BLEKINGE INSTITUTE OF TECHNOLOGY
APPLIED IT IN FOCUS

Implementation of Adaptive Filter Structures on a Fixed Point Signal Processor for Acoustical Noise Reduction

By

Krishna Chaitanya Chunduri(810507-P214)

Chalapathi Gutti(780208-P898)

Master thesis number, MEE 05:33

Supervised by

Benny Sällberg

THESIS

Presented to the department of signal processing

Blekinge Institute of Technology

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

BLEKINGE INSTITUTE OF TECHNOLOGY

ACKNOWLEDGEMENTS

We would like to thank our thesis advisor Benny Sällberg (phd student) for providing us with the opportunity to work over the past six months in the Signal Processing Laboratory, Department of Electrical Engineering. We thank him for the guidance he has provided and the confidence he has shown in our work. We also thank him for the spontaneous replies from him when we had queries and providing us necessary materials when we are in need of them.

We thank **Analog devices** for providing an evaluation chip for the thesis. We thank Benny Sällberg for providing the addon card, head sets, amplifiers required for the thesis.

We would also like to thank all the members of the Signal Processing Department for their timely help and support.

Index

Chapters *page no.*

Abstract

1. Introduction 5
    1.1 Why we used fixed point processor 7
    1.2 Thesis outline.....8

2. Problem Formulation 9

3. Fixed Point Arithmetic 18
    3.1 Fixed point representation 18
    3.2 Two's compliment summary 19
    3.3 Dynamic range and precision 20
    3.4 Conversion 21
    3.5 Numerical operations..... 22
        3.5.1 Scalar multiplication 22
        3.5.2 Finite impulse response filters..... 23
        3.5.3 Recursive filters..... 24
        3.5.4 Division 25

4. Real Time Implementation 27
    4.1 ADUC 7026 processor 27
        4.1.2 Configuration and features 28
    4.2 ANC implementation on a Real time processor..... 31
        4.2.1 System identification 31
        4.2.2 Pseudo noise generator..... 31
        4.2.3 FXLMS adaptation..... 33
        4.2.4 Experimental details..... 34

5. Evaluation and Results..... 35
    5.1 Evaluation 35
    5.2 Results 37

6. Further Research 51

7. Summary and Conclusions..... 52

ABSTRACT

The problem of controlling the noise level in the environment has been the focus of a tremendous amount of research over the years. Active Noise Cancellation (ANC) is one such approach that has been proposed for reduction of steady state noise. ANC refers to an electromechanical or electro acoustic technique of canceling an acoustic disturbance to yield a quieter environment. The basic principle of ANC is to introduce a canceling "anti-noise" signal that has the same amplitude but the exact opposite phase, thus resulting in an attenuated residual noise signal. Wideband ANC systems often involve adaptive filter lengths, with hundreds of taps. Using sub band processing can considerably reduce the length of the adaptive filter. This thesis presents Filtered-X Least Mean Squares (FXLMS) algorithm to implement it on a fixed point digital signal processor (DSP), ADUC7026 micro controller from Analog devices. Results show that the implementation in fixed point matches the performance of a floating point implementation.

CHAPTER 1

INTRODUCTION

Acoustic noise have increased in magnitude due to noisy engines, heavy machinery, pumps, high speed wind buffeting and several other noise sources. Exposure to high sound pressure levels may damage humans from both a physical and a psychological aspect. The problem of controlling the noise level in the environment has been the focus of a tremendous amount of research over the years.

The classical approach to noise cancellation is a passive acoustic approach. Passive silencing techniques such as sound absorption and isolation are inherently stable and effective over a broad range of frequencies provided that the thickness of the insulator is larger than wave length of the signal to insulate. However, passive techniques tend to be expensive, bulky and generally ineffective for canceling noise at the lower frequencies. The performance of these systems is also limited to a fixed structure and proves impractical in a number of situations where space is at a premium and the added bulk can be a hinder. The shortcomings of the passive noise reduction methods have given impetus to the research and applications using alternate methods of controlling noise in the environment.

Various signal processing techniques have been proposed over the years for noise reduction in the environment. The explosive growth of digital processing algorithms and technologies has resulted in an opportunity to implement active noise controlling techniques in real applications. Digital Signal Processors (DSP) have shrunk tremendously in size while their processing capabilities have grown exponentially. At the same time the power consumption of these DSPs has steadily

decreased following the path laid down by Gene's law. This has enabled the use of DSPs in a variety of portable hearing enhancement devices such as hearing aids, headsets, hearing protectors, etcetera.

There are two different approaches for noise reduction. The first approach is passive noise reduction techniques. Passive techniques can be found in hearing aids, cochlear implants etcetera and uses a microphone to record exterior sound. The recorded sound is processed using signal processing techniques and a clean restored signal is output through a loudspeaker to the listener. One of the important assumptions of this technique is that the listener is acoustically isolated from the environment. This assumption is however not valid in a large number of situations particularly those where the ambient noise has a very large amplitude. In such situations, the second approach of Active Noise Cancellation (ANC) is applicable. ANC refers to an active electromechanical or electro acoustic technique of canceling acoustic disturbance by emitting controlled sounds to yield a quieter environment. The basic principle of ANC is to introduce a canceling "anti-noise" signal that has the same amplitude but the exact opposite phase of the disturbance, thus resulting in an attenuated residual noise signal. ANC has been used in a number of applications such as hearing protectors, headsets, etcetera.

The traditional wideband ANC algorithms work best in the lower frequency bands but when lager filter lengths are used, the algorithm may not converge desirably fast. Further, as the ANC system is combined with other communication and sound systems, it is necessary to have a frequency dependent noise cancellation system to avoid adversely affecting the desired signal.

1.1 Why we used fixed point processor?

This section will discuss the advantages and disadvantages of fixed point processors when compared with floating point processors.

1) First, the integer number representation format is straightforward in that it represents integer numbers from 0 up to the largest whole number that can be represented with the available number of bits. What you refer to is fractional representation commonly used in fixed point arithmetics, there you may represent numbers between -1 and 1 with a 'binary point' assumed to lie just after the most significant bit. The most significant bit in both cases carries the sign of the number.

- The size of the fraction represented by the smallest bit is the precision of the fixed point format.
- The size of the largest number that can be represented in the available word length is the dynamic range of the fixed point format

Floating point format has the remarkable property of automatically scaling all numbers by moving, and keeping track of, the binary point so that all numbers use the full word length available but never overflow. Floating point numbers have two parts: the mantissa and the exponent. The mantissa is similar to the fixed point part of the number, and an exponent which is used to keep track of how the binary point is shifted. Every number is scaled by the floating point hardware:

- If a number becomes too large for the available word length, the hardware automatically scales it down, by shifting it to the right
- If a number is small, the hardware automatically scale it up, in order to use the full available word length of the mantissa

In both cases the exponent is used to count how many times the number has been shifted. In floating point numbers the binary point comes after the second most significant bit in the mantissa.

Secondly, coding is time consuming and difficult in fixed point processors due to eventual scaling to prevent arithmetic over flow when compared with floating point processors.

3) Finally, fixed point processors have a majority of market shares as opposed to floating point processors. Mainly due to their power efficiency and price awareness as is very important in many industrial applications. Floating point processors have most of their applications in scientific and research purposes but some industries use floating point applications as well.

This thesis has three major implementation parts:

1. Implementation of a fixed point and a floating point arithmetic on a personal computer using matlab software.
2. Implementation of a fixed point arithmetic active noise canceller on a personal computer using c programming.
3. Implementation of fixed point arithmetic active noise canceller in real time on a digital signal processor.

The outline of the thesis is as fallows:-

Chapter two describes the actual problem and a suitable algorithm to implement it. Chapter three summarizes fixed point arithmetic. Chapter four discusses real time implementation on a fixed point processor. Chapter five discusses evaluation and results and chapter six gives an introduction to further research.

CHAPTER 2

PROBLEM FORMULATION

ANC traditionally involves passive methods such as enclosures, barriers and silencers to attenuate noise. These techniques use either the concept of impedence change or the energy loss due to sound absorbing materials. These methods are however not effective for low frequency noise. A technique to overcome this problem is ANC, which is sound field modification by electracoustic means. ANC is an electro-acoustic system that cancels the primary unwanted noise by introducing a canceling "antinoise" of equal amplitude but opposite phase, thus resulting in an attenuated residual noise signal as shown in Figure 2.1.

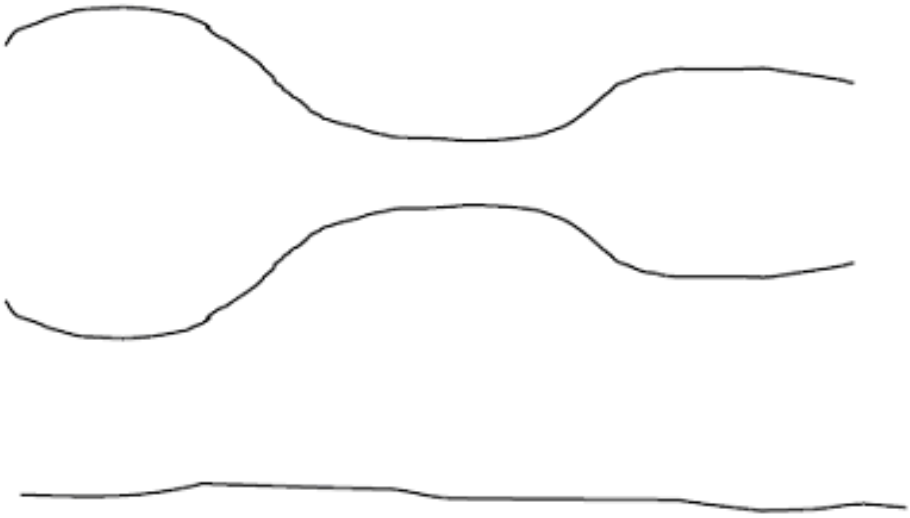


Figure 2.1 Wave fields in Active Noise Control, Primary noise waveform (upper), secondary noise waveform (middle) and residual noise waveform (lower).

Adaptive algorithms can be used in active noise control applications. It continuously adjusts its coefficients such that an estimate of the noise is produced and cancels the unwanted noise.

Adaptive filters are normally defined for problems such as electrical noise canceling where the filter output is an estimate of a desired signal. In control applications, however, the adaptive filter works as a controller controlling a dynamic system containing actuators and amplifiers etcetera. The estimate (anti-vibrations or anti-sound) in this case can thus be seen as the output signal from a dynamic system, i.e. a forward path. Since there is a dynamic system between the filter output and the estimate, the selection of adaptive filter algorithms must be made with care. A conventional adaptive algorithm such as the LMS algorithm is likely to be unstable in this application due to the phase shift (delay) introduced by the forward path. The well-known filtered-XLMS (FXLMS) algorithm is, however, an adaptive filter algorithm which is suitable for active control applications. The forward path is estimated by using system identification and with the results of the system identification, primary channel is estimated by using FXLMS adaptation. FXLMS algorithm is developed from the LMS algorithm, where an estimate of the forward path is introduced in the filter coefficient adaptation. The forward path is the dynamical system from the output of the filter to the error. That means a forward path is introduced between the input signal and the algorithm for the adaptation of the coefficient vector. Figure 2.2 shows an adaptive filter with a forward path introduced.

