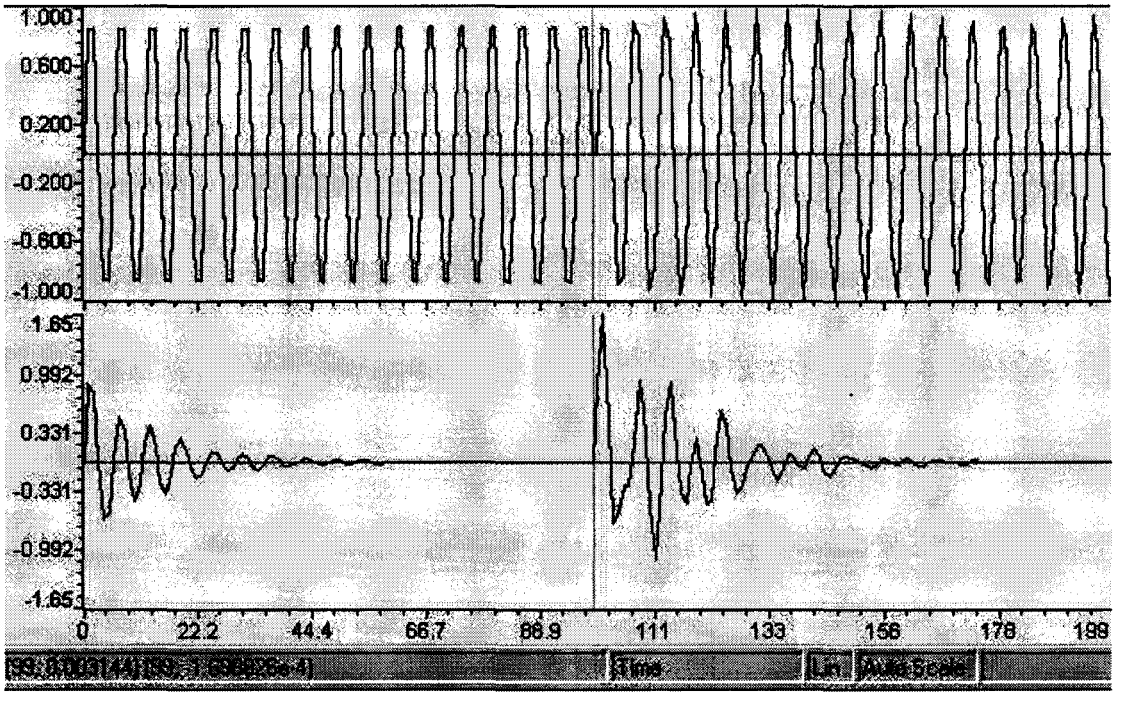

Figure 6.9Output Response with single input frequency and u = 0.2

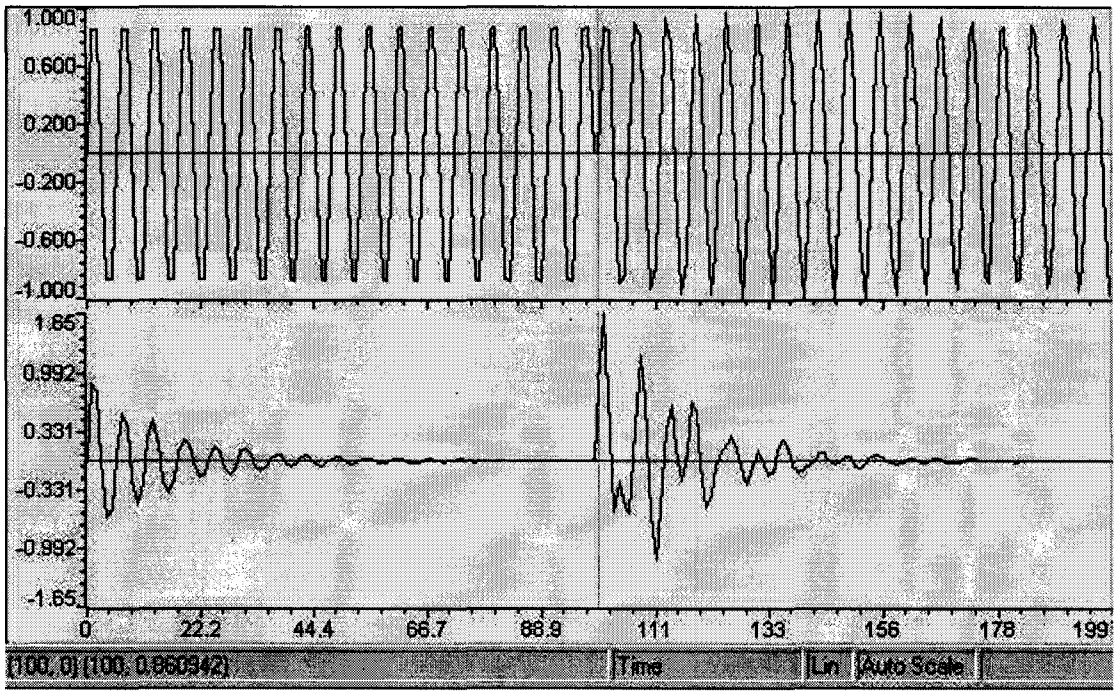

Figure 6.10 Output Response with single input frequency and u = 0.3

The following figures showed the results with multiple channel input (4KHz and 8KHz). It is observed that a very small step sizeis required (u=0.1) in order to notch off one of the noise signal frequency.

