"STUDY OF ADAPTIVE SIGNAL PROCESSING"

Submitted by:
Manas Ranjan patra
(109ei0334)

Under the guidance of
Prof. Upendra Kumar Sahoo

National Institute of Technology, Rourkela
Orissa-769008

April 2013

ACKNOWLEDGEMENT

I wish to thank Prof. ***Sukadev Meher*** *(HOD) NIT,Rourkela.I express my gratitude to my supervisor, prof.* ***Upendra ku. Sahoo*** *for his invaluable guidance .I am thankful to my faculty advisor Prof.* ***Ayas Kanta Swain*** *for helping me in clearing doubts & directing me towards the completion of my work. Last but not the least,I wish to thank prof.* ***Sarat kumar Patra*** *HOD at computer center, for his guidance & encouragement. I would also like to thank all professors& friends of NIT,Rourkela for their supports, given not just over the past four years, but always.*

Manas Ranjan Patra

CERTIFICATE

National Institute Of Technology Rourkela

This is to certify that the thesis entitled “STUDY OF ADAPTIVE SIGNAL PROCESSING” submitted by MANANAS RANAJAN PATRA **(***Roll. No. 109EI0334***)** in partial fulfillment of the requirements for the award of Bachelor of Technology in **Electronics & Instrumentation Engineering** during session2012-13 at ***National Institute of Technology, Rourkela***. A bonafide record of research work carried out by them under my supervision and guidance. The candidates have fulfilled all the prescribed requirements. The Thesis which is based on candidates’ own work, have not submitted elsewhere for a degree/diploma.

In my opinion, the thesis is of standard required for the award of a bachelor of technology degree in Electronics & Instrumentation Engineering.

Prof Upendra kumar Sahoo

Electronics & Communication

Department,NITRourkela

(HOD) Prof. Sukadev Meher
Electronics& Communication

Department,NITRourkela

ABSTRACT

An adaptive filter is a digital filter that can adjust its coefficients to give the best match t An adaptive filter is a digital filter that can adjust its coefficients to give the best match to a given desired signal. When an adaptive filter operates in a changeable environment the filter coefficients can adapt in response to changes in the applied input signals. Adaptive filters depend on recursive algorithms to update their coefficients and train them to near the optimum solution. An everyday example of adaptive filters is in the telephone system where, impedance mismatches causing echoes of a signal are a significant source of annoyance to the users of the system. The adaptive signal process is here to estimate and generate the echo path and compensate for it. To do this the echo path is viewed as an unknown system with some impulse response and the adaptive filter must mimic this response.

Adaptive Filters are generally implemented in the time domain which works well in most scenarios however in many applications the impulse response become long, and increasing the complexity of the filter beyond a level where it can no longer be implemented efficiently in the time domain. An example of acoustic echo cancellation applications is in hands free telephony system. However there exists an alternative solution and that is to implement the filters in the frequency domain. The Discrete Fourier Transform or Fast Fourier Transform (FFT) allows the conversion of signals from the time domain to the frequency domain in an efficient manner. Despite the efficiency of the FFT the overhead involved in converting the signals to the frequency domain does place a restriction on the use of the algorithm. When the impulse response of the unknown system and hence the impulse response of the filter is long enough however this is not an issue since the computational cost of the conversion is much less than that of the time domain algorithm. The actual filtering of the signals requires little computational cost in the frequency domain. Investigation of the so-called crossover point, the point where the frequency domain implementation becomes more efficient than the time domain implementation is important to establish the point where frequency domain implementation becomes practical

CONTENTS:

Chapter 1

1 Introduction:-

1. Project Specification

Adaptive *signal* are widely used in many Situations where the Characteristics of some *filter* or other are unknown. One application for these filters is quite an important part of today's telephone system, compensating for the echo problem, and the task of the adaptive filter is to mimic this impulse response. Implementation Adaptive filter is common in the time domain, however it for an unknown system with a very long impulse response to it becomes more efficient in Place the filter in the frequency domain. The objective of this project is to investigate the Design and Implementation of adaptive filtering algorithms in the frequency domain, with emphasis on their use in "system identification" problems. Simulate the adaptive filtering algorithms in the time and frequency domains and compare them from the point of view of performance and Implementation complexity (in particular, to determine the "crossover point" where frequency-domain Implementation becomes more efficient). A suitable laboratory bench test a real circuit Involving should be constructed in order to Demonstrate the functionality in near-real-time. With the completion of the project we have a fully operational should adaptive filter based in the frequency- domain. This adaptive signal process will be applicable where the length of the unknown system's impulse response is long enough for practical Implementation of frequency domain adaptive filtering. Time domain adaptive filtering entirely Will not be Replaced as it is. A possible application for the adaptive filter in the frequency domain acoustic Arises in Echo cancellation for hands free telephony.

1.2 Echo Cancellation in long distance telephony

Echoes in the telephone system Occur When a speech signal encounters an impedance mismatch in the telephone circuit. A portion of the signal is reflected when it meets any impedance mismatch as illustrated in Figure 1.1. If the delay between the speech and the returning cast is long then the cast can be a significant source of annoyance to the users of the telephone system. The most Effective solution to this problem has been cast that of adaptive cancellation, Figure 1.1 shows the layout of such a system [1].