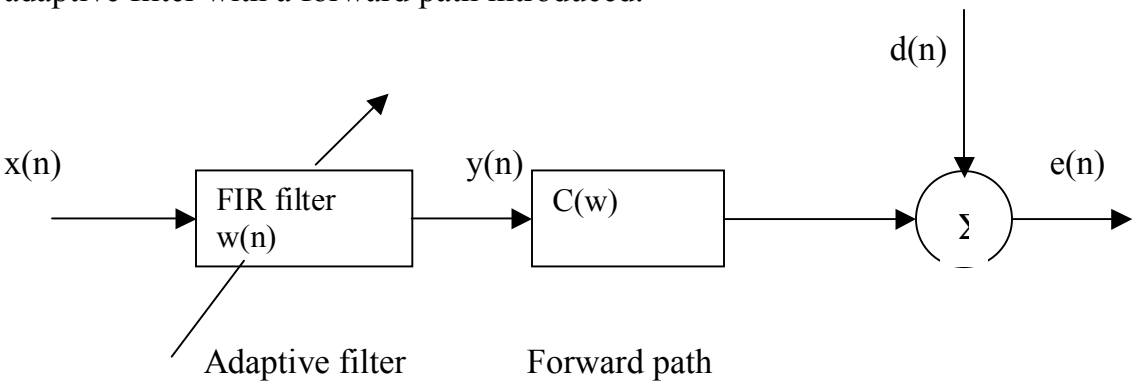


Figure 2.2 Active noise control system with an additional forward path

A digital ANC employing the FXLMS can use Finite Impulse Response (FIR) filters in the adaptive filter and the forward path estimate. The Finite Response Filter (FIR) output is given by the vector inner product according to

$$y(n) = W^T(n).X(n) \tag{2.1}$$

where

$$X(n) = [x(n), x(n-1), ..., x(n-M+1)]^T \tag{2.2}$$

is the input signal vector to the adaptive filter and

$$W(n) = [w_0(n), w_1(n), ...w_{M-1}(n)]^T \tag{2.3}$$

is the adjustable filter coefficient vector. In control applications, the estimation error e(n) is defined as the difference between the desired signal (desired response) d(n) and the output signal from the forward path or plant under control, according to

$$e(n) = d(n) - y_c(n) \tag{2.4}$$

Assuming that the forward path estimate can be expressed by an Ith order FIR filter according to

$$h_c(n) = \begin{cases} c_n \text{ when } n \in \{0, ..., I-1\} \\ \\ 0 \text{ otherwise.} \end{cases} \tag{2.5}$$

it follows that the estimation error e(n) can be expressed as

$$e(n) = d(n) - \sum_{i=0}^{I-1} c_i \sum_{m=0}^{M-1} w_m(n-i)x(n-i-m) \tag{2.6}$$

The Wiener (Minimum Mean Square Error) solution of the coefficient vector is obtained by minimizing the quadratic function

$$J(n) = E[e(n)^2] \tag{2.7}$$

and this can be carried out by using the gradient vector of the mean square error

$$\nabla_{w(n)} J(n) = 2E[e(n)\nabla_{w(n)} e(n)] \tag{2.8}$$

By taking advantage that the desired signal d(n) is independent of the filter coefficients, the gradient vector of the estimated error can be expressed as

$$\nabla_{w(n)} e(n) = \begin{pmatrix} -\sum_{i=0}^{I-1} c_i x(n-i) \\ -\sum_{i=0}^{I-1} c_i x(n-i-1) \\ . \\ . \\ -\sum_{i=0}^{I-1} c_i x(n-i-M+1) \end{pmatrix} \tag{2.9}$$

By inserting this expression in Eq. 2.8, following relation can be obtained for the gradient vector of the mean square error

$$\nabla_{w(n)} J_f(n) = -2E[e(n)X_C(n)] \tag{2.9.1}$$

where $x_c(n)$ is given by

$$x_c(n) = \begin{pmatrix} \sum_{i=0}^{I-1} c_i x(n-i) \\ \sum_{i=0}^{I-1} c_i x(n-i-1) \\ . \\ . \\ \sum_{i=0}^{I-1} c_i x(n-i-M+1) \end{pmatrix} \tag{2.10}$$

In other words, an LMS algorithm with a gradient estimate as in

$$\nabla_{w(n)} J(n) = -2e(n)x_c(n) \tag{2.11}$$

would solve the problem of producing an estimate via a dynamic system. From this it follows that the conventional LMS algorithm is likely to be unstable in control applications. The conventional LMS algorithm will in some cases also find a poor solution when it converges. This can be explained by the fact that the LMS algorithm uses a gradient estimate x(n)e(n) which is not correct in the mean.

A compensated algorithm is obtained by filtering the reference signal to the coefficient adjustment algorithm using a model of the forward path as illustrated in Fig. 2.2. The algorithm obtained is the well-known filtered-x LMS algorithm defined by Eq. 2.4:

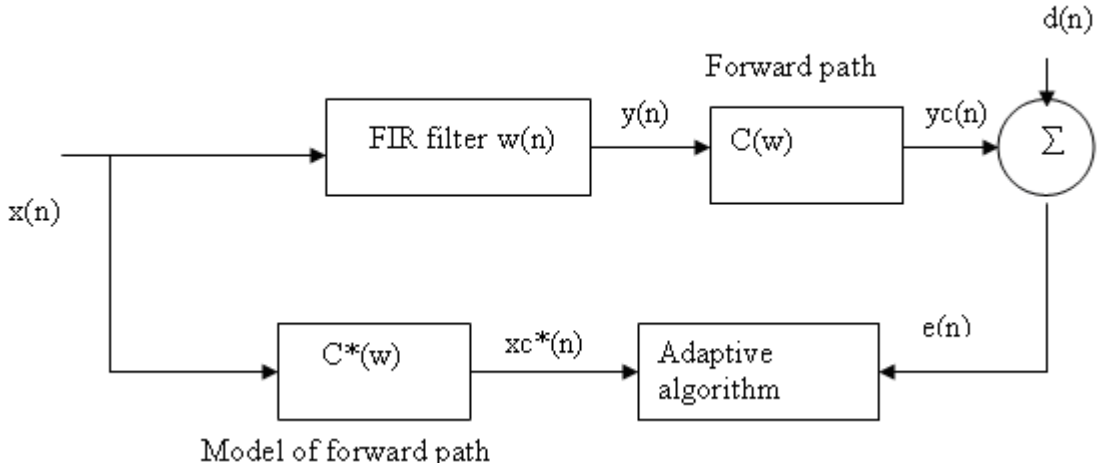


Figure 2.3: Active control system with a controller based on the filtered-x LMS-algorithm[5].

The filter update coefficient is

$$w(n+1) = w(n) + \mu x_C^{*}(n)e(n) \tag{2.12}$$

Here c_i^* is the coefficient of an estimated FIR filter model of the forward path:

It is in practice customary to use an estimate of the impulse response for the forward path. As a result, the reference signal $x_C^*(n)$ will be an approximation, and differences between the estimate of the forward path and the true forward path influence both the stability properties and the convergence rate of the FXLMS algorithm. However, the algorithm is robust to errors in the estimate of the forward path. The model used should introduce a time delay corresponding to the forward path at the dominating frequencies. In the case of narrow-band reference signals to the algorithm, e.g. $\sin(w_0 t)$, the algorithm will converge with phase errors in the estimate of the forward path with up to ±90, provided that the step length μ is sufficiently small. Furthermore, phase errors in the estimate of the forward path smaller than ±45 will have only a minor influence on the algorithm convergence rate.

The FXLMS algorithm relies principally on the assumption that the adaptive FIR filter and the forward path "commute". This is approximately true if the adaptive filter varies in a time scale which is slow in comparison with the time constant for the impulse response of the forward path. This expression can be written as follows:

$$\sum_{i=0}^{I-1} c_i \sum_{m=0}^{M-1} w_m(n-i)x(n-i-m) \approx \sum_{m=0}^{M-1} w_m(n) \sum_{i=0}^{I-1} c_i x(n-m-i) \tag{2.13}$$

where

$$w(n) \approx w(n-i), i \in \{1,2,...,I-1\} \tag{2.14}$$

where I is the length of the impulse response of the forward path. In practice, the FXLMS algorithm exhibits stable behavior even when the coefficients change within the time scale associated with the dynamic response of the forward path . In order to ensure that the action of an LMS algorithm is stable the maximum value for the step length μ should be given approximately by:

$$\mu < \frac{2}{ME[x^2(n)]} \tag{2.15}$$

However, in the case of the FXLMS algorithm, Elliot et al[10]. have found that the maximum step length μ not only depends on the length of the adaptive filter and the variance of the filtered reference signal but also on the delays in the forward path C. If the reference signal $x_C^*(n)$ is a white noise process it has thus been found that an upper limit for the step length μ is given by

$$\mu_{max} \approx \frac{2}{E[x^2_{C^*}(n)](M + \delta)} \tag{2.16}$$

where δ is the overall delay in the forward path (in samples). In the case of a non-white reference signal Elliot et al,[10] suggest that $\frac{1}{\mu_{max}}$ is proportional to 1.2M and not 0.5M .The probable explanation is that the covariance matrix for the reference signal will have a poor conditioning .

This broadband ANC system utilizes two main structures. First, an adaptive system identification framework is used to estimate the forward path as shown in fig.2.4. The estimated forward path coefficients are stored in memory and used in the later FXLMS adaptation for noise cancellation. That is, the algorithm requires certain knowledge of the forward path before being able to actively cancel noise.

Essentially, an adaptive filter W(z) is used to estimate an unknown plant C(z) which consists of the acoustic response from the reference sensor to the error

sensor. The objective of the adaptive filter W(z) is to minimize the residual error signal e(n). However, the main difference from the traditional system identification scheme is the use of an acoustic summing junction instead of the subtraction of electrical signals.

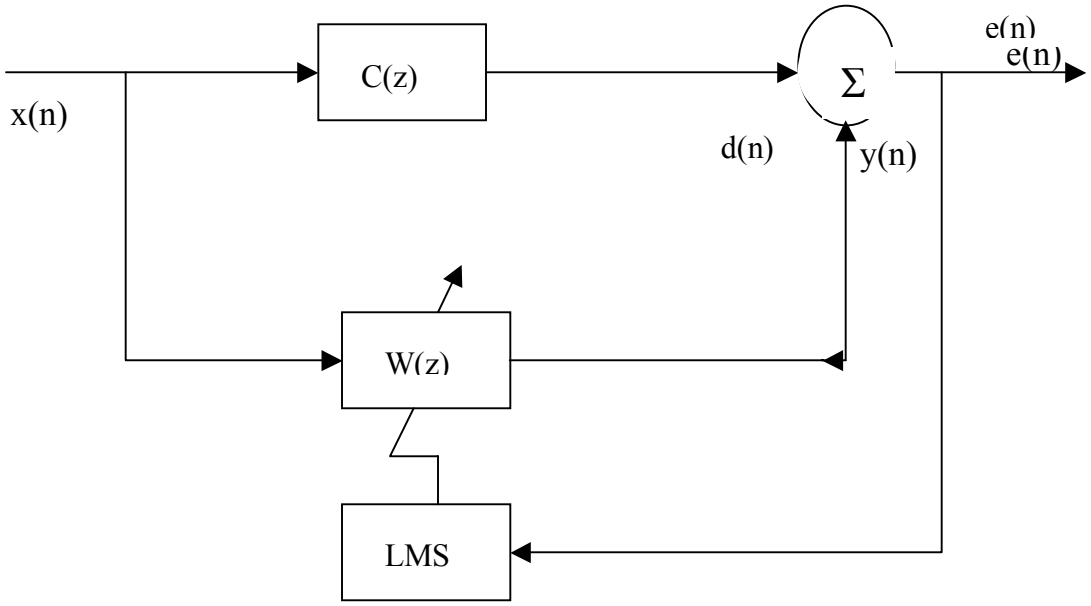


Fig2.4 . Reference framework for systemidentification using an adaptive filter.

In system identification, a random white noise can be generated in the loud speakers of the hearing defenders where the x(n) is the signal from the speakers of the head phones and d(n) is the signal taken from the error microphone. Both x(n) and d(n) signals are given as the inputs to the LMS algorithm. The LMS algorithm steers filter coefficients such that the estimate of the forward path is obtained. Hence, an estimate of the forward path is obtained, C*(z), and can be used in FXLMS adaptation.