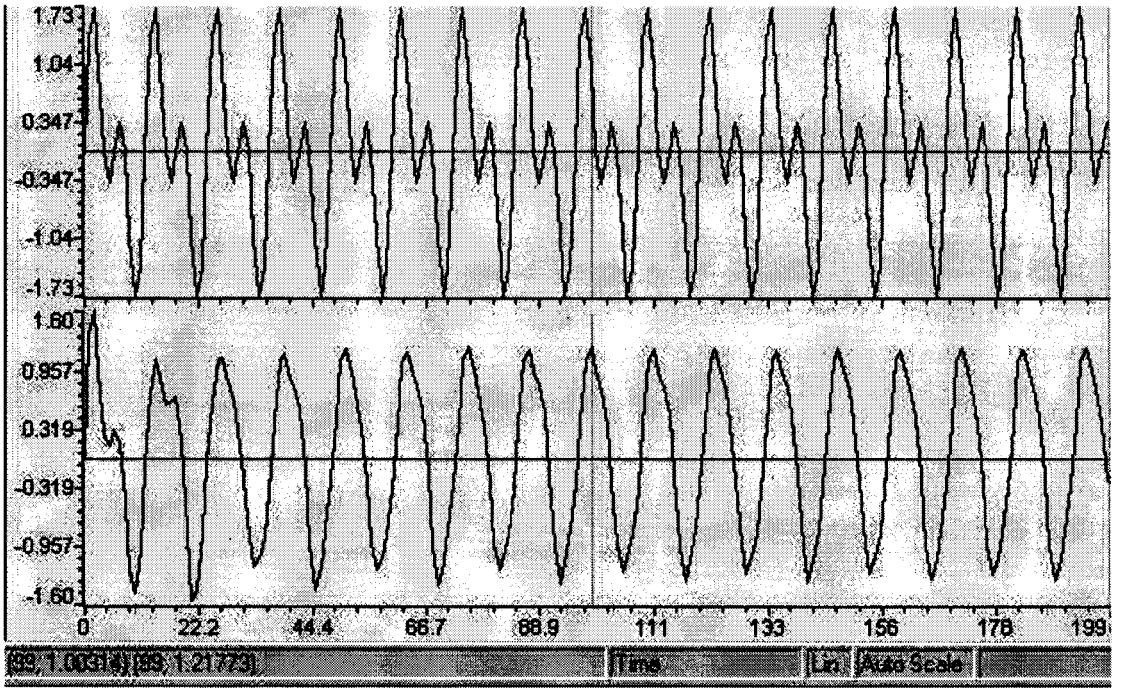

Figure 6.11Output Response with multiple frequencies input and u = 0.1

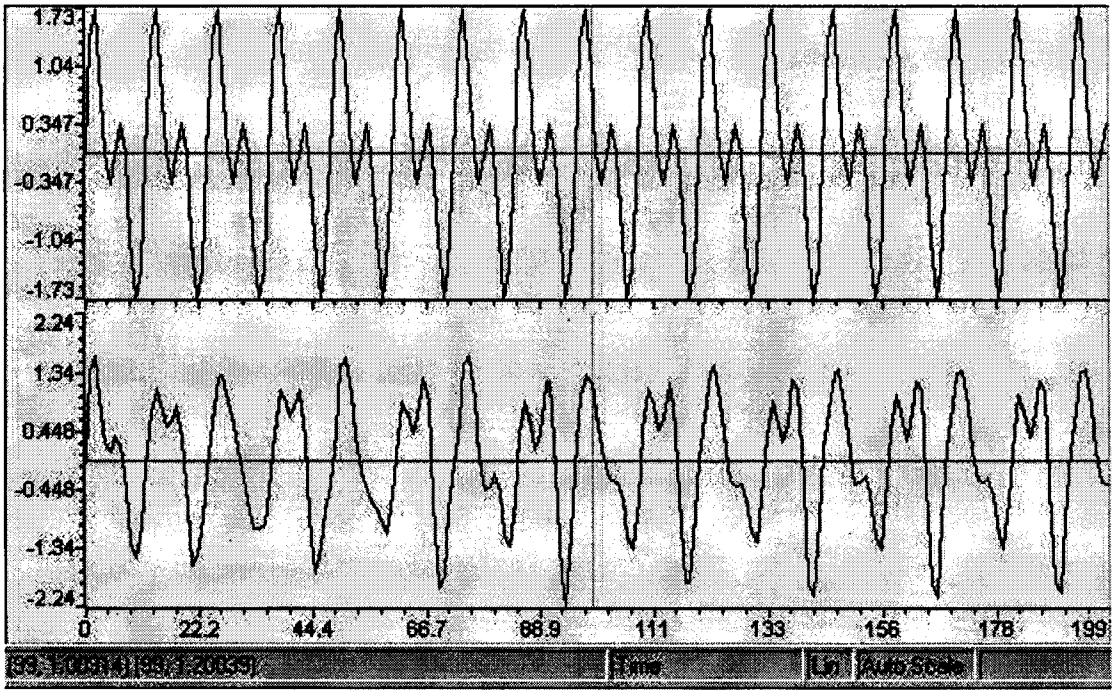

Figure 6.12 Output Response with multiple frequencies input and u = 0.15

6.1.4.3 Cascade Adaptive IIR Notch Filter

The cascade filter is implemented by having two section of the basic notch filter and adapting at the end of the second section. Figure 6.13 and Figure 6.14 showed simulation result with different input signals.