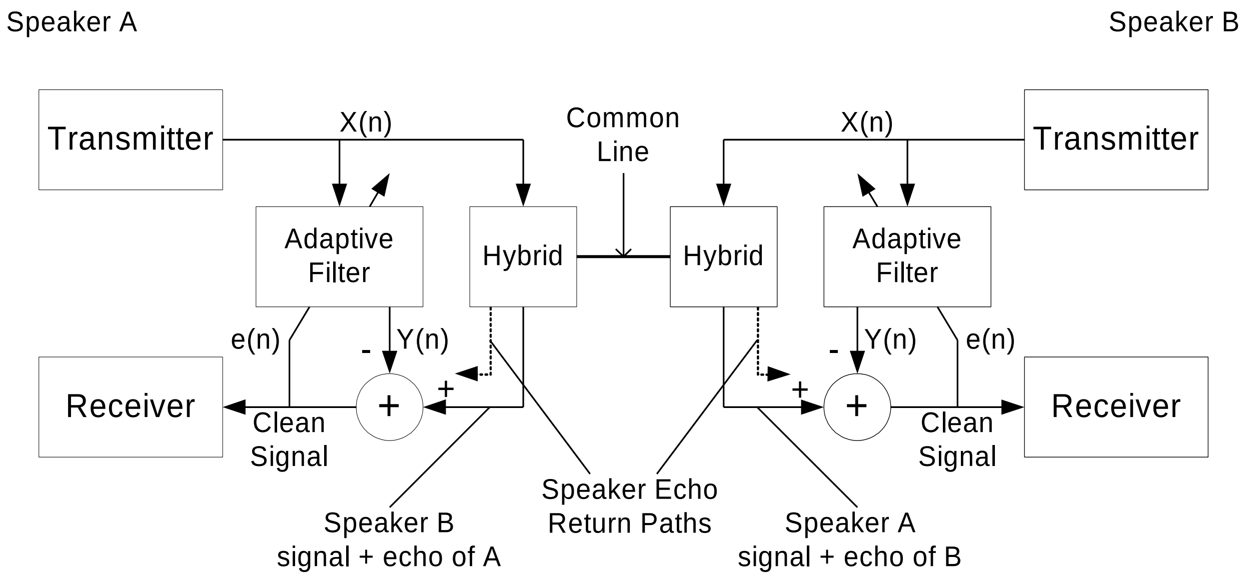

Figure 1.1 Echo Cancellations

The hybrid is basically a bridge circuit with three ports and it is here that impedance mismatches can occur if the bridge is not perfectly balanced [1]. The adaptive filter is in essence the echo canceller. This filter operates in system identification mode (see Section 3.2) with the echo path representing the unknown system the filter is to identify. Notice the inputs to the adaptive filter, speech from the transmitter constitutes the desired response and Equation 1.0 represents the error signal.

$$e(n) = s(n) - y(\mathrm{n}) \qquad \mathbf{1}$$

where ***y(n)*** is the output of the adaptive filter and ***s(n)*** is composed of the other speaker’s speech and the echo of the speaker's own speech. For satisfactory operation of the echo canceller, the length of the filter's impulse response must be greater than the length of the longest echo path.

Chapter 2
Filtering

Having introduced the project in the previous chapter it is time to introduce digital filtering. Adaptive filters are mainly implemented with digital filters, digital filtering therefore has a significant role to play in this project.

2.1 Digital Filters

2.1.1 Analogue Filters

Analogue filters are systems or networks that alter a given signal in a specified manner. The wave-shape, amplitude-frequency and the phase-frequency are three components of any given signal that can be altered.

2.1.2 Discrete Time Signals

It is now appropriate to explain what we mean by a discrete time (digital) signal. Generally signals that occur naturally are continuous, i.e. they are defined for all points in time. A discrete time signal however is only defined at specific points in time, for all other points in time it is zero. Discrete time signals may in fact be described as sampled versions of analogue signals. The sampling rate of the discrete time signal must be chosen high enough to adequately represent the information contained in the sampled signal. In order to ensure that this is the case we use at least the Nyquist rate, which is twice the frequency of the highest frequency component of the given signal.

$$\textit{Nyquist Rate} = 2\, f_{max} \qquad \textbf{2}$$

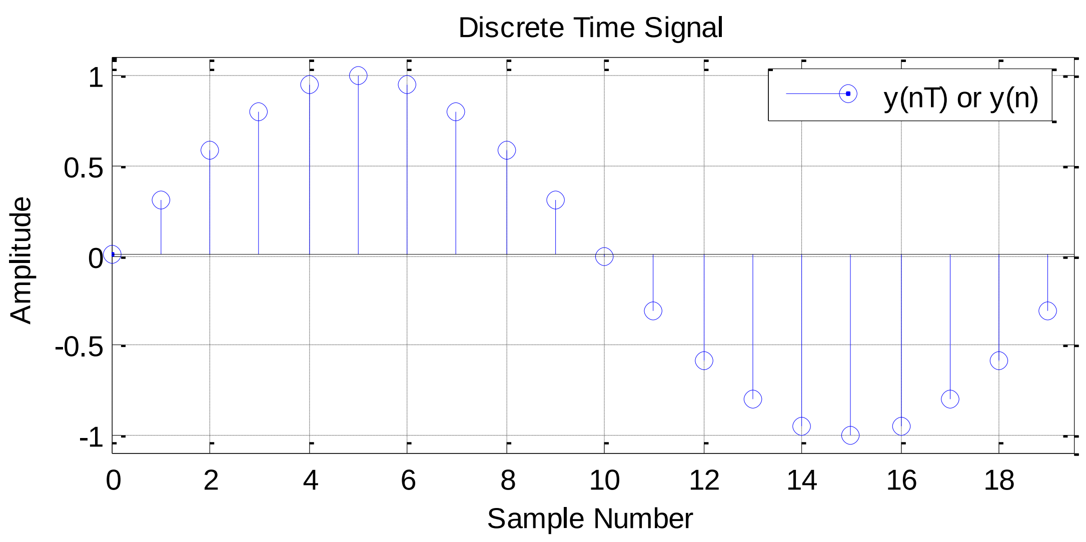

Figure 2.1 A Discrete Time Signal obtained by sampling an analogue sine wave

2.1.3 Introduction to Digital Filters

The system or network in the case of digital filters are mathematical algorithms, these algorithms operate on digital signals and attempt to modify these signals in the same way analogue filters modify analogue signals. Equation 2.1 defines the operation of linear digital filters [2].

$$y(n) = \sum_{k=0}^{N-1} h(k) - x(n-k) \qquad \textbf{2.1}$$

where ***h****(k), k = 0,1…N-1* are the filter coefficients, ***x(n)*** is the filter input and ***y(n)*** is the filter output. We note at this point that the above Equation represents the convolution of the input signal ***x(n)*** with the filter's impulse response ***h****(k)* to give the filter output ***y(n)***. There will be further discussion of convolution in Section 2.2.