The introduction of the secondary path transfer function in a system using the standard LMS algorithm leads to instability. This is because, it is impossible to compensate for the inherent delay due to C(z) if the primary path P(z) does not contain a delay of equal length. Also, a very large FIR filter would be required to

effectively model 1/C(z). This can be solved by placing an identical filter in the reference signal path to the weight update of the LMS equation. This is known as the filtered-X LMS algorithm. The block diagram of an ANC system using the FXLMS algorithm is shown in Figure 2.5

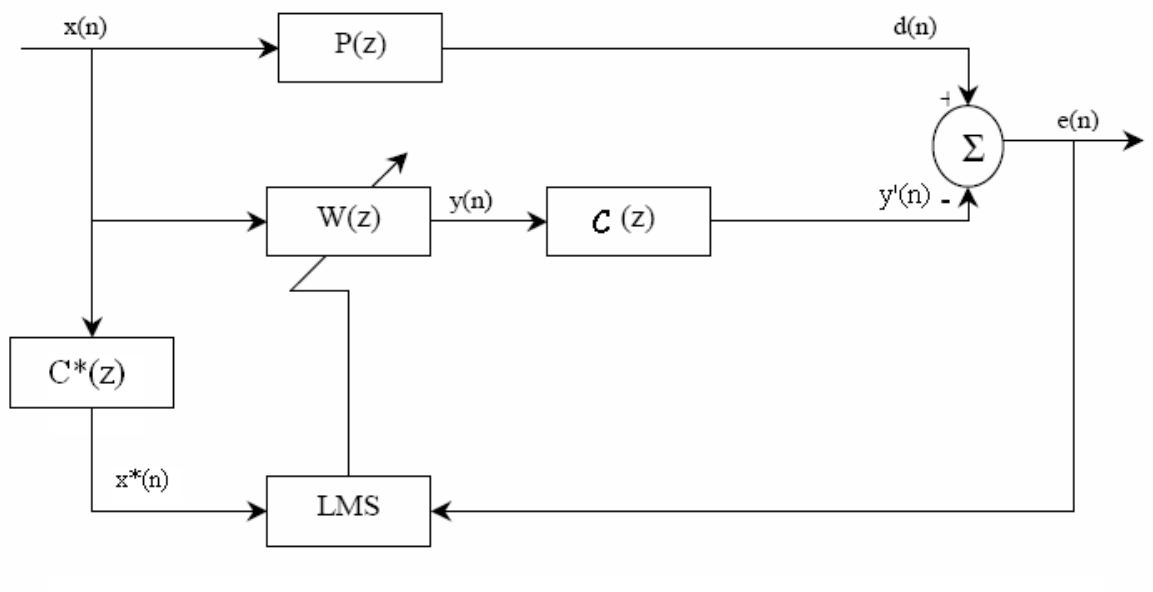


Figure 2.5 Schematic diagram of ANC system using FXLMS algorithm

CHAPTER 3

FIXED POINT ARITHMETIC

3.1 ***Fixed Point Representation***

Given that there are processors that do not support floating point numbers, how can a fraction number 0.5 can be represented? This is where fixed point math comes into play. As the name implies, a fixed point number places the "decimal" point between the whole and fractional parts of a number at a fixed location, providing *f* bits of fractional precision. For example, an 8.24 fixed point number has an eight bit integer portion and twenty four bits of fractional precision. Since the split between the whole and fractional portion is fixed, it is known exactly what the range and precision will be.

Using a 16.16 fixed point format (which, handily enough, fits in 32-bit integer), the high 16-bits represent the whole part, and the low 16-bits represent the fractional portion, according to the hexadecimal number

0xWWWWFFFF

With 16-bits in the whole part, it can be represented 2^{16} (65536) discrete values (0-65535 unsigned or -32768 to +32767 signed). The fractional part gives us, 2^{16} steps to represent the values from 0 to 65535/65536 or approximately 0.99999. Our fractional resolution is 1/65536, or about 0.000015

The 16.16 fixed point value in hexa decimal format is 0x00104ACF i.e. approximately equals the decimal value 16.29222. The high sixteen bits are 0x0010 (16 decimal) and the low 16-bits are 0x4ACF(19151), so 19151/65536.0

~= 0.29222. The simple method to convert from a fixed point value of *f* fractional bits is to divide by 2^f so:

$$\frac{0x00104ACF}{65536.0} \approx 16.29222$$

However, certain care must be exercised if integers are stored in 2's complement form. The value decimal value -16.29222 corresponds to a hexa decimal value is 0xFFEFB531, notice that the fractional bits (0xB531) are *not* the same as for the positive value. So, it is not possible to just mask the fractional bits directly, sign matters.

3.2 Two Complement Summary

Fundamentally 2's complement solves the problem of representing negative numbers in binary form. Given a 32-bit integer, how negative numbers can be represented?

The obvious method is to incorporate a *sign bit*, which is precisely what floating point numbers in IEEE-754 format do. This signed magnitude form reserves one bit to represent positive/negative, and the rest of the bits represent the number's magnitude. Unfortunately there are a couple problems with this.

1. The value of zero has two representations, positive (0x00000000) and negative (0x80000000), making comparisons cumbersome. It's also wasteful since two bit configurations represent the same value.
2. Adding a positive integer to a negative integer requires logic. For example, -1 (0x80000001 in 32-bit sign magnitude form) summed with +1 (0x00000001 in 32-bit sign magnitude form) yields 0x8002, or -2 in 16-bit sign-bit form..

Researchers came up with a much better system for number representation called 2's complement arithmetic. In this system positive numbers are represented as usual, how ever negative numbers are represented by *their complement, plus one*. This requires taking all bits in the positive representation, reversing them, then adding one.

So to encode the decimal value -1, its positive hexadecimal representation 0x00000001 is inverted (complemented) to 0xFFFFFFFE and added by one to give us 0xFFFFFFFF, which may be recognized as the two-complements form of the decimal value -1.

3.3 Dynamic Range and Precision

Different operations often require different amounts of range and precision. A format like 16.16 is in general sufficient, but depending on the specific problem, highly precise formats such as 2.30, or longer range formats such as 28.4 are needed. For example, the sine function only returns values in the range -1.0 to +1.0, so representing sine values with a 16.16 is wasteful since 15-bits of the integer component will always be zero. Not only is that, but the difference between sine(90*π/180) and sine(89.9*π/180) a very small number, ~.0000015230867, which our 16-bits of fractional precision cannot represent accurately. In this particular case, 2.30 format would be more suitable.

This brings to light a serious problem with fixed point; overflow of underflow can occur without expecting it, especially when dealing with sums-of-products or power series. This is one of the most common sources of errors when working on a fixed point code base.

A very common case is vector normalization, which has a sum-of-products during the vector length calculation. Normalizing a vector consists of dividing each of the

vector's elements by the length of the vector. The length of a vector v=(vx vy vz)^T is:

$$\|v\|_2 = \sqrt{v_x v_x + v_y v_y + v_z v_z} \quad (3.1)$$

This is why choosing the range and precision is very important and putting in a *lot* of range and overflow sanity checks is extremely important if robust software is needed.

3.4 Conversion

A fixed point value can be derived from a floating point value by multiplying by 2^*f* (it means it will be raised to f and after that it will be truncated). A fixed point value is generated from an integer value by shifting left *f* bits:

F = [r * 2^f] where [] corresponds to truncation

r ~= F/2^f

$$F = i \ll f \quad (3.1)$$
$$F = r\,2^f$$

- F = fixed point value
- I = integer value
- r = floating point value
- f = number of fractional bits for F

Converting back to floating point is trivial – just perform the reverse of the float-to-fixed conversion:

$$r = \frac{F}{2^f} \quad (3.2)$$

However, converting to an integer is a bit trickier due to rounding rules and the 2's complement encoding. In the world of floating point there are four common methods for converting a floating point value to an integer:

1. round-towards-zero
2. round-towards-positive-infinity
3. round-towards-negative-infinity
4. round-to-nearest

3.5 Numerical Operations

Multiplication and division are most prominent in Numerical operations of fixed point arithmatics.

3.5.1 Scalar Multiplication

Consider the scalar multiplication

$y = x \cdot \alpha$,

where x is an integer value and the constant α is an arbitrary floating point value. Depending on the constant, α, the output, y, does not necessary has to be an integer value, but instead a floating point. This operation can be approximated in fixed point, two cases exists:

CASE I

The first case is valid for, $|\alpha| \leq 0.5$.

$$A = \lfloor \alpha \cdot 2^{16} \rfloor \tag{3.3}$$
$$z = x \cdot A \tag{3.4}$$
$$y = \lfloor z \cdot 2^{-16} \rfloor. \tag{3.5}$$

CASE II

The second case is valid for, $|\alpha| > 0.5$

$$A = \lfloor \alpha \cdot 2^{p} \rfloor \tag{3.6}$$
$$z = x \cdot A \tag{3.7}$$
$$y = \lfloor z \cdot 2^{-p} \rfloor \tag{3.8}$$

where p should be selected such that $|\alpha| \cdot 2^{p}$ is as close as possible to 2^{15}, although not exceeding 2^{15}. Note that shifts other than p=8 and p=16 can be costly in terms of number of instructions required. Hence, it is wise to optimize the implementation such that it is possible to use either p=8 or p=16.

3.5.2 Finite Impulse Response Filters

Assume a Finite Impulse Response (FIR) filter is to be implemented according to

$$y(n) = \sum_{k=0}^{L-1} h_k \cdot x(n-k) \tag{3.9}$$

where h_k are the filter coefficients and x(n) is the input signal. Directly implementing the FIR filter is often not possible since the coefficients are likely

floating point values. However, using the Case-I-reformulation under the Section "Scalar Multiplication", it can be derived that

$$H_k = \lfloor h_k \cdot 2^{16} \rfloor \tag{3.10}$$

$$z(n) = \sum_{k=0}^{L-1} H_k \cdot x(n-k) \tag{3.11}$$

$$y(n) = \lfloor z(n) \cdot 2^{-16} \rfloor \tag{3.12}$$

.

This operation is allowed if the filter coefficients are fulfilling the requirement $|h_k| \le 0.5$.

3.5.3 Recursive Filters

A recursive filter, often denoted Infinite Impulse Response (IIR), can be implemented on fixed point, but it requires some more attention than the FIR filters do. Example of a problem in recursive filtering is limit cycle oscillations. There are two ways of circumventing the problems of fixed point recursive filters: 1) use high enough multiplier, 2) feed forward of the remainder. For the discussion, consider the following first order Auto Regressive Moving Average (ARMA) process

$$y(n) = \alpha \cdot y(n-1) + \beta \cdot x(n) \tag{3.13}$$

where α and β are the process coefficients. For simplicity it can be assumed that $|\alpha| \le 0.5$ and $|\beta| \le 0.5$.

CASE I

$$A = \lfloor \alpha \cdot 2^{16} \rfloor \tag{3.14}$$

$$B = \lfloor \beta \cdot 2^{16} \rfloor \tag{3.15}$$

$$z(n) = A \cdot y(n-1) + B \cdot x(n) \tag{3.16}$$

$$y(n) = \lfloor z(n) \cdot 2^{-16} \rfloor \tag{3.17}$$

CASE II

$$A = \lfloor \alpha \cdot 2^{16} \rfloor \tag{3.18}$$

$$B = \lfloor \beta \cdot 2^{16} \rfloor \tag{3.19}$$

$$z(n) = A \cdot y(n-1) + B \cdot x(n) + R(n-1) \tag{3.20}$$

$$y(n) = \lfloor z(n) \cdot 2^{-16} \rfloor \tag{3.21}$$

$$R(n) = z(n) - y(n) \cdot 2^{16} \tag{3.22}$$

Where R(n) is the remainder.

Note that for the second case, care must be taken such that the remainder does not suffer from sign sensitivity. The second case is much more robust than the first case, but it requires some more instructions and memory storage (the remainder).

3.5.4 Division

Division is similar to multiplication. Consider the following example. Take 2.0 and 1.0 again in 16.16 fixed point form: 0x20000 and 0x10000. If the former is divided by the latter, the resulting value will be 2. Not 0x20000. Which is correct, granted, but it is not in a 16.16 output form. The result is taken and shifted back by the number of bits that are needed, needed. However, shifting after the integer division means that a lost in precision is achieved. But...that's wrong, since shifting *after* the integer divide means losing precision before the shift. In other

words, if A=0x10000 and B=0x20000 The final result would be 0, instead of 0x8000 (0x8000 = 0.5 in 16.16 fixed point, since 0x8000/65536.0 = 0.5). But if shift is performed integer division, it can be managed to retain our precision and also have a properly scaled result.