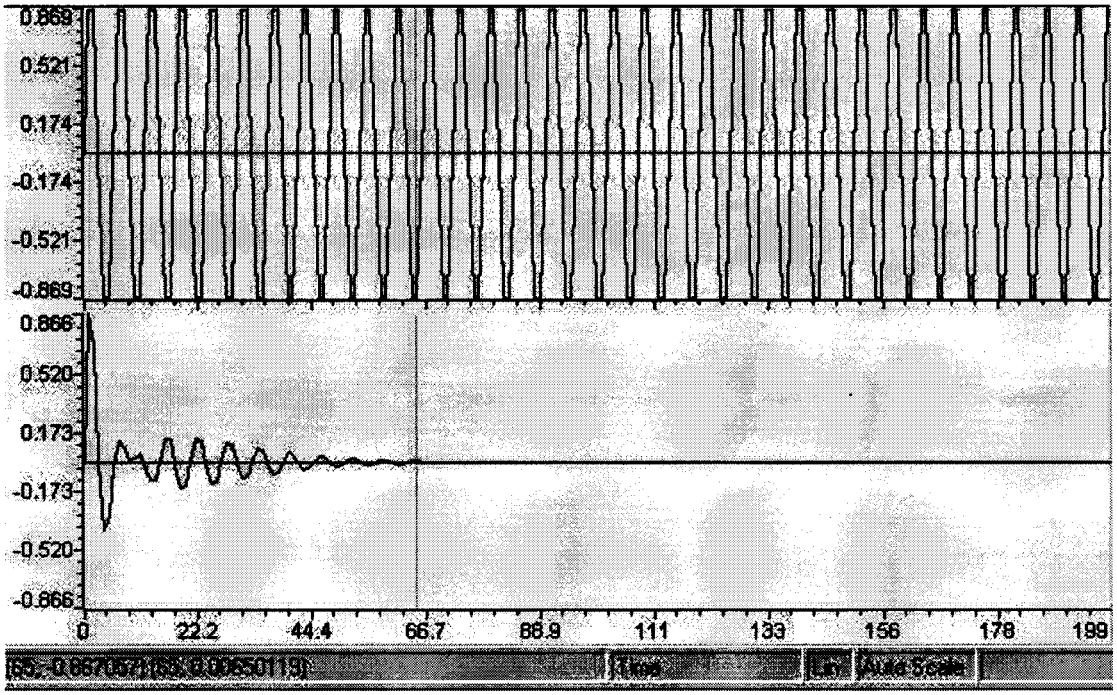

Figure 6.13 Cascade Section output with u = 0.25 single frequency input

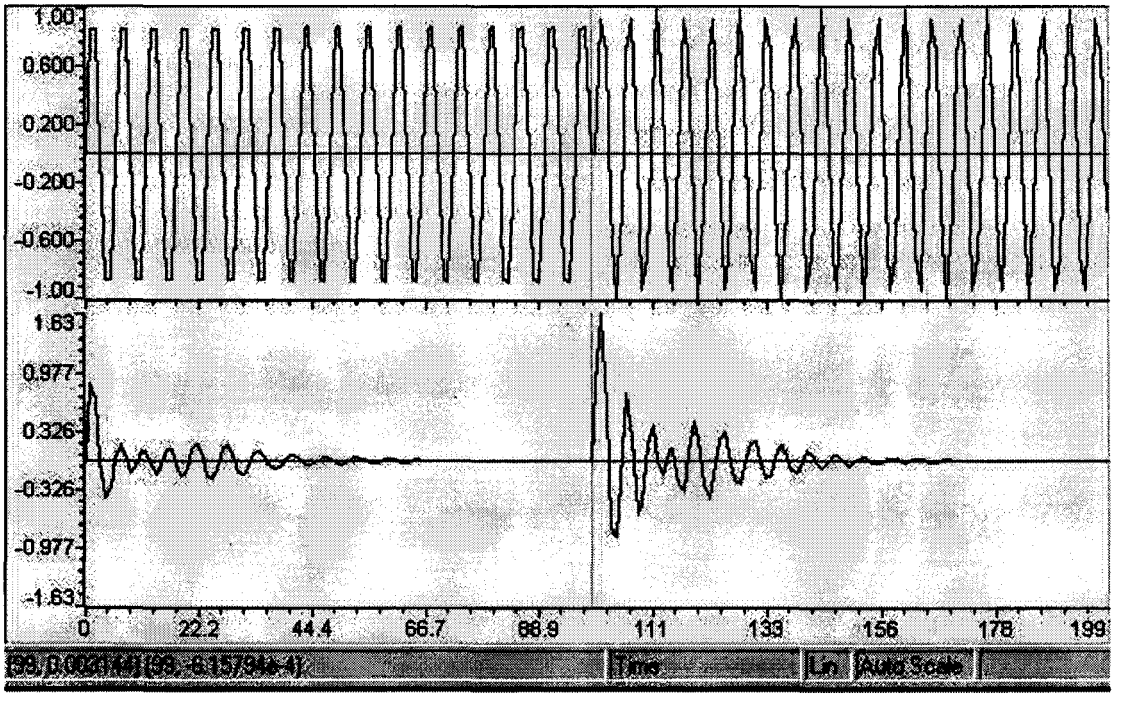

Figure 6.14 Cascade Section output with u = 0.25 multiple frequency input

This page has intentionally been left blank

6.1.4.4 Multiple Notch Filter

The cascaded notch filter is extended to cancel out the multiple sinusoidal noise interference. It is implemented by having two different notch frequencies. The simulated output result is shown in Figure 15 and Figure 6.16.

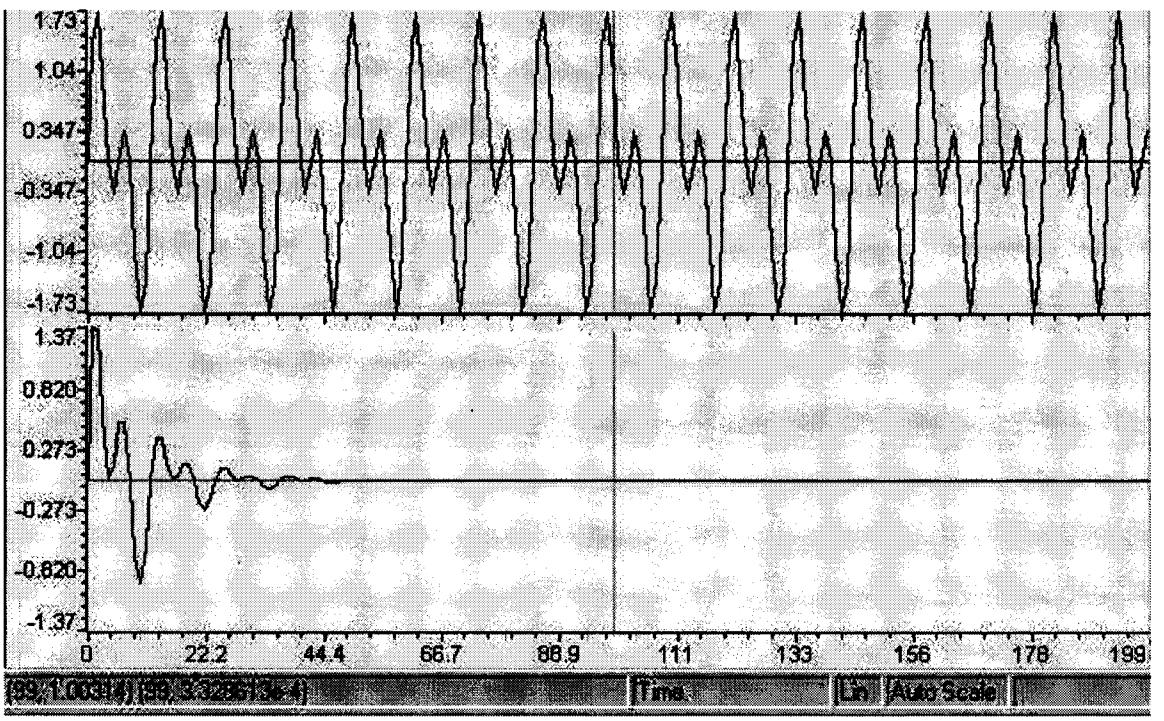

Figure 6.15Output Response of multiple notch filter .