2.1.4 Advantages of Digital Filtering

- Automatic updating of frequency Response if a programmable processor is used, which means that adaptive filtering can be implemented more easily.
- The filter output and input can be stored for further use.
- Some characteristics are not possible with analogue filters.
- Can easily take advantage of advances in VLSI technology.
- Performance is not affected by temperature variations or by slight differences in components making it possible to repeat the characteristics from one filter to the next.
- More than one signal can be filtered at a time.
- Precision is only limited by word length.
- It is possible to use digital filters at lower frequencies, which are often found in biomedical applications. [2]

2.1.5 Disadvantages of Digital Filtering

- The speed at which the filter operates may be limited by the speed of the processor or by the number of arithmetic operations involved. This number increases as the specifications of the filter are tightened.
- Digital filters are subject to round-off noise encountered in computation and if the input signal was sampled to Analogue to Digital Conversion noise.
- The design of digital filters is a far more complex task than designing an analogue filter. Computer aided design techniques in the right hands however do help to overcome this problem. Also once designed the system is usable with little or no modification for other different digital signal processing tasks. [2]

2.2 Convolution

In this sub-section, we will define the above term for the purposes of this project. Let us first define Correlation, in essence the Cross Correlation of two waveforms is a method for comparing the waveforms. Consider two waveforms, both sampled at the same rate. The sum of the products of the corresponding pairs of points is represented as a measure of the correlation of the two waveforms. Convolution can now be introduced as the cross correlation of two signals with one of them reversed [4]. Equation 2.2 defines the process for continuous time signals.

$$y(t) = x(t) * h(t) \qquad \textbf{2.2}$$

where * denotes convolution, when ***x(t)*** *and* ***h(t)*** are two finite digital signals this Equation can be represented as

$$y(n) = \sum_{k=0}^{N-i} h(k) - x(n-k) \qquad \textbf{2.3}$$

Notice that this is the same as Equation 2.1, which describes the operation of digital filters. In this case one of the sequences to be convolved is the impulse Response of the digital filter. Therefore it can be recognized that digital filtering is an application of convolution. Convolution can be viewed as a description of how the input to a given system reacts with the system's impulse response to produce an output. Convolution in the time domain is very computationally intensive and the fact that the convolution of two sequences can be calculated more efficiently if the signals are transformed from the time domain to the frequency domain is exploited in following chapters in an attempt to reduce the computation necessary for digital filters.

2.3 FIR and IIR Filters

There are two main types of digital filters, finite impulse response (FIR) and infinite impulse response (IIR) filters. The filter output is calculated in a similar manner for both types. Equation 2.1 gives the equation for an FIR filter, the equation for an IIR filter is of the same form only that the sum is taken to infinity rather than to N-1 [2]. From this it is clear that for IIR filters the impulse response is of infinite duration whereas for FIR filters it is only of length N.

There are a number of advantages specific to the different filters mentioned above [2].

1. FIR filters provide a linear phase response, this means that every frequency component of the applied signal will be delayed by the same amount, in other words there will be no phase distortion introduced by the filter.
2. Word length has less impact on FIR filters
3. If the filter specifications are tight, demanding a sharp cut-off frequency for example, IIR

filters require fewer coefficients. It will be shown later however that the computational efficiency of the FFT will significantly improve the implementation of FIR filters.

4. The stability of IIR filters can often not be guaranteed whereas FIR filters realized according to Equation 2.1 will always be stable.
5. Analog Filters can be readily transformed to equivalent IIR filters. This is not the case with FIR filters, however it is easier to synthesize filters of arbitrary frequency response with FIR filters.

In general FIR filters are desirable and would be used where possible, however when filter specifications are very high IIR filters may be best suited to meet them. For this project we will deal only with FIR filters.

2.4 Recursive and Non Recursive Filters

Non-recursive filters are filters where the output depends only on current and previous input samples as depicted in Figure 2.3. An important property of non-recursive filters is they will always be stable. FIR filters are in general non-recursive, which in turn means that they are also always stable [2].

A recursive filter on the other hand is a filter whose output samples may depend on previous output samples as well as the current and previous input samples.

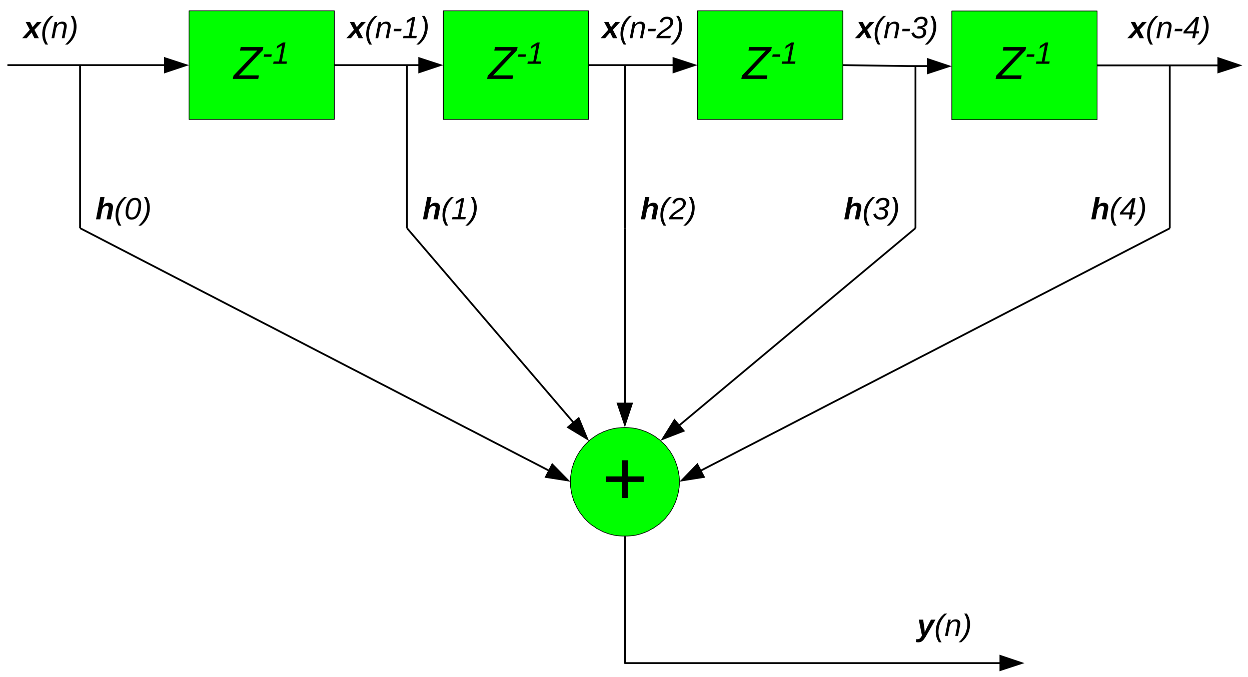

Figure 2.3 A Non-Recursive filter structure

Having introduced the concept of digital filters and the fact that the basis for digital filtering is time domain convolution, in particular stable FIR filters we are now ready to move to the next chapter where we shall discuss Adaptive Filtering.