But it is assumed that A and B are of the same fractional precision. Consider the folowinng formulation of division

$$\frac{A2^m}{B2^n} = \frac{A}{B}2^{m-n} \tag{3.23}$$

What this says is two fixed point numbers are taken of *m* and *n* fractional bits, and were divided, the quotient will be a value with *m-n* fractional bits. With the previous example of 16.16 divided by a 16.16, that means 0 fractional bits. If *f* fractional bits are expected, the numerator must be scaled by 2^*f-(m-n)*. Ths means, the numerator is scaled before the division operation is done. So the final equation looks like:

$$C2^f = \frac{A\,2^m 2^{f-(m-n)}}{B2^n} = \frac{A}{B}2^{m-n}2^{f-(m-n)} \tag{3.24}$$

Order of operations matters, so actual evaluation would use the middle form, where the numerator is fully expanded before division. The far right form is just to show what the final result will look like.

Overflow is still a concern, so make sure that adjustment is done before the division to avoid rounding errors. Unfortunately, for example, a 16.16 divided by a 16.16 needs to be prescaled by 2^16, it is a guaranteed overflow. Again, modern CPUs can come to the rescue, this time by providing 64-bit / 32-bit division operations.

CHAPTER 4

REAL TIME IMPLEMENTATION

4.1 ADUC 7026 processor

The ADuC7026 are fully integrated, 1 MSPS, 12-bit data acquisition systems incorporating high performance multichannel Analog to Digital Converters (ADC), 16-bit/32-bit MCUs and Flash/EE memory on a single chip.

The ADC consists of up to 12 single-ended inputs. An additional four inputs are available but are multiplexed with the four Digital to Analog Converter (DAC) output pins. The four DAC outputs are only available on certain models e.g ADuC7026. However, in many cases where the DAC is not present, this pin can still be used as an additional ADC input, giving a maximum of 16 ADC input channels. The ADC can operate in single-ended or differential input modes. The ADC input voltage is 0 to VREF.

Low-drift bandgap reference, temperature sensor, and voltage comparator complete the ADC peripheral set. The ADuC7026 also integrate four buffered voltage output DACs on-chip. The DAC output range is programmable to one of three voltage ranges.

The devices operate from an on-chip oscillator and a PLL generating an internal high frequency clock of 45MHz. This clock is routed through a programmable clock divider from which the MCU core clock operating frequency is generated. The microcontroller core is an ARM7TDMI, 16-bit/32-bit RISC machine, which offers up to 45 MIPS peak performance. Eight kilobytes of SRAM and 62 kilobytes of nonvolatile Flash/EE memory are provided on-chip. The ARM7TDMI core views all memory and registers as a single linear array. On-chip factory firmware supports in-circuit serial download via the UART or I2C serial interface

ports, while no intrusive emulation is also supported via the JTAG interface. These features are incorporated into a low-cost QuickStart™ Development System supporting this MicroConverter® family.

The parts operate from 2.7 V to 3.6 V and are specified over an industrial temperature range of −40°C to +125°C. When operating at 45 MHz, the power dissipation is typically 120 mW. The ADuC7026 are available in a variety of memory models and packages.

4.1.2 Configuration&Features:

Analog I/O

- Multichannel, 12-bit, 1 MSPS ADC Up to 16 ADC channels
- Fully differential and single-ended modes
- 0 to VREF analog input range
- 12-bit voltage output DACs Up to 4 DAC outputs available
- On-chip voltage reference
- On-chip temperature sensor (±3°C)
- Voltage comparator

➢ Microcontroller

- ARM7TDMI core, 16-bit/32-bit RISC architecture
- JTAG port supports code download and debug

➢ Clocking options

- Trimmed on-chip oscillator (±3%)
- External watch crystal
- External clock source up to 44 MHz
- 45 MHz PLL with programmable divider

- Memory : 62 kB flash/EE memory, 8 kB SRAM
    - In-circuit download, JTAG-based debug
    - Software triggered in-circuit reprogram ability

- On-chip peripherals
    - UART, 2 × I2C® and SPI® serial I/O
    - Up to 40-pin GPIO port1
    - 4 × general-purpose timers
    - Wake-up and watchdog timers
    - Power supply monitor
    - Three-phase, 16-bit PWM generator1
    - Programmable logic array (PLA)
    - External memory interface, up to 512 kB
- Power
    - Specified for 3 V operation
    - Active mode: 11 mA @ 5 MHz; 40 mA @ 45 MHz
- Packages and temperature range
    - From 40-lead 6 × 6 mm LFCSP to 80-pin LQFP1
    - Fully specified for –40°C to +125°C operation
- Tools
    - Low-cost QuickStart™ development system
    - Full third-party support
- APPLICATIONS

- Industrial control and automation systems
- Smart sensors, precision instrumentation
- Base station systems, optical networking.

Functional block diagram of ADUC7026 micro controller is shown in Figure 4.1.

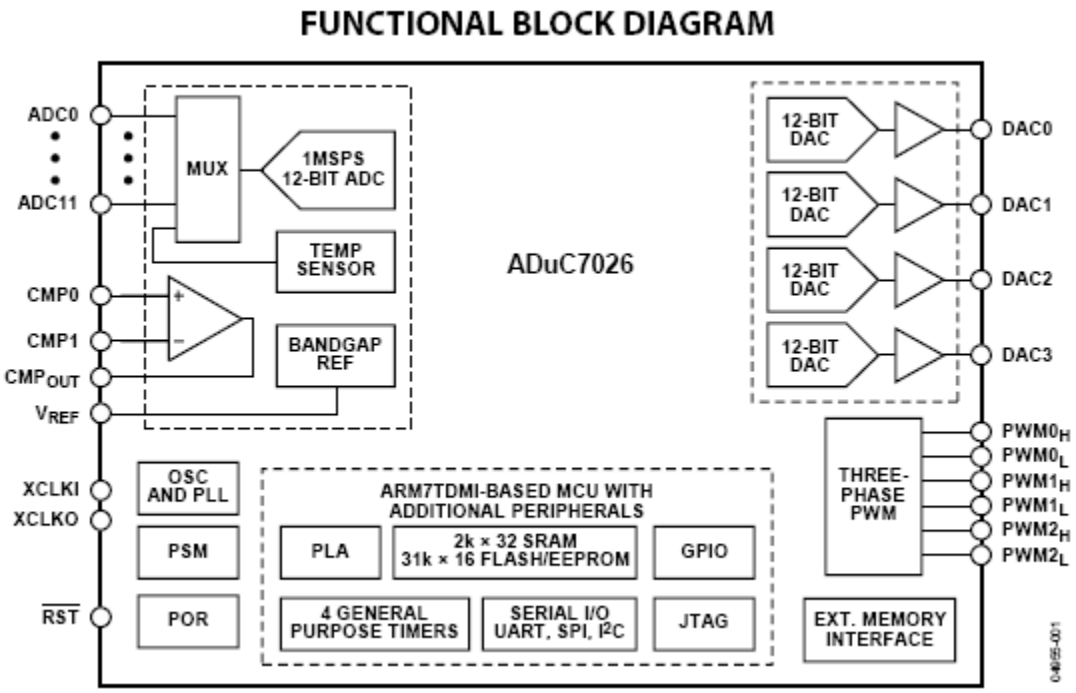


Figure 4.1 The functional Block diagram of ADUC 7026micro controller.

4.2 Real Time ANC Implementation on a Processor

As discussed in chapter 2, in the FXLMS,a system identification is first performed where the coefficients are stored to memory and later read back when the FXLMS structure operates and later with those results FXLMS adaptation is implemented.

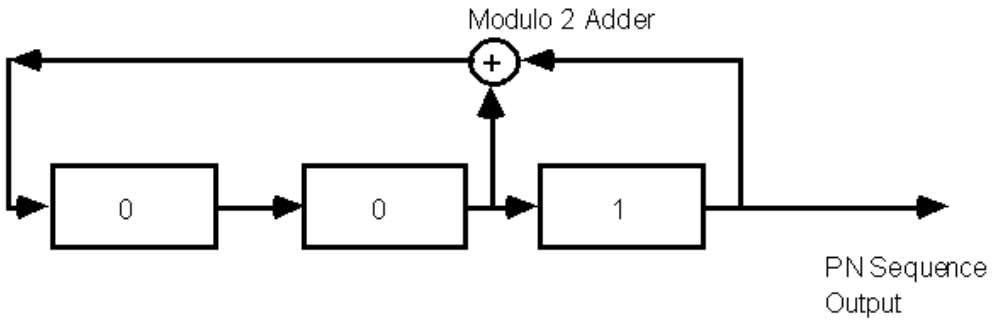
4.2.1 System identification

In system identification, the transfer function between error microphone and head phone transducer is measured. For this purpose, a pseudo noise generator program can be used. It generates random noise and plays it. Here in this system identification, there is no need of noise reference microphone. So, the microphone output is fed as input to the adaptive algorithm and hence the filter coefficients give the transfer function. A brief idea on pseudo noise generation is discussed.

4.2.2 Pseudo noise Generator

A Pseudo-random Noise (PN) sequence is a sequence of binary numbers, e.g. ±1, which appears to be random; but are in fact deterministic. The sequence appears to be random in the sense that the binary values and groups or runs of the same binary value occur in the sequence in the same proportion they would if the sequence were being generated based on a fair "coin tossing" experiment. In the experiment, each head could result in one binary value and a tail the other value. The PN sequence appears to have been generated from such an experiment. A software or hardware device designed to produce a PN sequence is called a PN generator.

Example: - A PN generator is typically made of N cascaded flip-flop circuits and a specially selected feedback arrangement as shown below.



The flip-flop circuits when used in this way are called a shift register since each clock pulse applied to the flip-flops causes the contents of each flip-flop to be shifted to the right. The feedback connections provide the input to the left-most flip-flop. With N binary stages, the largest number of different patterns the shift register can have is 2^N. However, the all-binary-zero state is not allowed because it would cause all remaining states of the shift register and its outputs to be binary zero. The all-binary-ones state does not cause a similar problem of repeated binary ones provided the number of flip-flops input to the module 2 adder is even. The period of the PN sequence is therefore 2^{N-1}, but IS-95 introduces an extra binary zero to achieve a period of 2^N, where N equals 15.

Starting with the register in state 001 as shown, the next 7 states are 100, 010, 101, 110, 111, 011, and then 001 again and the states continue to repeat. The output taken from the right-most flip-flop is 1001011 and then repeats. With the three stage shift register shown, the period is 2^3 -1 or 7.

The PN sequence in general has $2^N/2$ binary ones and $[2^N/2]$-1 binary zeros. As an example, note that the PN sequence 1001011 of period 2^3-1 contains 4 binary ones and 3 binary zeros. Furthermore, the numbers of times the binary ones and zeros repeat in groups or runs also appear in the same proportion they would if the PN sequence were actually generated by a coin tossing experiment.

The flip-flops which should be tapped-off and fed into the module 2 adder are determined by an advanced algebra which has identified certain binary polynomials called primitive irreducible or un-factorable polynomials. Such polynomials are used to specify the feedback taps. For example, IS-95 specifies the in-phase PN generator shall be built based on the characteristic polynomial

PI(x) = x15 + x13 + x9 + x8 + x7 + x5 + 1 (1) (4.2.1)

Now visualize a 15 stage shift register with the right-most stage numbered zero and the successive stages to the left numbered 1, 2, 3 etcetera., until the left-most stage is numbered 14. Then the exponents less than 15 in Eq. (1) tell that stages 0, 5, 7, 8, 9, and 13 should be tapped and summed in a module 2 adder. The output of the adder is then input to the left-most stage. The shift register PN sequence generator is illustrated in the figure 4.2.