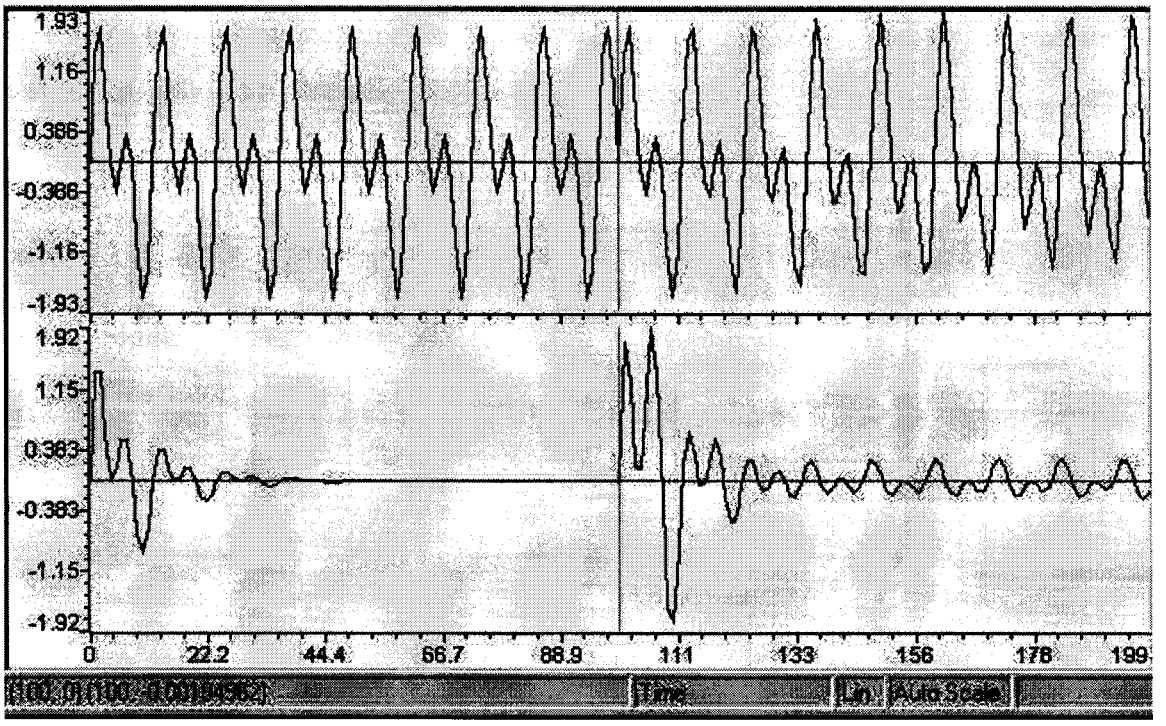

Figure 6.16 Output Response of multiple notch filter with frequency drift

6.2 HARDWARE IMPLEMENTATION

After the simulation process in both MATLAB and Code Composer Studio, the algorithm is ready to be implemented in a hardware system. The TMS320C54X EVM is used and the test set up is shown in the following figure.

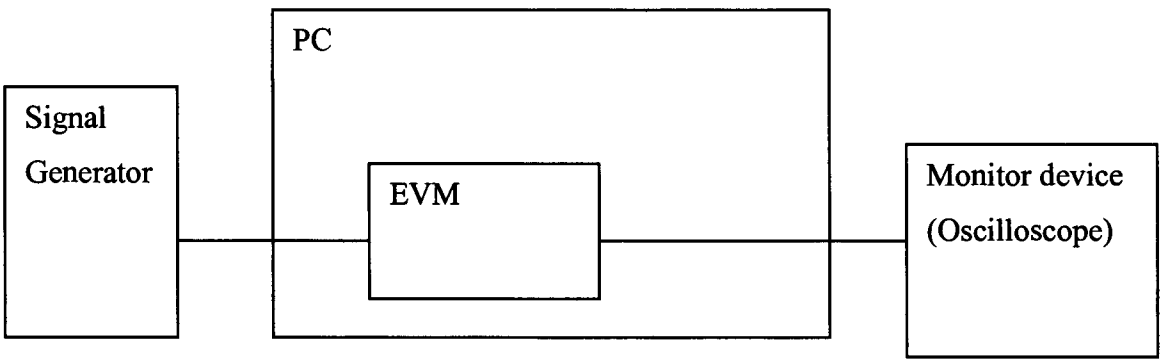

Figure 6.17 Hardware Implementation Test Set up

The signal generator is used to inject an analog input to the analog input connector of the EVM. A simple adder is used if multiple frequencies input is required. The analog output of the EVM is monitored by either an oscilloscope or other equivalent equipment which can perform the same task. Figure 6.18 showed the location of the EVM board in the PC.