Chapter 3 Time Domain Adaptive Filtering

3.1 Introduction

An adaptive filter is no more than a digital filter, which can adjust its characteristics. It adapts to changes in its input signals automatically according to a given algorithm. The algorithm will vary the coefficients according to a given criteria, typically an error signal to improve it's performance.

In essence an adaptive filter is a digital filter combined with an adaptive algorithm, which is used to modify the coefficients of the filter.

Adaptive filters are used in many diverse applications in today's world for example telephone echo canceling, radar signal processing, equalization of communication channels and biomedical signal enhancement.

Adaptive filters are useful [2]

- when the characteristics of a given filter must be variable
- when the spectrum of a given signal overlaps with the noise spectrum
- if the frequency band occupied by noise is unknown or may vary with time.

In most real world scenarios adaptive filters are realized using Finite Impulse Response (FIR) filters, since they are guaranteed to be stable and are simple to use.

3.2 Adaptive Filters in System Identification mode

Since there are many applications for adaptive filters such as those just described, we often alter the structure of the filter used to suit the application. In this project we are interested in the use of adaptive filters for the purposes of system identification in which case a block diagram of the filter will be of the form shown in Figure 3.1 [2]. Notice that the error signal in this case is the difference between the filter output and the desired response, the output of the unknown system.

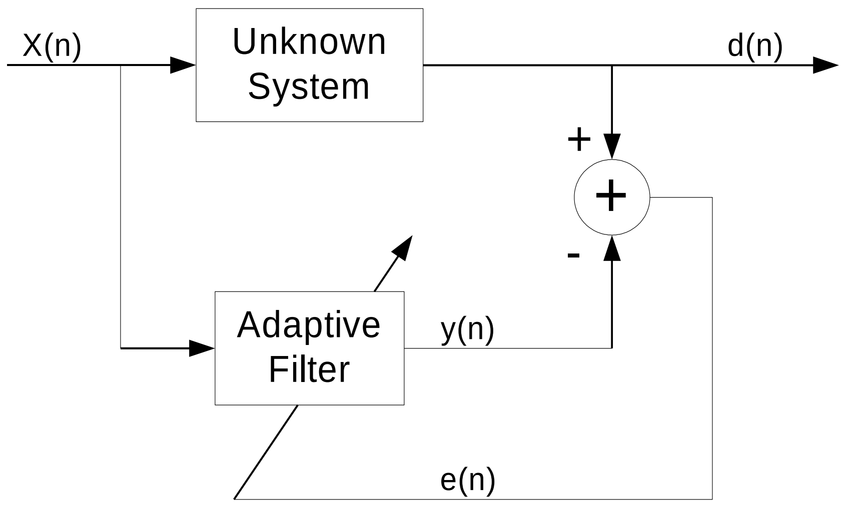

Figure 3.1 An adaptive filter in system identification mode

3.3 Time Domain Adaptive Filtering Algorithms

There are many algorithms used to adjust the coefficients of the digital filter in order to match the desired response as well as possible. The LMS Algorithm is the more successful of the algorithms because it is the most efficient in terms of both storage requirement and indeed computational complexity [2]. Similar to the steepest descent algorithm on which the LMS algorithm is based upon, the basic LMS algorithm updates the filter coefficients after every sample.

3.3.1 The LMS Algorithm explained

The Least-Mean-Square algorithm in words [1]:

$$\boxed{\begin{array}{c}\text{Updated Value}\\ \text{of tap-weight}\\ \text{vector}\end{array}} = \boxed{\begin{array}{c}\text{Old Value of}\\ \text{tap-weight}\\ \text{vector}\end{array}} + \left[\boxed{\begin{array}{c}\text{Learning}\\ \text{rate}\\ \text{parameter}\end{array}} \text{X} \boxed{\begin{array}{c}\text{Tap-input}\\ \text{vector}\end{array}} \text{X} \boxed{\begin{array}{c}\text{Error}\\ \text{Signal}\end{array}}\right]$$

Figure 3.2 The LMS Algorithm in words

The simplicity of the LMS algorithm and ease of implementation means that it is the best choice for many real-time systems [2].

The implementation steps for the LMS algorithm

1. Define the desired response. Set each coefficient weight to zero.

$$h(i)=0, i=1,2,3,\ldots,N, \qquad \textbf{3.0}$$

For each sampling instant (k) carry out steps (2) to (4):

2. Move all the samples in the input array one position to the right, now load the current data sample k into the first position in the array. Calculate the output of the adaptive filter by multiplying each element in the array of filter coefficients by the corresponding element in the input array and all the results are summed to give the output corresponding to that data that was earlier loaded into the input array.

$$y(k)=\sum_{i=0}^{N-1} h(i)\,x(i) \qquad \textbf{3.1}$$

3. Before the filter coefficients can be updated the error must be calculated, simply find the difference between the desired response and the output of the adaptive filter.

$$e(k)=y(k)-d(k) \qquad \textbf{3.2}$$

4. To update the filter coefficients multiply the error by μ, the learning rate parameter and then multiply the result by the filter input and add this result to the values of the previous filter coefficients.

$$h(k+1)=h(k)+2\mu\,e(k)\,x(k) \qquad \textbf{3.3}$$

There are also other LMS based algorithms, which include the complex LMS, the block LMS algorithm and the Time sequenced LMS algorithm [2].