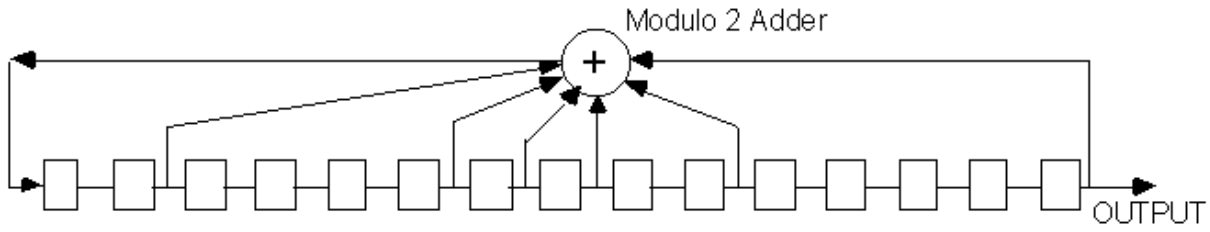


Figure 4.2 Shift register PN sequence generator

4.2.3 FXLMS adaptation

After the transfer function between the error microphone and the head phone transducer is estimated and it can be used in FXLMS adaptation. In FXLMS adaptation the noise reference microphone is used. An external low frequency noise is played and the noise microphone and the error microphone records it. Then the two responses are fed input to the adaptive filter such that an anti noise is produced in head phone transducer and cancels the noise.

4.2.4Experiment details

Before conducting the experiment the following details should be observed:

1) Amplitude levels should be checked before conducting the experiment real time. If the high amplitude level is more, it may lead to the breakage of DSP.
2) The volume of the amplifier of micro phone should not be changed while carrying out the experiment. If the volume is changed or the gap between the error microphone and head phone transducer is disturbed, the transfer function varies and system identification has to be repeated.
3) The project was done on evaluated version of ADUC 7026 micro controller. So some problems are experienced while loading the program into the flash memory of the chip. So press reset and serial port buttons such that the light of the reset button will be turned off. Then ¨the program can be loaded into the flash memory. Other wise it will lead to the JITAG failure.
4) Also, care must be exercised to not violate the sample cycle time as the clock speed is limited for 44.1 MHz.

CHAPTER 5

EVALUATION AND RESULTS

Evaluation in this project will give a clear idea on the suitable and maximum permissible filter lengths and step lengths on matlab fixed, floating point and on a C language. If the suitable filter lengths and step lengths are used, the better Signal to Noise Ratio (SNR) results can be obtained.

5.1 Evaluation

The thesis is done both in simulation and real time implementation. The simulation is done in matlab and DEV C++ and the real time implementation is done by using KEIL code composer interface. After getting the results and convergence of the algorithm, plots are made. These plots are compared with the variation of filter lengths and variation of step lengths and minimum possible filter lengths and step lengths are measured. Then it concludes the simulation part. After simulation, this algorithm is implemented on a real time processor.

Two real time signals are needed for this simulation part.

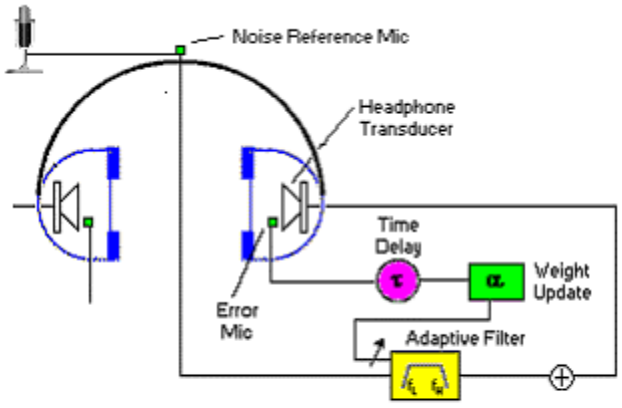


Figure 5.1 Adaptive noise cancellation head phones

- The transfer function between speaker to the micro phone is illustrated in Fig 5.1.

- The transfer function between error microphone and head phone transducer are computed by $$|H(w)| = \frac{P_{xy}(w)}{P_{xx}(w)} \quad 5.1$$

where the spectral power density of the output signal and the spectral power density of the input power signal are computed. By using the Eq 5.1, transfer function of a channel can be calculated.

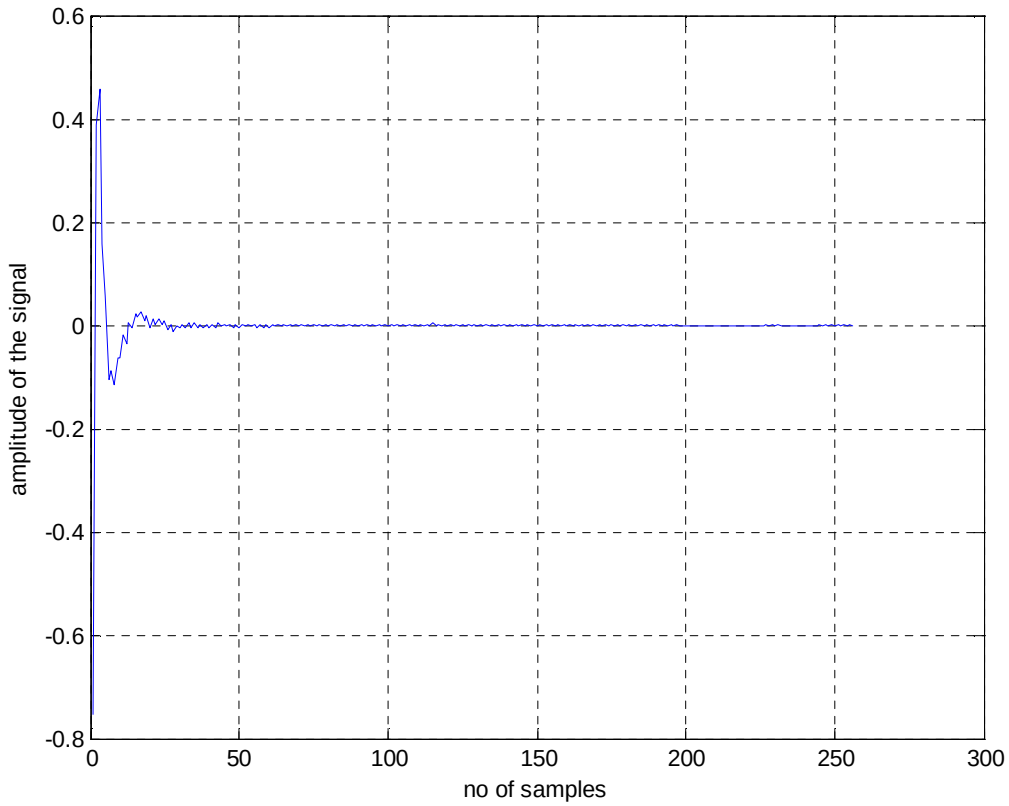


Figure 5.2 Measured response taken from reference microphone to error sensor

In Fig 5.3 a typical impulse response function between the head phone transducer and the error microphone is illustrated.

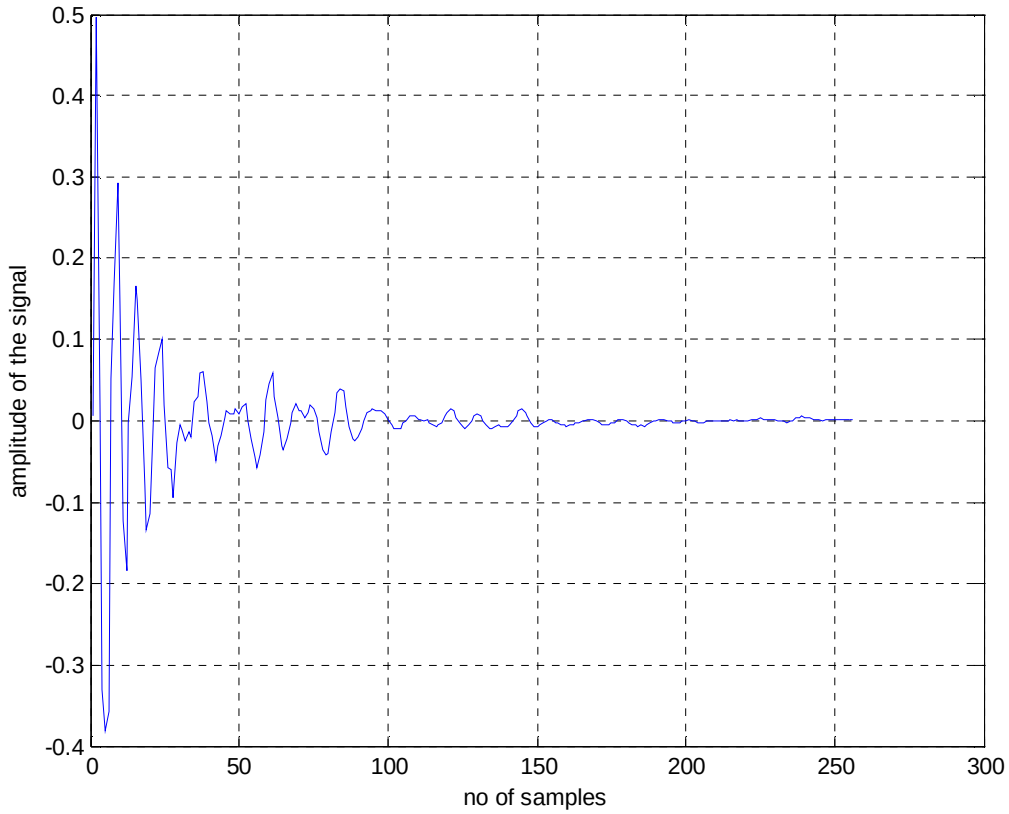


Figure 5.3 Measured response from head phone transducer to error microphone

5.2 Results

After getting these responses, H(w) and C(w) filters the random signal and taken as the signals d(n) and x(n). So the input signals are ready for the adaptive filtering. Now the signals are passed to the adaptive filtering sample by sample and LMS algorithm is implemented to it in matlab fixed point, matlab floating point and c fixed point.

After getting the results, the last 1000 samples are taken and Mean Square Error (MSE) is calculated for those thousand samples and the graphs are compared in Figure 5.4. This graph is plotted with the variation of steplength on X axis and error mean square on Y axis in C fixed point. The different colors show the

variation of different filter lengths. From the graph it is clear that the minimum step length can be used as 26. At steplength 26, it has minimum MSE value.

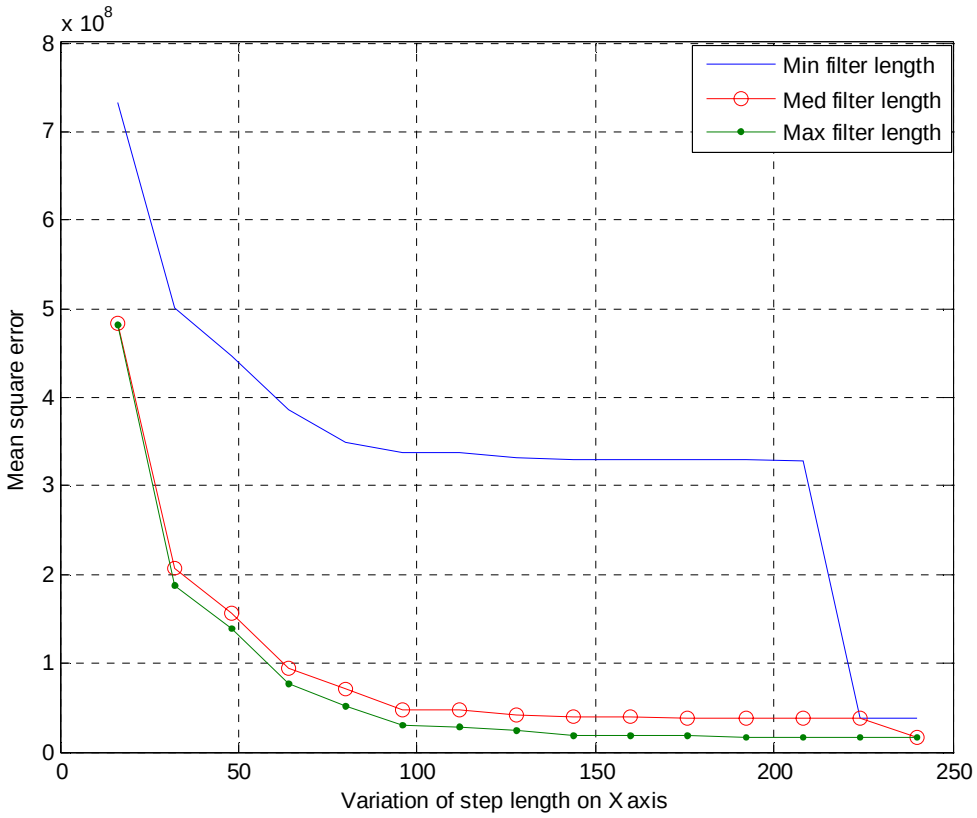


Figure 5.4 Mean square error for a C fixed point implementation as a function of various step length and different filter lengths: Minimum filter length (blue), medium filter length (red), maximum filter length (green)

The results are plotted now in matlab fixed point with the same dimensions ie. Variation of steplength on X axis and variation of emeansquare on Y axis. The different colours show the variation of filterlengths.