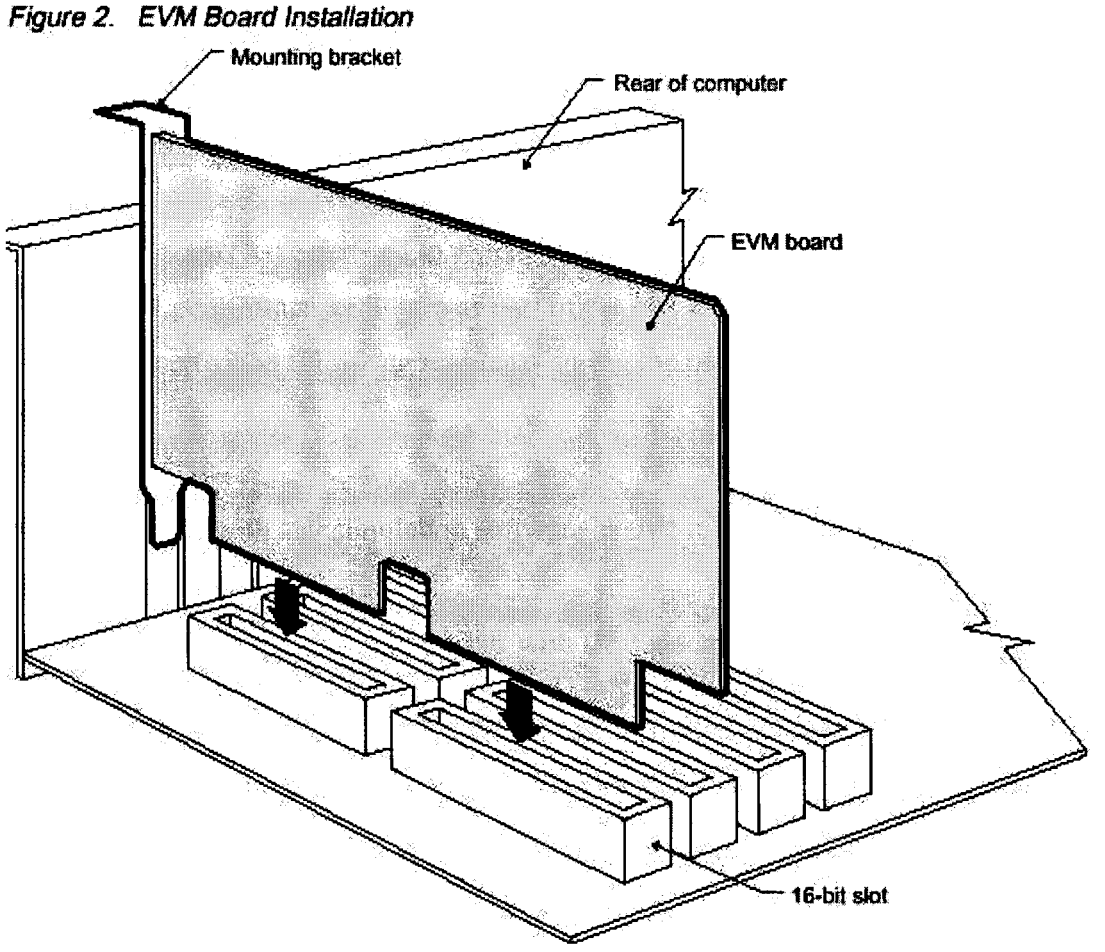

Figure 6.18 Location of EVM in the PC

7 PROBLEMS AND RECOMMENDATIONS

7.1 TECHNICAL PROBLEMS

(1) The project teams have decided to replace the TMS320C5X DSK by TMS320C54X EVM with Code Composer Studio as part of the software development system tool. The team encountered the following problems during the project phase with the initial DSK board :

- Not compatible with any simulation software available in the working company.

- As both team members are not familiar with DSP assembly programming, we encountered difficulties in learning the TMS320C5X assembly code in short period. We decided to switch to higher level programming language, "C" Program, which we are more comfortable. Unfortunately, the company did not purchase the development tool for TMS320C5X with "C" compiler features. After some discussion and advice from colleagues who is working in DSP field, the TMS320C5X was replaced with TMS320C54X EVM which the processor is supported by the Code Composer Studio. Code Composer Studio. Support accepts programs written in both assembly and C language and generate output file for the Evaluation Module.

(2) The adaptation of parameter, r was not successful and did not produce the desired response as the derived approach. It is suggested to look into alternative adaptive algorithm

(3) The hardware implementation was not able to perform during submission of the report. It may be due to Host/Target interface problems

7.2 GENERAL PROBLEMS

(1) Due to the short summer semester, both the project members felt the project 2 is running in a very tight schedule, only a total of 8 weeks was given We are not able to complete the project on schedule. It is suggested that to extend the future entire project which include both Project 1 and Project 2 three semester

(2) Students may not have enough exposure to latest technology development, the project proposed by student may not able to fulfil the requirement for the subject in all areas. It is suggested to have a list of projects recommended by lecturer and allow student to have an option for the student to select the project which they are interested to work on.

8 CONCLUSION

In this project, it is shown that the LMS algorithm is able to extended from the FIR structure to the more general IIR structure. The simplified LMS algorithm derived in Project Phase I was simulated with both MATLAB and Code Composer Studio. The basic MATLAB simulation provides a general performance analysis and Code Composer Studio showed results closed to physical hardware implementation. Different types of Notch filter which include Basic Notch filter, Single & Multiple Channel Adaptive Notch Filter and Cascade Section Notch Filter were simulated successfully. Unfortunately, the hardware implementation was not successful until the due date of the report submission due to some technical problems.

This project provides us an opportunity to explore and learn about the adaptive processing technique, which is a rapid growing field in DSP environment. The knowledge and experience which we had learnt in the project will definitely add value in our career advancement.

REFERENCES

[1] *Adaptive Signal Processing*, Bernard Widrow Samuel D. Stearns

[2] *Digital Signal Processing- A Practical Approach,,*

Emmanuel C. Ifeachor Barrie W. Jervis

[3] *Digital Filter Design, T.*W. Parks C.S. Burrus

[4] Digital Signal Processing Application

[5] *An introduction to adaptive filt*ers, C F N Cowan
DSP: Theory, Applications and Implementation, IEE Younger Members Tutorial Seminar on , 1996.