3.4 The importance of *μ* and *N*

A very important part of the algorithm is the updating of the filter coefficients as would be typical for all adaptive filter algorithms. μ is critical for the update and must be chosen accurately to ensure the filter converges. Updating the filter coefficients is important because this is the part of the code that governs how well the filter will converge to the desired response. Another element that has a key role in this convergence is the number of filter coefficients N. Intuitively the number of coefficients must at least equal the length of the impulse response of the unknown system. The effects of varying both the value of μ and the number of filter coefficient, N is demonstrated in the project by varying these numbers and indicating the resulting effects by means of plotting error signals. The system that we are trying to identify for testing purposes is a modem echo path, which was provided by the project supervisor and is shown in Figure 3.3. Figures 3.4 and 3.5 plot the error obtained against different values of μ and N respectively. The error signal in both cases consists of the mean square error calculated using the last fifty samples of the actual response obtained for each value of μ and N respectively. The error is defined as the difference between the actual and the desired response of the adaptive filter. In both cases the programs are run for 1000 samples for each value. In the case of μ it is true to say that better convergence can be expected if the program was to run for more samples. However, it is felt that the representation in Figure 3.4 of the comparison of the different values of μ offers the best view to realize the most suitable value of μ resulting in fast convergence and low final error.

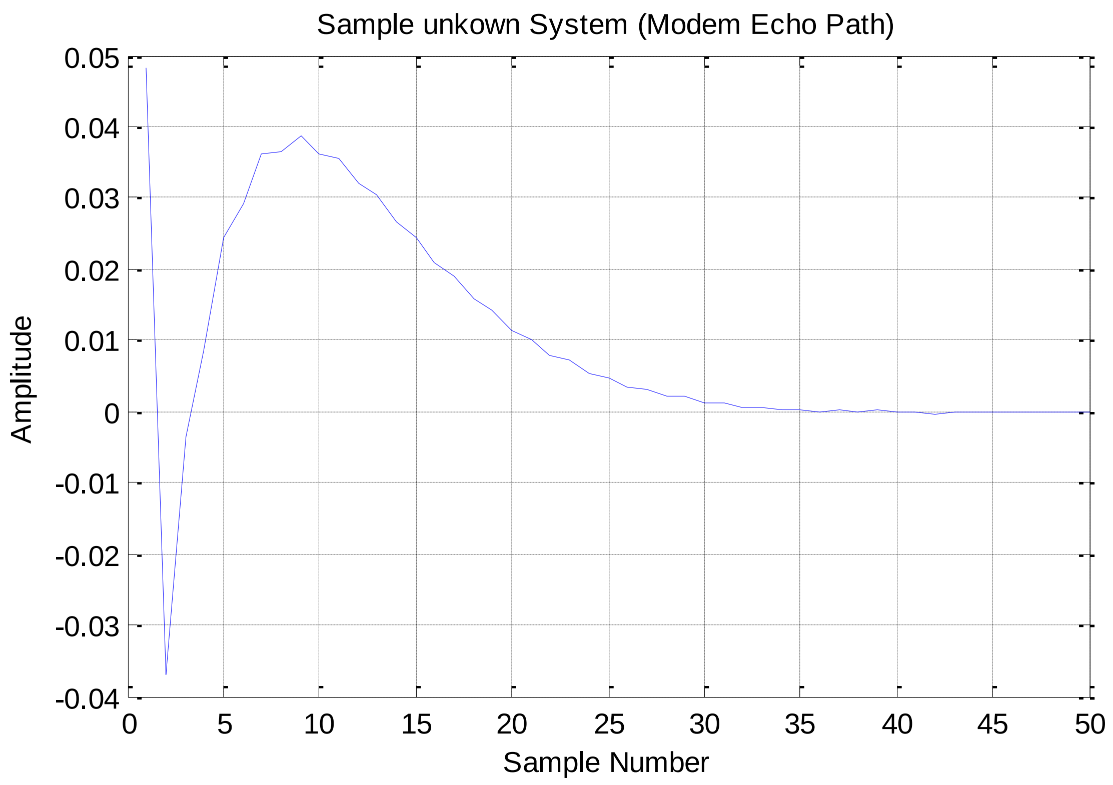

Figure 3.3 The sample unknown system used represents a modem echo path

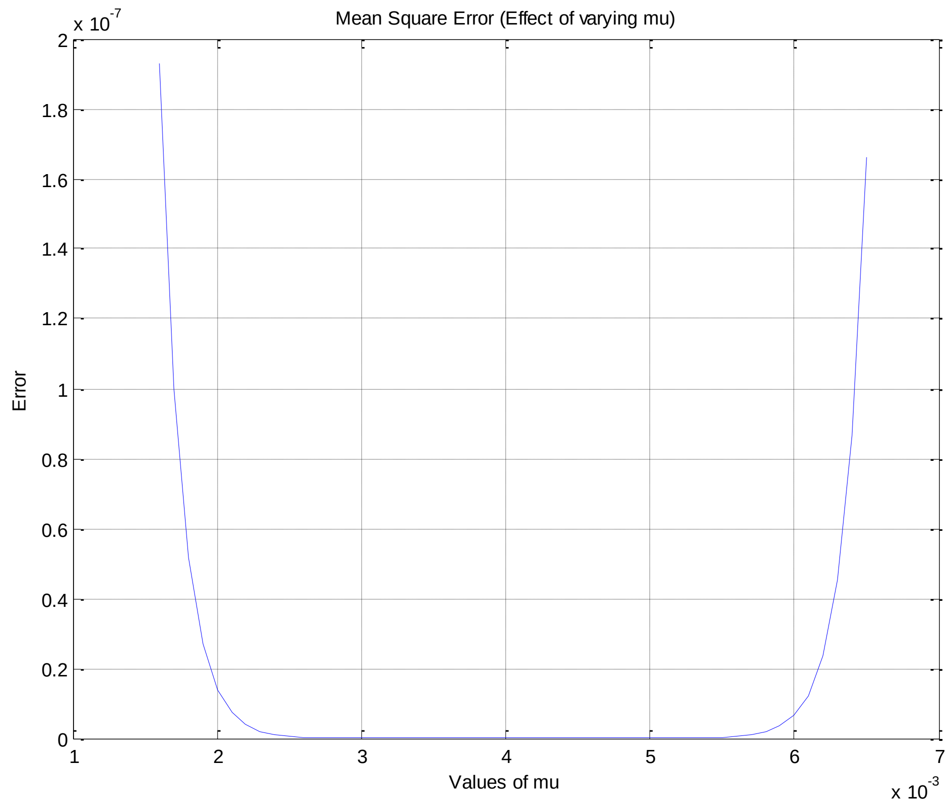

Figure 3.4 The effect of varying mu.