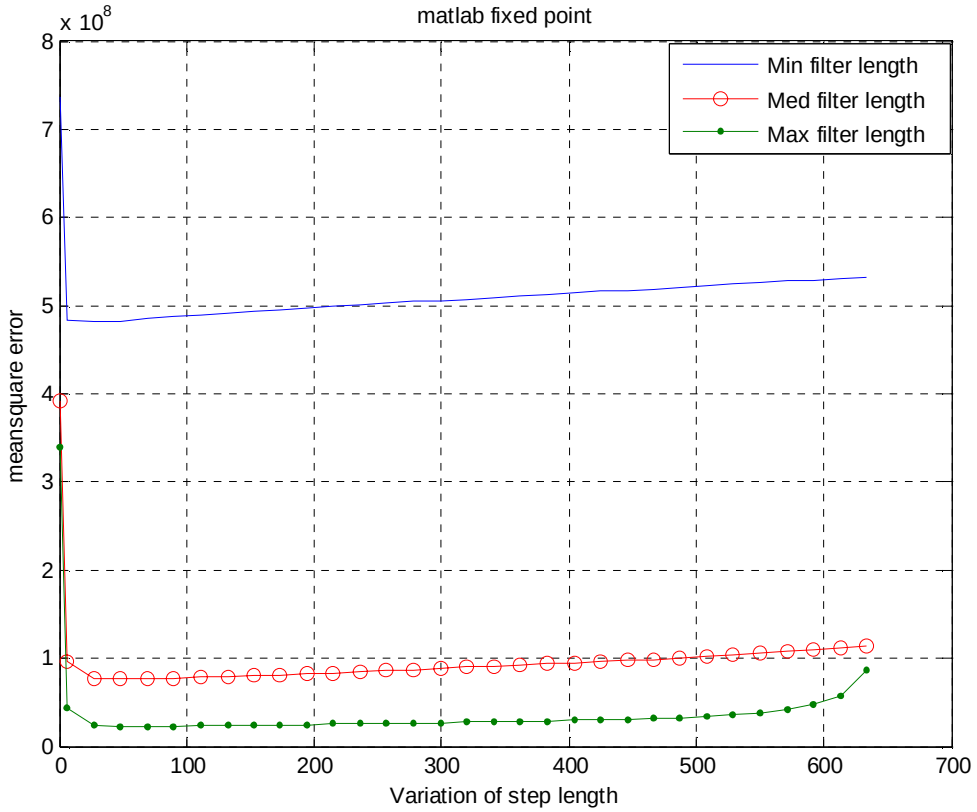


Figure 5.5 Mean square error for a Matlab fixed point implementation as a function of various step length and different filter lengths: Minimum filter length (blue), medium filter length (red), maximum filter length (green)

In this graph even, the min steplength is found to be 26 and the permissible steplengths for most cases is from 26 to 100.

From these cases, matlab fixed point. C fixed point, it can be noticed that the min step length is same for matlab fixed point and c fixed point subjected that the same input signal is used for both the cases.

Next coming to matlab floating point, it varies a bit. Matlab floating point with the same dimensions. ie. Variation of steplength on X axis and variation of emeansquare on Y axis. The different colours show the variation of filterlengths.

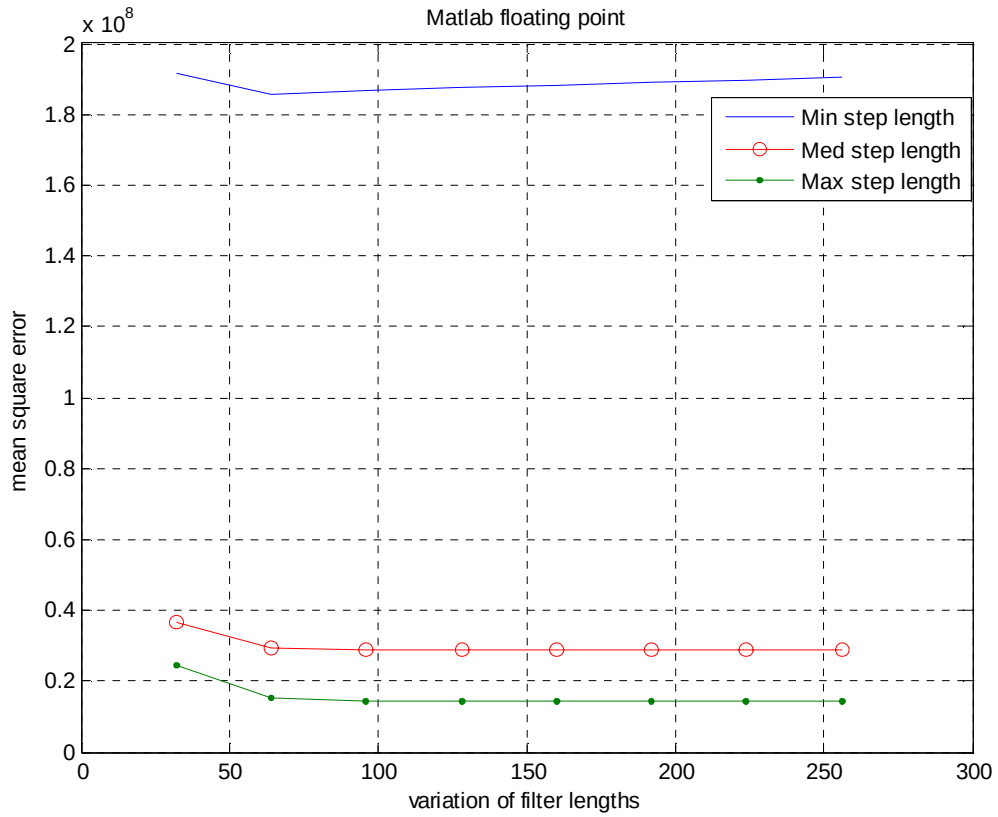


Figure 5.6 Mean square error for a Matlab floating point implementation as a function of various step length and different filter lengths: Minimum filter length (blue), medium filter length (red), maximum filter length (green)

Here the min steplength is found to be 60. This variation is because, in the fixed point it truncates the decimal point and where as in floating point it considers them. The permissible values are from 60 to 250. The different colors represent the different filter lengths. If the values in the MSE are noticed, it can be found maximum values in fixed point than in floating point.

Till now, the variation of steplengths and its effect on meansquare error are discussed. Now, variation of filterlengths and its effect on mean square error will be discussed. Now it is the turn to vary filter lengths and observe the plots for c fixed point as shown in figure 5.7. The minimum filter length is found to be 196 and the permissible range for filter lengths is found to be 96 to 200. So at filter length 196, it has min MSE value. As it is well known that as the filter length increases, the MSE value should decrease. But here in this case, it increases drastically. So those values have been truncated and zeros are kept instead. This is happening because simultaneously, step length values are increasing.

$$\mu = \frac{2}{(p+1).\left|R_{xx}(0)\right|} \tag{5.2}$$

So as the filter length and steplength values increases which are inversely proportional to each other, the resulting mean square error will increase.

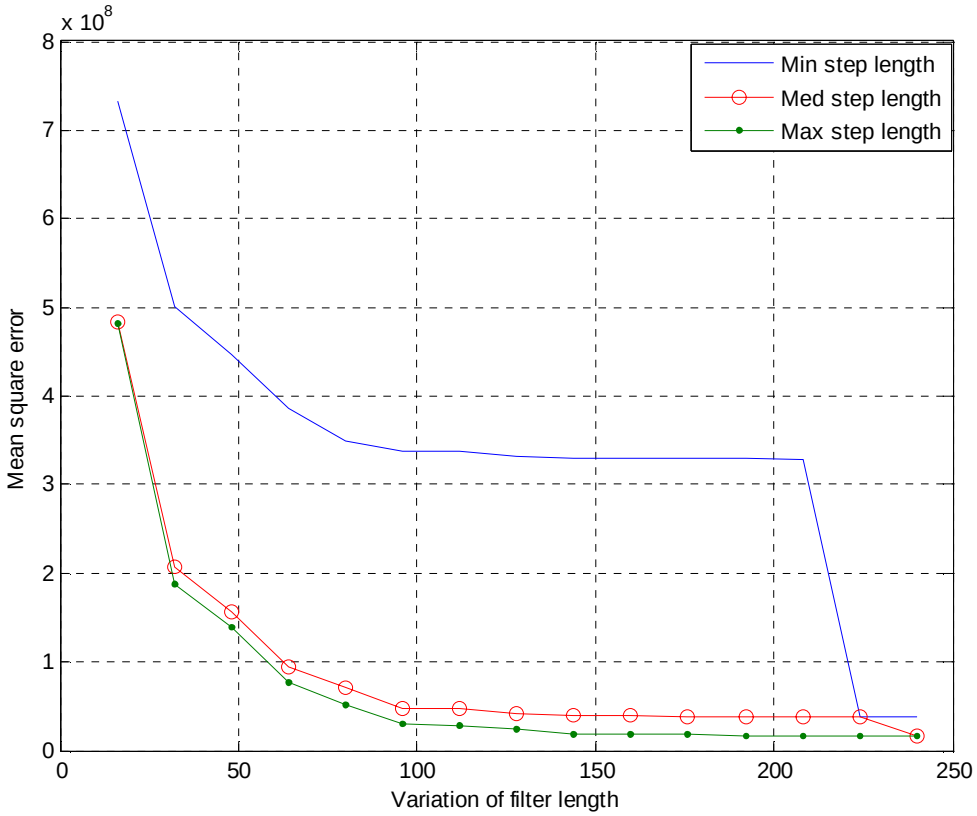


Figure 5.7 Mean square error for a C fixed point implementation as a function of various filter length and different step lengths: Minimum step length (blue), medium step length (red), maximum step length (green)

Plot for matlab fixed point is plotted with the same dimensions i.e variation of filterlengths on x axis, emeansquare on y axis and variation of step lengths on z axis. Here the color lines mean the variation of steplengths. The min emeansquare value is found at 196 filterlength and max permissible range is from 96 to 208.