[6] TMS320C54X Evaluation Module Technical Reference, TEXUS INSTRUMENTS, 1995

[7] TMS320C54X Installation Guide, TEXUS INSTRUMENTS, 1995

[8] PC/TMS320C54x Evaluation Module Communication Interface, P. Geremia,, 1997

[10] TMS320C54x Code Composer Studio Tutorial , TEXUS INSTRUMENTS, 2000

APPENDIX A
MATLAB FILE

```
%Filename :Freq_RespR90.m
%Description : Plot frequency response with different values
of r

fs=48000; %sampling frequency in Hz
fc=2000; %angular frequency of the notch in Hz
w=2*pi*fc/fs; %
a=2*cos(w);
r=0.90;
den=[1,-a*r,r^2]; % Denominator of the response
nume=[1,-a,1];    % Numerator fo the response
[H,F]=freqz(nume,den,512,fs); % Compute Frequency response
HdB=20*log10(H);
figure(1);
plot(F,HdB);
xlabel ('Frequecny in Hz');ylabel('Gain in dB');
title ('r=0.90');
```

```
%Filename :Bnotch_R95.m
%Description : The file plot the output response of the basic
notch filter
%              with the notch filter transfer function (Poles
& zero within
%              the unit circle

fs=48000; %sampling frequency in Hz
fc=4000; %angular frequency of the notch in Hz
fi=4000; %angular input frequency in Hz
w=2*pi*fc/fs;
a=2*cos(w);
r=0.95; %|r| <1,poles within unit circle
den=[1,-a*r,r^2]; % Denominator of the response
nume=[1,-a,1];    % Numerator fo the response
nT=(0:1:255); % Range of sampling steps
xn=sin(2*pi*fi*nT/fs);%Input signal function
yn=filter(nume,den,xn); % Compute filter output response
subplot(2,1,1);plot (nT,xn);%Plot input signal
xlabel ('No. of steps ');ylabel('Input Sequence');
title ('Output Sequence with poles within unit circle');
subplot(2,1,2);plot(nT,yn); %plot output signal
;ylabel('Input Sequence');
```

```
%Filename :Bnotch_R105.m
%Description : The file plot the output response of the basic
notch filter
%              with the notch filter transfer function (Poles
& zero outside
%              the unit circle

fs=48000; %sampling frequency in Hz
fc=4000; %angular frequency of the notch in Hz
fi=4000; %angular input frequency in Hz
w=2*pi*fc/fs;
a=2*cos(w);
r=1.05; %|r| >1,poles outside unit circle
den=[1,-a*r,r^2]; % Denominator of the response
nume=[1,-a,1];    % Numerator fo the response
nT=(0:1:255); % Range of sampling steps
xn=sin(2*pi*fi*nT/fs);%Input signal function
yn=filter(nume,den,xn); % Compute filter output response
subplot(2,1,1);plot (nT,xn);%Plot input signal
xlabel ('No. of steps ');ylabel('Input Sequence');
title ('Output Sequence with poles outside unit circle');
subplot(2,1,2);plot(nT,yn); %plot output signal
;ylabel('Input Sequence');
```

```
%Filename :Single_ad.m
%Description : The file plot the output response of the
adaptive notch filter
%              with a single frequency input.

fs=48000; %sampling frequency in Hz
fc=8000; %angular frequency of the notch in Hz
fi=8000; %angular input frequency in Hz
w=2*pi*fc/fs;
a=2*cos(w);
u=0.01; %step size
r=0.90;
nT=0:255;
x=sin(2*pi*fi*nT/fs);%input sequence

x1=0;
x2=0;
y1=0;
y2=0;
a1=0;

for i=1:255,

   y(i)=x(i)-a*x1+x2+a*r*y1-r*r*y2;
   a1=a+2*u*y(i)*(x1-r*y1);

   a=a1;
   y2=y1;
   y1=y(i);
   x2=x1;
   x1=x(i);
```

```
end

subplot (2,1,1), plot(x);title ('Single Channel Adaptive
Notch Filter');
xlabel('No. of Samples');ylabel('Input Sequence');
subplot (2,1,2), plot(y);ylabel('Output Sequence');
```

```
%Filename :Single_drift.m
%Description : The file plot the output response of the
adaptive notch filter
%              with a single frequency input with frequency
drift
%           after 128 samples.

fs=48000; %sampling frequency in Hz
fc=8000; %angular frequency of the notch in Hz
fi=8000; %angular input frequency in Hz
w=2*pi*fc/fs;
a=2*cos(w);
u=0.01; %step size
r=0.90;
nT=0:127;
x=sin(2*pi*fi*nT/fs);%input sequence
x(128:255)=sin(2*pi*fi*nT/fs)% Frequency drift by 100Hz

x1=0; %Initialisation
x2=0;
y1=0;
y2=0;
a1=0;

for i=1:255,

   y(i)=x(i)-a*x1+x2+a*r*y1-r*r*y2;
   a1=a+2*u*y(i)*(x1-r*y1);

   a=a1;
   y2=y1;
```

```
    y1=y(i);
    x2=x1;
    x1=x(i);
end

subplot (2,1,1), plot(x);title ('Single Channel Adaptive
Notch Filter with Frequency Drift');
xlabel('No. of Samples');ylabel('Input Sequence');
subplot (2,1,2), plot(y);ylabel('Output Sequence');
```

```
%Filename :Single_multi.m
%Description : The file plot the output response of the
adaptive notch filter
%               with multiple frequency input

fs=48000; %sampling frequency in Hz
fc=8000; %angular frequency of the notch in Hz
fi1=8000; %angular input frequency in Hz
fi2=4000; % Second input frequency in Hz
w=2*pi*fc/fs;
a=2*cos(w);
u=0.01; %step size
r=0.90;
nT=0:255;
x=sin(2*pi*fi1*nT/fs)+sin(2*pi*fi2*nT/fs);%input sequence

x1=0; %Initialisation
x2=0;
y1=0;
y2=0;
a1=0;

for i=1:255,

   y(i)=x(i)-a*x1+x2+a*r*y1-r*r*y2;
   a1=a+2*u*y(i)*(x1-r*y1);

   a=a1;
   y2=y1;
   y1=y(i);
   x2=x1;
```

```
   x1=x(i);
end

subplot (2,1,1), plot(x);title ('Single Channel Adaptive
Notch Filter with Multiple Frequency Input');
xlabel('No. of Samples');ylabel('Input Sequence');
subplot (2,1,2), plot(y);ylabel('Output Sequence');
```

```
%Filename :Single_u001.m
%Description : The file plot the output response of the
adaptive notch filter
%              with a single frequency input and drift
%           after 128 samples.