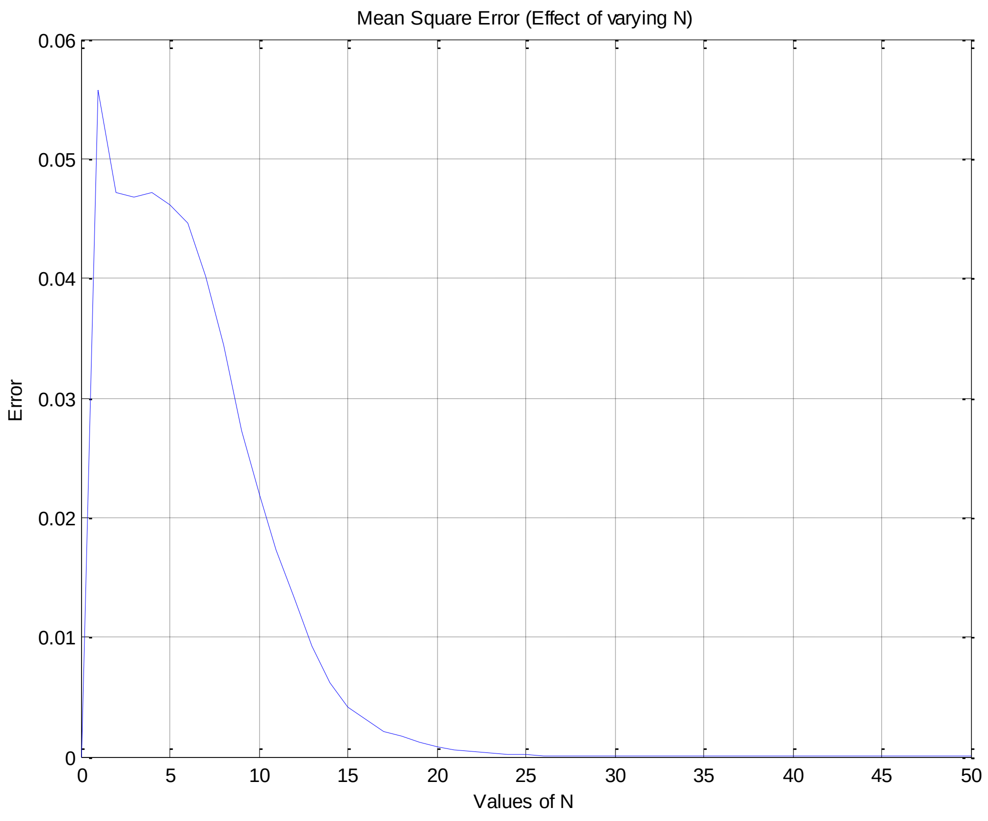

Figure 3.5 The effect of varying the number of filter coefficients

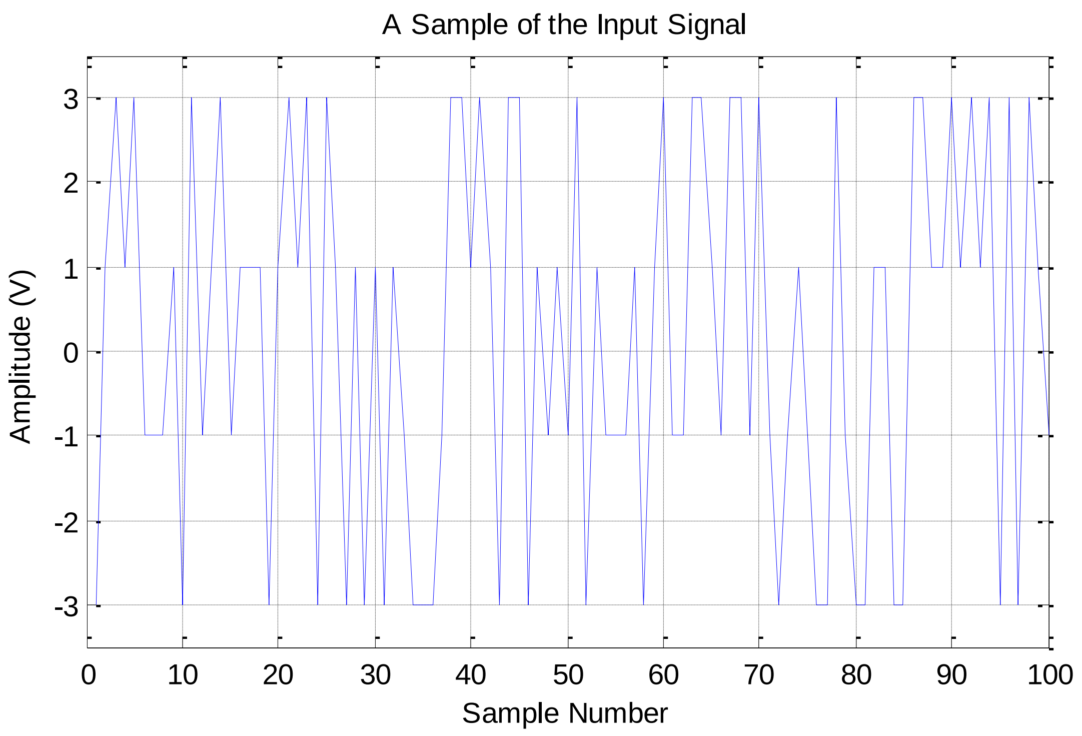

Figure 3.6 A sample of the input signal

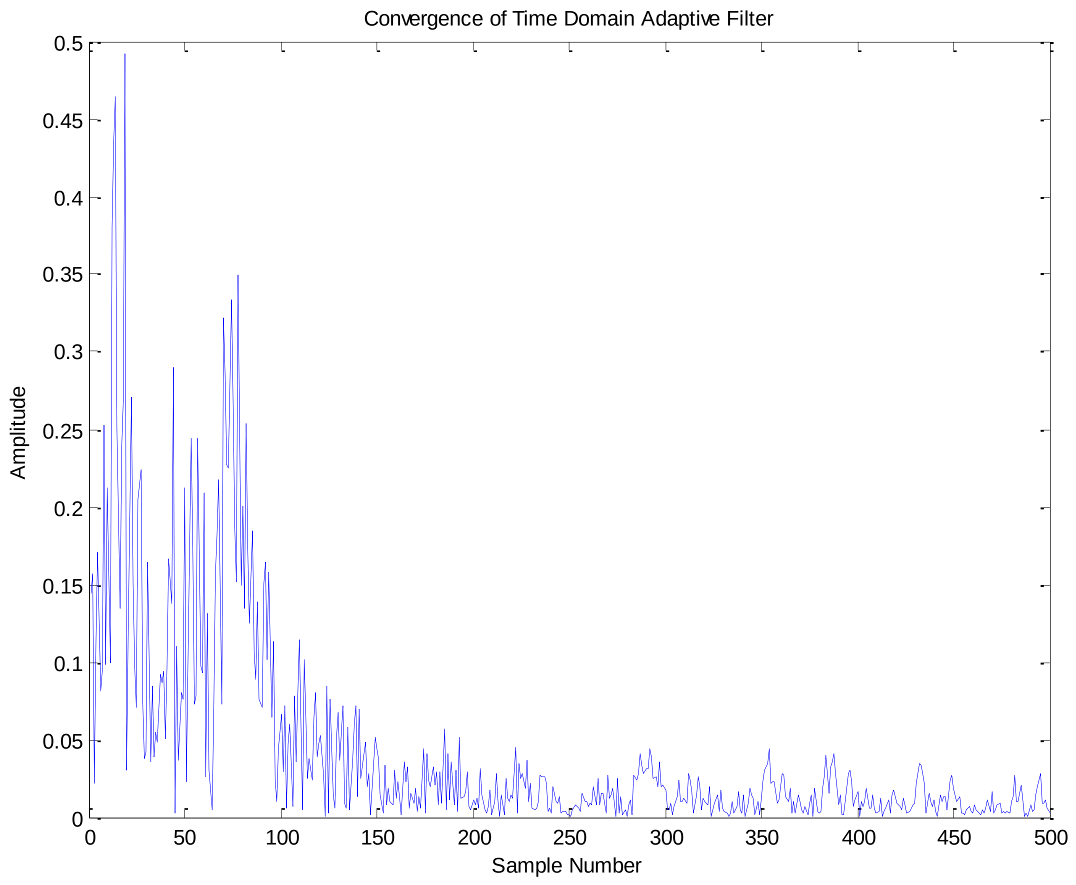

Figure 3.7 The error convergence of the Time Domain Adaptive Filter

An examination of Figures 3.3 and 3.5 show the relationship between the length of the impulse response of the unknown system and the number of required filter coefficients. Although the unknown system in Figure 3.3 is fifty values at sample intervals close inspection indicates the last twenty years or so contribute little to the system. From this we can expect about thirty samples are sufficient to describe the system. This is reflected in Figure 3.5 where the error is dramatically reduced after twenty or filter coefficients thus clear that twenty five coefficients are sufficient for the adaptive filter.
To illustrate the above points, the error signal shown in Figures 3.4, 3.5 and 3.7 were able to establish the value of μ to $3 \times 10\text{-}3$, and the value of N to 25. The applied input signal is a randomly generated signal comprising amplitudes of -3, -1, 1 and 3, and these amplitudes are common in telephone lines [3] An example of the applied input signal is shown in Figure 3.6. Figure 3.7 shows the convergence of time domain adaptive filter by plotting the difference between the desired filter response and real.

3.5 The Weakness of the LMS Algorithm

The LMS algorithm has a profound weakness, and that increases in computing costs to an undesirable level as the length of the impulse response increases. This is mainly due to the fact that the algorithm is in the time domain, leaving the algorithm in an obvious disadvantage when an impulse response is very long. The computing power required simply becomes too high for efficient use. There block LMS algorithm LMS versions that attempt to compensate for this problem in the time domain, frequency domain however adaptive filtering is the key to solving the problem very long impulse response.

In the next chapter we will discuss some algorithms first fixed frequency domain based algorithms for adaptive frequency domain.

Chapter 4 Filtering in the Frequency Domain

4.1 Transforms

4.1.1 Introduction