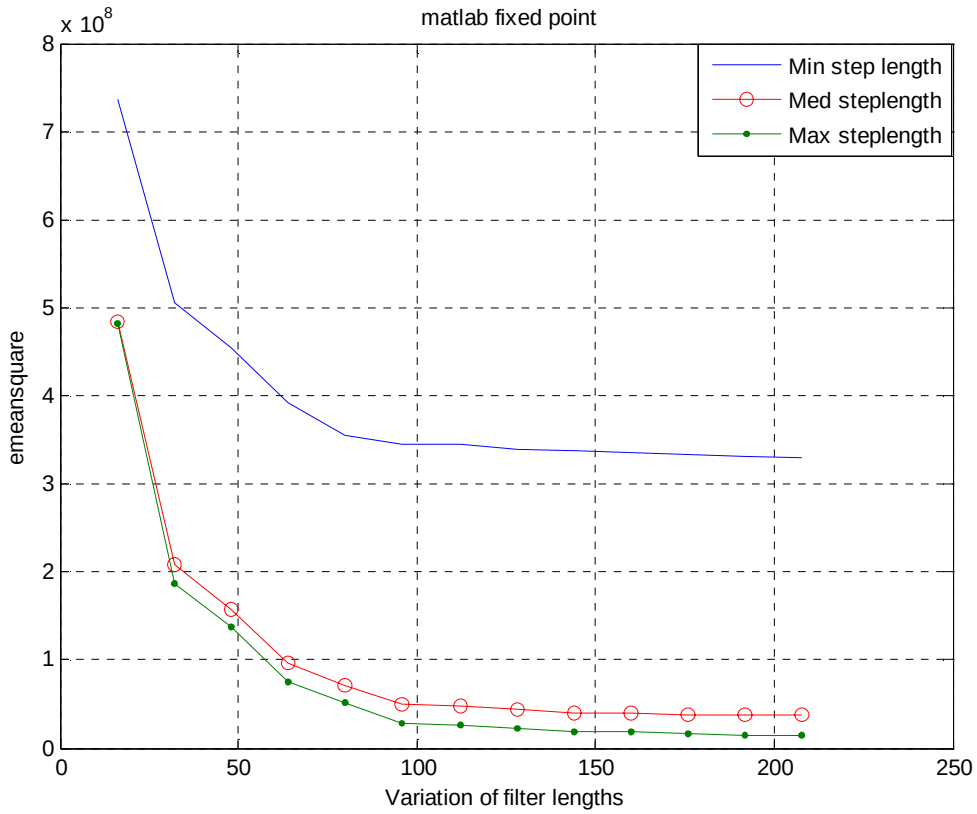


Figure 5.8 Mean square error for a Matlab fixed point implementation as a function of various filter length and different step lengths: Minimum step length (blue), medium step length (red), maximum step length (green)

Matlab and C language almost have the same numerical precision in fixed point. So it will have the same values for fixed point. These curves are meant to get an idea for which values the algorithm converges better.

In the formula

$$\mu = \frac{2}{(p+1).\left|R_{xx}(0)\right|} \tag{5.3}$$

μ Is steplength, P is filterlength

Rxx is autocorrelation matrix of the input signal.

Now, it is turn to have a look for Matlab floating point. The graph is plotted with the same dimensions as in the previous cases. Here, the MSE is less when compared to previous cases.

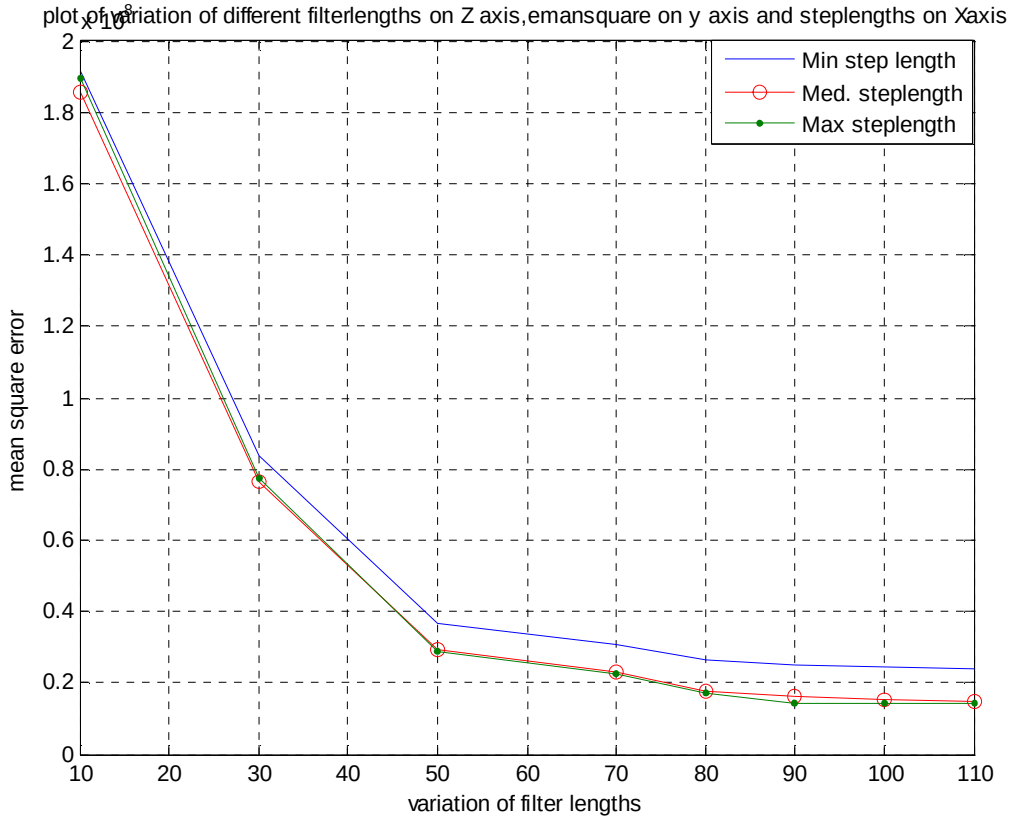


Figure 5.9 Mean square error for a Matlab floating point implementation as a function of various filter length and different step lengths: Minimum step length (blue), medium step length (red), maximum step length (green)

.

The figure 5.10 is the graph plotted taking variation of filter length on x axis, variation on steplengths on z axis and mean square error on y axis in c fixed point.

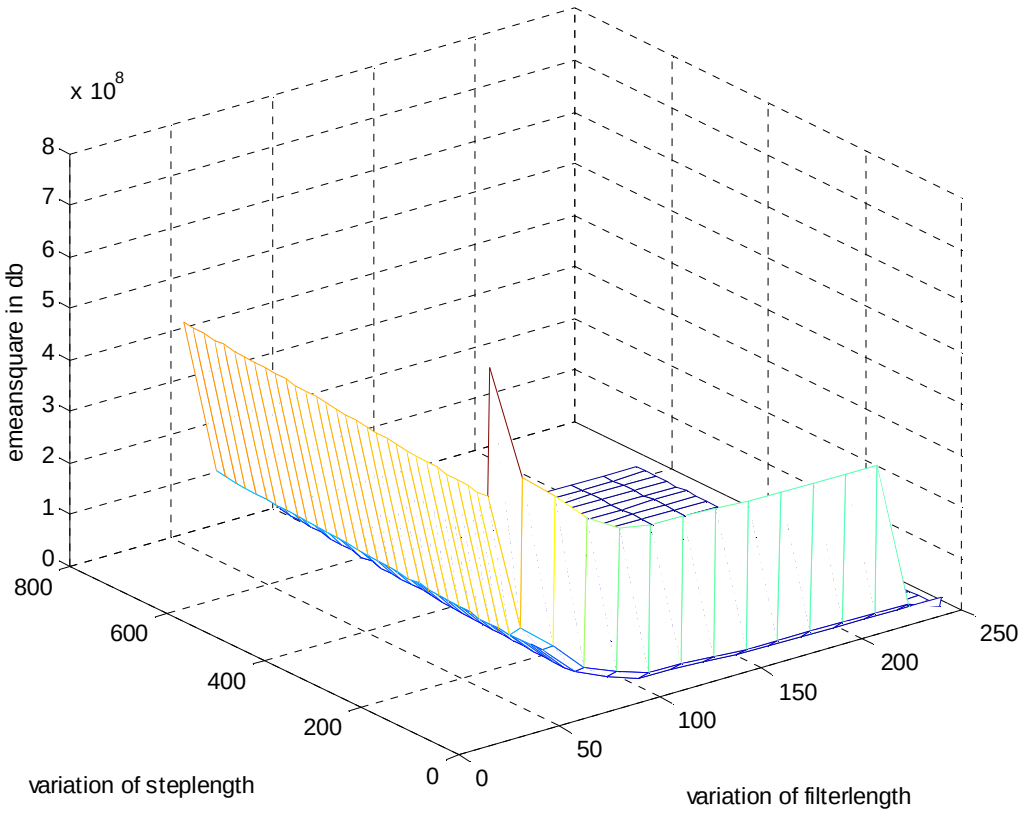


Figure 5.10 Mean square error for a Cfixed point implementation as a function of various filter length and different step lengths.

As the filter length increases, error mean square decreases. Hence it can be concluded that it is beneficial to have more filter taps to have a better attenuation of noise. But the filter length should be as small as possible to have a better convergence. So the selection of steplength is a tradeoff between fast convergence rate and high noise attenuation.

In Fig. 5.11 the graph is plotted taking variation of filter length on x axis, variation on steplengths on z axis and mean square error on y axis in matlab fixed point.

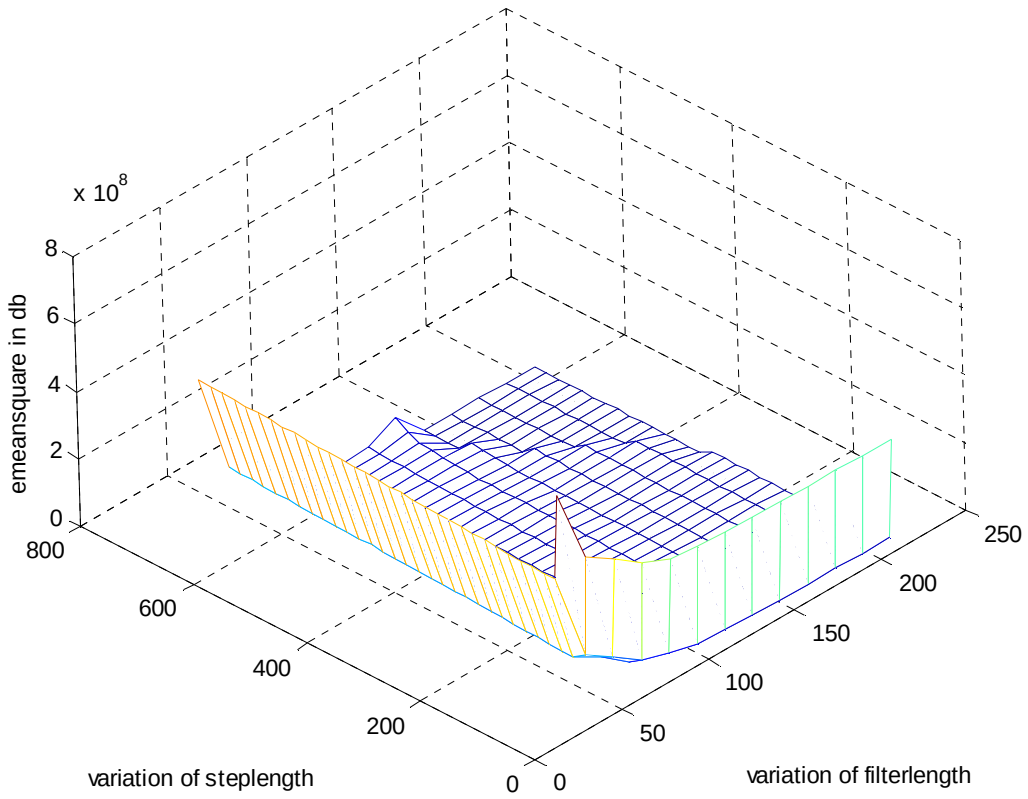


Figure 5.11 Mean square error for a Matlab fixed point implementation as a function of various filter length and different step lengths

Fig 5.12 is the graph plotted taking variation of filter length on x axis, variation on steplengths on z axis and meansquare error on y axis in matlab floating point.