fs=48000; %sampling frequency in Hz
fc=8000; %angular frequency of the notch in Hz
fi=8000; %angular input frequency in Hz
w=2*pi*fc/fs;
a=2*cos(w);
u=0.01; %step size
r=0.90;
nT=0:127;
x=sin(2*pi*fi*nT/fs);%input sequence
x(128:255)=sin(2*pi*fi*nT/fs)% Frequency drift by 100Hz

x1=0; %Initialisation
x2=0;
y1=0;
y2=0;
a1=0;

for i=1:255,

   y(i)=x(i)-a*x1+x2+a*r*y1-r*r*y2;
   a1=a+2*u*y(i)*(x1-r*y1);

   a=a1;
   y2=y1;
   y1=y(i);
```

```
    x2=x1;
    x1=x(i);
end

subplot (2,1,1), plot(x);title ('Single Channel Adaptive
Notch Filter with u=0.01');
xlabel('No. of Samples');ylabel('Input Sequence');
subplot (2,1,2), plot(y);ylabel('Output Sequence');
```

```
%Filename :cascade_ad.m
%Description : The file plot the output response of the
cascade adaptive notch filter
%              with a single frequency input.

fs=48000; %sampling frequency in Hz
fc=8000; %angular frequency of the notch in Hz
fi=8000; %angular input frequency in Hz
w=2*pi*fc/fs;
a=2*cos(w);
u=0.25; %step size
r=0.90;
nT=0:255;
x=sin(2*pi*fi*nT/fs);%input sequence

x1=0;
x2=0;
y1=0;
y2=0;
a1=0;
z1=0;
z2=0;

for i=1:255,

   y(i)=x(i)-a*x1+x2+a*r*y1-r*r*y2;
   z(i)=y(i)-a*y1+y2+a*r*z1-r*r*z2;
   a1=a+2*u*z(i)*(y1-r*z1);

   a=a1;
   an(i)=a1;
```

```
    y2=y1;
    y1=y(i);
    x2=x1;
    x1=x(i);
    z2=z1;
    z1=z(i);
end

subplot (2,1,1), plot(x);
title ('Cascade Adaptive Notch Filter with Single Input
Frequency ');
xlabel('No. of Samples');ylabel('Input Sequence');
subplot (2,1,2), plot(z);ylabel('Output Sequence');
```

```
%Filename :cascade_drift.m
%Description : The file plot the output response of the
cascade adaptive notch filter
%               with frequency drift.

fs=48000; %sampling frequency in Hz
fc=8000; %angular frequency of the notch in Hz
fi=8000; %angular input frequency in Hz
w=2*pi*fc/fs;
a=2*cos(w);
u=0.25; %step size
r=0.90;
nT=0:127;
x=sin(2*pi*fi*nT/fs);%input sequence
x(128:255)=sin(2*pi*(fi+200)*nT/fs);%Frequency drift by 200Hz
x1=0;
x2=0;
y1=0;
y2=0;
a1=0;
z1=0;
z2=0;

for i=1:255,

   y(i)=x(i)-a*x1+x2+a*r*y1-r*r*y2;
   z(i)=y(i)-a*y1+y2+a*r*z1-r*r*z2;
   a1=a+2*u*z(i)*(y1-r*z1);

   a=a1;
   an(i)=a1;
```

```
    y2=y1;
    y1=y(i);
    x2=x1;
    x1=x(i);
    z2=z1;
    z1=z(i);
end

subplot (2,1,1), plot(x);
title ('Cascade Adaptive Notch Filter with Frequency Drift
');
xlabel('No. of Samples');ylabel('Input Sequence');
subplot (2,1,2), plot(z);ylabel('Output Sequence');
```

```
%Filename :cascade_multi.m
%Description : The file plot the output response of the
cascade adaptive notch filter
%               with mulitple frequencies input.

fs=48000; %sampling frequency in Hz
fc=8000; %angular frequency of the notch in Hz
fi=8000; %angular input frequency in Hz
w=2*pi*fc/fs;
a=2*cos(w);
u=0.25; %step size
r=0.90;
nT=0:255;
x=sin(2*pi*fi*nT/fs)+sin(2*pi*1000*nT/fs);%input sequence

x1=0;
x2=0;
y1=0;
y2=0;
a1=0;
z1=0;
z2=0;

for i=1:255,

   y(i)=x(i)-a*x1+x2+a*r*y1-r*r*y2;
   z(i)=y(i)-a*y1+y2+a*r*z1-r*r*z2;
   a1=a+2*u*z(i)*(y1-r*z1);

   a=a1;
   an(i)=a1;
```

```
    y2=y1;
    y1=y(i);
    x2=x1;
    x1=x(i);
    z2=z1;
    z1=z(i);
end

subplot (2,1,1), plot(x);
title ('Cascade Adaptive Notch Filter with Multiple Frequency
Input');
xlabel('No. of Samples');ylabel('Input Sequence');
subplot (2,1,2), plot(z);ylabel('Output Sequence');
```

```
%Filename :multiple_freq.m
%Description : The file plot the output response of the
multiple channel adaptive notch filter

fs=48000; %sampling frequency in Hz
fc1=8000; %angular frequency of the notch in Hz
fc2=4000;
fi1=4000; %angular input frequency in Hz
fi2=8000;
w1=2*pi*fc1/fs;
w2=2*pi*fc2/fs;
a1=2*cos(w1);
a2=2*cos(w2)
u=0.001; %step size
r=0.90;
nT=0:255;
x=sin(2*pi*fi1*nT/fs)+sin(2*pi*fi2*nT/fs);%input sequence
x1=0;
x2=0;
y1=0;
y2=0;
a11=0;
a21=0;
z1=0;
z2=0;

for i=1:255,

   y(i)=x(i)-a1*x1+x2+a1*r*y1-r*r*y2;
   a11=a1+2*u*y(i)*(x1-r*y1);
   z(i)=y(i)-a2*y1+y2+a2*r*z1-r*r*z2;
```

```
    a21=a2+2*u*z(i)*(y1-r*z1);

    a1=a11;
    a2=a21;

    y2=y1;
    y1=y(i);
    x2=x1;
    x1=x(i);
    z2=z1;
    z1=z(i);
end

subplot (2,1,1), plot(x);
subplot (2,1,2), plot(z);
```

```
%Filename :multiple_drift.m
%Description : The file plot the output response of the
multiple channel adaptive notch filter
%               with frequency drift.

fs=48000; %sampling frequency in Hz
fc1=8000; %angular frequency of the notch in Hz
fc2=4000;
fi1=4000; %angular input frequency in Hz
fi2=8000;
w1=2*pi*fc1/fs;
w2=2*pi*fc2/fs;
a1=2*cos(w1);
a2=2*cos(w2)
u=0.001; %step size
r=0.90;
nT=0:127;
x=sin(2*pi*fi1*nT/fs)+sin(2*pi*fi2*nT/fs);%input sequence
x(128:255)=sin(2*pi*(fi1+100)*nT/fs)+sin(2*pi*(fi2+100)*nT/fs
);%Frequency drift by 100Hz
x1=0;
x2=0;
y1=0;
y2=0;
a11=0;
a21=0;
z1=0;
z2=0;

for i=1:255,
```

```
    y(i)=x(i)-a1*x1+x2+a1*r*y1-r*r*y2;
    a11=a1+2*u*y(i)*(x1-r*y1);
    z(i)=y(i)-a2*y1+y2+a2*r*z1-r*r*z2;
    a21=a2+2*u*z(i)*(y1-r*z1);

    a1=a11;
    a2=a21;

    y2=y1;
    y1=y(i);
    x2=x1;
    x1=x(i);
    z2=z1;
    z1=z(i);
end

subplot (2,1,1), plot(x);
title ('Multiple Channel Adaptive Notch Filter with Frequency
Drift');
xlabel('No. of Samples');ylabel('Input Sequence');
subplot (2,1,2), plot(z);ylabel('Output Sequence');
```