Before discussing frequency domain filtering should probably take a step back and study the transition of data between time and frequency domains. The representation in the frequency domain of a signal is merely an alternative representation. It is often the case that the signals are represented in the frequency domain so that the use of discrete transforms to reduce the processing required for signal processing applications, such as convolution. Although some changes between the two domains, the Fourier Transform is the most widespread. The advantages of the Fourier transform in other transforms include [2].The efficiency of the Fast Fourier transform

- Adequate representations of data for even short data lengths
- More faithful representation of data
- Components are sinusoidal and are not distorted when transmitted over linear systems
- A high degree of familiarity and thus a lot of development.

4.1.2 Discrete Fourier Transform

The discrete Fourier transform is a Fourier transform that can be used to transform the discrete-time signals to the frequency domain.

The equation used to calculate the discrete Fourier transform shown below.

$$X(k)=\sum_{n=0}^{N-1} x(nT)\,e^{-jkwnT} \qquad \textbf{4.0}$$

where k is the harmonic number of the transform component.
It can be shown that this equation is analogous to the equation for calculating the Fourier Transform for continuous time signals [2].

4.1.3 Properties of the Discrete Fourier Transform

Symmetry

An important property of the Discrete Fourier Transform can be illustrated by comparison of X (k) with X (k + N). The fact that these two elements are the same indicates the frequency with period N. In general, the amplitude spectrum is also symmetric around harmonic N / 2 [2].

Convolution

The discrete Fourier transform can be used to calculate the circular convolution if we use zeros increase linear convolution can be calculated. The time convolution theorem states that convolution in the time domain is equivalent to multiplication in the frequency domain [2]. Equation 4.1 illustrates this in a mathematical way

$$x(n) = x_1(m) * x_2(n) = F^{-1}\{X_1(k)X_2(k)\} \qquad \textbf{4.1}$$

where x, x_1 and x_2 are finite periodic sequences of equal length and * denotes circular convolution F^{-1} denotes the Inverse Discrete Fourier Transform. This is a very important property of the Discrete Fourier Transform in the context of this project, since digital filtering is a form of convolution it is clear that we can now use multiplication to perform filtering if we transform the signals we use to the frequency domain [2].