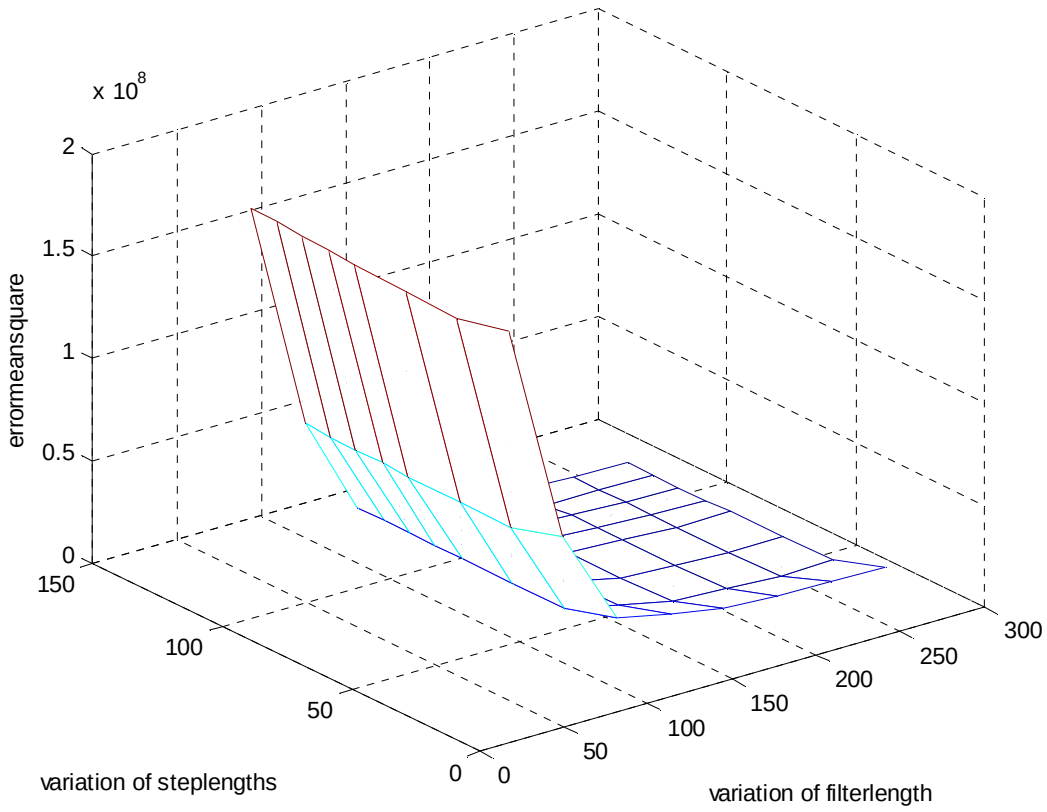


Figure 5.12 Mean square error for a Matlab floating point implementation as a function of various filter length and different step lengths.

Exponentially averaged error envelop is taken with 1 ms time constant and with suitable min. steplength and filterlength values from the learning curves for system identification. The exponentially averaged error envelop is presented in fig 5.13

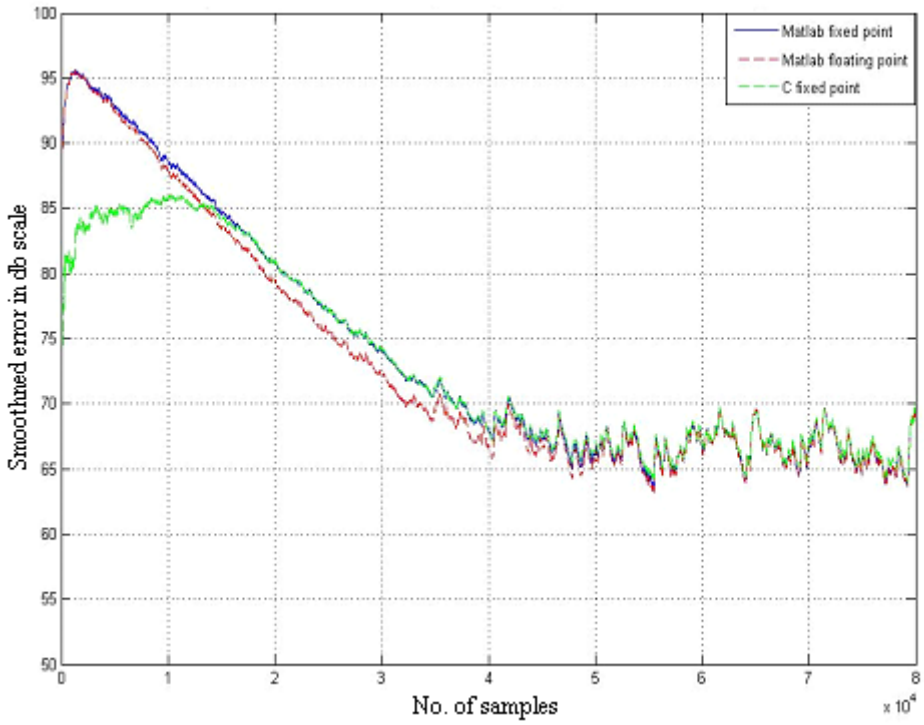


Figure 5.13 Comparision of averaged errorenvelop of floating and fixed point in system identification with min. filterlength of 32 taps for different implementations: Matlab fixed point (blue), maltab floating point (red), C fixed point (green)

From the Figure 5.13, it can be observed that

- The matlab floating and fixed points have a small variation. This is due to the fixing of the value in fixed point.
- The c fixed point varies with the other ones. This is because of the numerical precision and more over c has the integer number format values that is limited between -32,768 and +32,767 where as matlab has no such restrictions.

Now By using the C(w) channel from system identification, those values are substituted in fxlms adaptation part to get the final attenuation of noise. In fig 5.14 is the learning curve of FXLMS adaptation for c and matlab fixed and floating points.

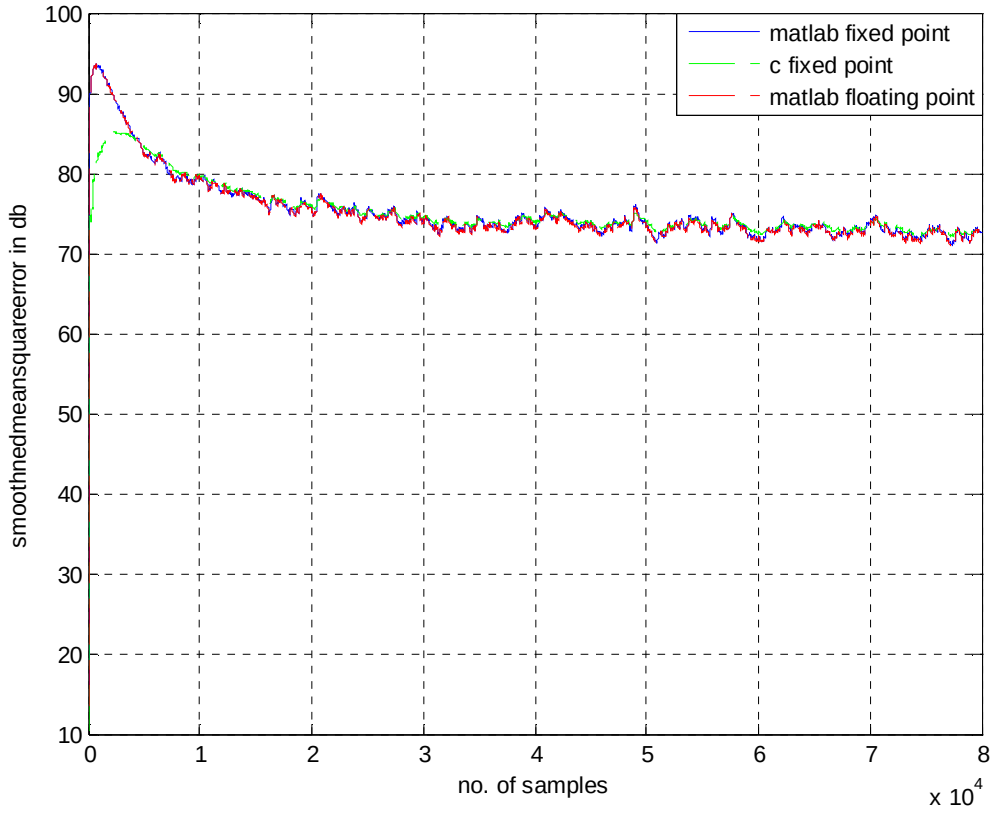


Figure 5.14 Comparision of smoothned error of floating and fixed point in FXLMS adaptation by taking min. filterlength and min. steplengthfor different implementations: Matlab fixed point (blue), maltab floating point (red), C fixed point (green).

The fallowing observations can be noticed from the figure 5.14

- The attenuation in the three lines differs because of the numerical precision and limitations as discussed earlier.
- In C fixed point, we have got the attenuation of noise for 13 db attenuation and for the rest, it is 22db attenuation.

If this algorithm is implemented on a real time processor, even more less attenuation can be recorded because the electronic component noise is added to it,which decreases SNR ratio further more.

CHAPTER 6

FURTHER RESEARCH

This thesis can be further modified by the following ways.

1) First, a full band approach is used in this report. But a sub band approach can also be implemented. In the sub band approach, the signal is divided into subbands by using a filter bank and each subband is processed by the adaptive filter and at last, they are again summed up. With this solution, better MSE with even less filter taps can be achieved. The subband approach is illustrated in fig 7.1

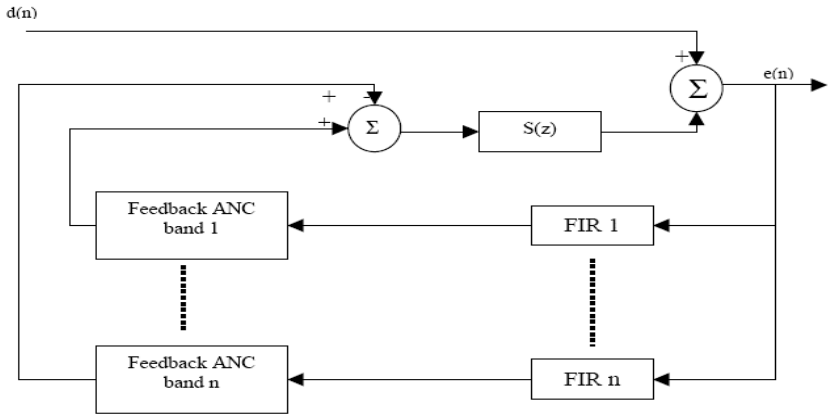


Fig 7.1 Schematic of a sub-band approach

In the figure 7.1, the signal d(n) is filtered into n signals with a filter bank and each filtered signal is processed by using feedback ANC and the output of the ANC is summed up resulting the error signal e(n) and the estimate of the out put signal y(n).

2) Till now, our approach is only a noise cancellation approach. It means only noise can be attenuated. It has one of the applications of the hearing protectors and active head sets. The noise cancellation with the introduction of sound can be used in pilot head phones. The presented implementation could be modified for that purpose.

.

CHAPTER 7

SUMMARY AND CONCLUSIONS

This thesis is successfully implemented and better knowledge is achieved on adaptive filtering, fixed point arithmetic's and getting better MSE in noise reduction. Finally almost 15 db attenuation of noise in c fixed point simulation was achieved. The MSE may be further decreased as the electronic component noise is added to the filtering part when it is implemented real time. Some modifications regarding sub-band approach are finally discussed.

REFERENCES

1. S M Kuo and D R Morgan, Active Noise Control Systems: algorithms and DSP implementations, *John Wiley and Sons, June 1999.*
2. S M Kuo and D.R.Morgan, Active Noise Control: a tutorial review, *Proceedings of the IEEE, Volume 87, Number 6,* June 1999.
3. The Filtered X LMS algorithm, an article by L.Håkansson, Blekinge institute of Technology, Sweden.
4. Statistical Digital Signal Processing and Modeling by Manson H Hayes,Wiley, 1996.
5. Adaptive Filter theory fourth edition by Simon Haykins, *Prentice-Hall,2001.*
6. Subband feed back active noise cancellation, thesis presented by Bharath siravara in University of Texas at Dallas.
7. Introduction to fixed point math, Article found on bookofhook.com.
8. B Widrow and S D Stearns, Adaptive Signal Processing, *Prentice-Hall, Inc. Englewood Cliffs, 1985.*
9. Least Mean Squares (LMS) by Bernard Widrow and Marcian Hoff
10. Active noise control. IEEE signal processing magazine, S.J. Elliott and P.A. Nelson, pages 12–35, October 1993.