APPENDIX B
'C ' PROGRAM LISTING

```
//Filename : Bnotchs9.c
//Basic Notch Filter


#include <std.h>

#define r 0.9
#define a 1.0 // 2cos(2xpix(8KHz/48KHz))

void main(void)
{

float output[512];
float input[512];
float y = 0;
float x = 0;
float x1 = 0;
float x2 = 0;
float y1 = 0;
float y2 = 0;



     for (i=0; i<512; i++)
     {
          x=input[i]; //Input signal file
          y = x-a*x1+x2+a*r*y1-r*r*y2;//Basic Notch filter
          y2 = y1;
          y1 = y;
          x2 = x1;
          x1 = x;

          output[i]=y; //Output to graphic Display
     }

}
```

```
//Filename : Anotch1
//Adaptive Notch Filter

#include <stdio.h>

#define r 0.9
#define u 0.1 //Step Size


float output[200];
float input[200];

void main(void)
{


float y = 0;
float x = 0;
float x1 = 0;
float x2 = 0;
float y1 = 0;
float y2 = 0;
float a = 1;
float a1 = 0;
int i;

    printf("input...../n");//Insert data file with signal
input

  for (i=0; i<200; i++)
  {
```

```
      x=input[i];
      y = x-a*x1+x2+a*r*y1-r*r*y2;
        a1 = a+2*u*y*(x1-r*y1); //Adaptive parameter


      y2 = y1;
      y1 = y;
      x2 = x1;
      x1 = x;
      a = a1;


         output[i]=y;
   }

    printf("output...../n");//Output to Graphic display
}
```

```
//Filename : Cnotch
//Cascade Section Notch Filter


#include <stdio.h>

#define r 0.9
#define u 0.25 /*Step Size*/

float output[200];
float input[200];

void main(void)
{

float q = 0;
float q1 = 0;
float q2 = 0;
float y = 0;
float x = 0;
float x1 = 0;
float x2 = 0;
float y1 = 0;
float y2 = 0;
float a = 1.0;
float a1 = 0;
int i;


    printf("input...../n"); //Input Signal data file
```

```
for (i=0; i<200; i++)
{
   x=input[i];

   q = x-a*x1+x2+a*r*q1-r*r*q2; //Adaptive algorithm
   y = q-a*q1+q2+a*r*y1-r*r*y2;
     a1 = a+2*u*y*(q1-r*y1);

       q2 = q1;
   q1 = q;
   x2 = x1;
   x1 = x;
     y2 = y1;
     y1 = y;

   a = a1;

       output[i]=y;
}

  printf("output...../n");//Output Graphic Display
```

```
//Filename : Mnotch
//Multiple Notch Filter

#include <stdio.h>

#define r 0.9
#define u 0.25 //Step Size

float output[200];
float input[200];

void main(void)
{
float q = 0;
float q1 = 0;
float q2 = 0;
float y = 0;
float x = 0;
float x1 = 0;
float x2 = 0;
float y1 = 0;
float y2 = 0;
float a = 1.73;
float a1 = 0;
float b = 1.73;
float a2 = 0;
int i;

    printf("input...../n");//Input Signal Data file

  for (i=0; i<200; i++)
```

```
   {
      x=input[i];

      q = x- a*x1+x2+a*r*q1-r*r*q2; //Adaptive algorithm
      a1 = a+2*u*y*(q1-r*y1);
      y = q- b*q1+q2+b*r*y1-r*r*y2;
      a2 = b+2*u*y*(q1-r*y1);

         q2 = q1;
      q1 = q;
      y2 = y1;
      y1 = y;
      x2 = x1;
      x1 = x;
      a = a1;
      b = a2;

        output[i]=y;

   }

    printf("output...../n");//Output to graphic display
}
```

```
//Filename : Sine.C
//the program use to generate single frequency input signal
// data file to be used in code composer simulation#include
<stdio.h>


#include <stdlib.h>
#include <math.h>
#include <conio.h>

float f(float t, unsigned int flag)
{
  float y;
  static float f,a;
  if(flag!=0)
  {
    printf("Sine wave function initialisation.");
    printf("\nEnter frequency (Hz) : ");
    scanf("%f",&f);
    printf("\nEnter amplitude (0-16384) : ");
    scanf("%f",&a);
  }
```

```
//Filename : Msine.C
//the program use to generate mutliple frequency input signal
// data file to be used in code composer simulation

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>

float f(float t, unsigned int flag)
{
  float y;
  static float f,f1,a;
  if(flag!=0)
  {
    printf("Sine wave function initialisation.");
    printf("\nEnter frequency (Hz) : ");
    scanf("%f",&f);
    printf("\nEnter frequency (Hz) : ");
    scanf("%f",&f1);
    printf("\nEnter amplitude (0-16384) : ");
    scanf("%f",&a);
  }

  y=a*(sin(2*3.142*f*t)+sin(2*3.142*f1*t));
  return(y);
}

void main(void)
{
  char out_file[10];
```

```
float T,y;
int N,n;
FILE *fp;

while(1)
{
  printf("\nEnter output filename : ");
  fflush(stdin);
  scanf("%s",out_file);
  fp=fopen(out_file,"a");
  if (fp!=0)
    break;
  printf("\nCannot open file '%s'.",out_file);
}

printf("\nEnter sampling period : ");
scanf("%f",&T);

while(1)
{
  printf("\nEnter no. of output samples : ");
  scanf("%d",&N);
  if (N==0)
    exit(0);
  f(0,1);

  for (n=0;n<N;n++)
  {
    y=f(n*T,0);
    fprintf(fp,"%d\n",(int)y);
  }
```

```
    }
    exit(0);
}
```