4.2 The Fast Fourier Transform

4.2.1 Introduction

The Fast Fourier Transform (FFT) algorithm is used to calculate the discrete Fourier transform (DFT) of a vector x or in other words, to convert the vector x to the frequency domain. It uses the built in redundancy in the DFT to minimize the number of calculations required and therefore make the algorithm more efficient [2]. Inverse FFT (IFFT) is a version of the FFT that converts the signals back into the time domain. [2]

4.2.2 The Computational Complexity

To calculate the DFT directly N2 complex multiplies are required, if the FFT algorithm is used the number of operations for N a power of 2 is reduced to (N / 2) log2 (2N) + N complex multiplications adds complex [5]. To maximize the efficiency of the FFT is important that the block length N is a power of two. Remember that adding extra zeros to a signal to be Fourier transform does not change the result of Fourier transform of this signal. The block length is the length of the input block for which the FFT is computed.

4.3 The Overlap-Save Algorithm

4.3.1 Introduction

The overlap-add and overlap-save are the two main algorithms implementing fixed-frequency domain convolution block and as we know from Section 2.2 of convolution and digital filtering are essentially the same. Therefore, these algorithms are two examples of algorithms that take advantage of the fact that multiplication in the frequency domain is equivalent to convolution in the time domain. In this project we focus on the overlay-Save method as it is more computationally efficient [2]. In the Overlap-Save algorithm input sequences overlap, a 50% overlap is usually considered the most efficient. The current values of the input sequence consisting of the current input block concatenated with the previous input block. This means that the latest N samples of the current input stream is stored to concatenate with the next input block. Circular convolution calculated results in the rotation of the circular artifacts, which appear as the last N samples of the IFFT output. These samples are simply discarded and the remaining samples are concatenated to give the output of the overlap save algorithm. The savings-matching algorithm was carried out in this project, included in the accompanying disc shaped overlapsave.m. To test the accuracy of the algorithm of its output was compared to the output of the filter function in MATLAB. The difference between the two methods was around 10-16. The algorithm is executed as illustrated in Figure 4.1 [4].

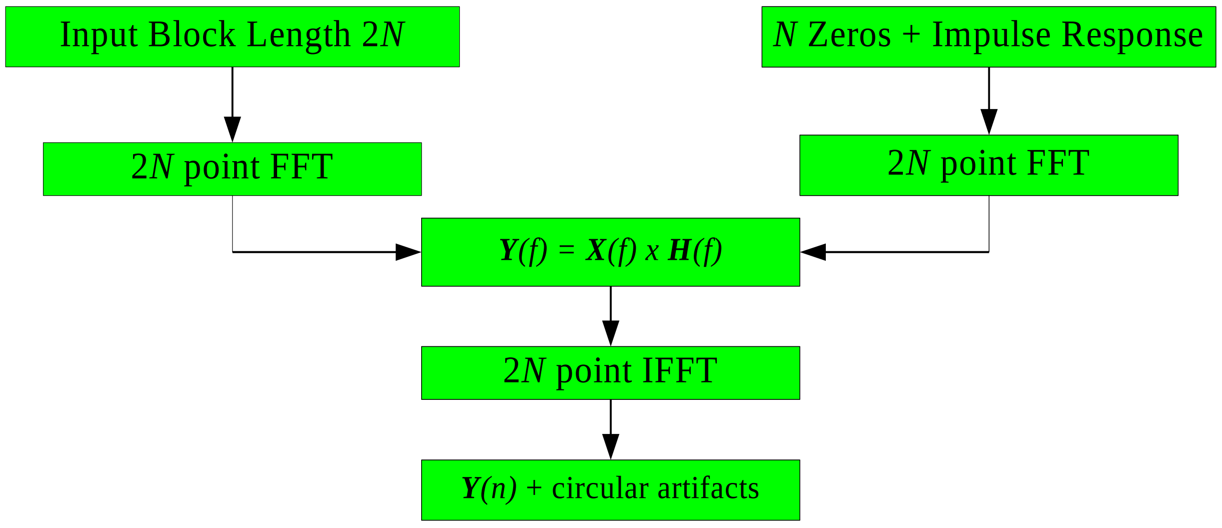
Input Block Length 2*N*
N Zeros + Impulse Response
2*N* point FFT
2*N* point FFT
Y*(f)* = ***X****(f)* x ***H****(f)*
2*N* point IFFT
Y*(n)* + circular artifacts

Figure 4.1 Block Diagram for the overlap-save algorithm.

4.3.2 The Overlap-Save Algorithm

Let *N* be the length of the impulse response of the system, the modem echo path as used in Chapter 3. The length of the input sequence is then twice this at 2*N*.

1. *N* zeros are added to the impulse response if necessary so that the result of the FFT will be the same length as that of the FFTs of the input sections.

2. The FFT of the impulse response is calculated and stored in memory because it will remain unchanged.

$$W(k) = FFT\{h(k)\} \qquad \textbf{4.2}$$

3. Next the current block of the input is taken and the FFT of it is calculated, for the first block there are *N* zeros in front of it for subsequent blocks the previous input block precedes the current input block.

$$X(k) = FFT\{x(k)\} \qquad \textbf{4.3}$$

4. The two FFTs are now multiplied , each element in one of the arrays will be multiplied by the corresponding element in the other. This procedure corresponds to convolution in the time domain. [1]

$$Y(k) = X(k).W(k) \qquad \textbf{4.4}$$

5. The IFFT of ***Y(k)*** must now be calculated to bring the results back to the time domain.

$$y(n) = IFFT\{Y(k)\} \qquad \textbf{4.5}$$

6. The second half of this result is dumped[1] for each convolution.
   The first half is added to an array as the output of the filter for the given input block.

7. The input block is updated applying 50% overlap and steps 3 to 7 are repeated.

[1] This seemingly important data can be simply discarded because zeros missing input data, which is usually added to convolution effects. A convolution rule is that if N2 is the length of the impulse response and N1 is the length of the input signal then N2-1 zeros are added to the input sequence and the N1-1 zeros must added to the impulse response allowing to obtain correct linear convolution. Note that the zeros are added at the beginning of the impulse response and in this case there is no corresponding zeros added to each block of input data and thus zeros instead of N data samples prior to N, what we have here is 2N data samples. With 50% overlap intact the second half of each convolution sum may be dumped as it contains data that is a result of the circular convolution.

8. This algorithm has been tried and tested in the overlapsave.m file, which is included on the disc accompanying this report. It is a fine algorithm so long as the filter coefficients are known. In the next section we will consider an algorithm that will enable adaptive filters to operate in the frequency domain.

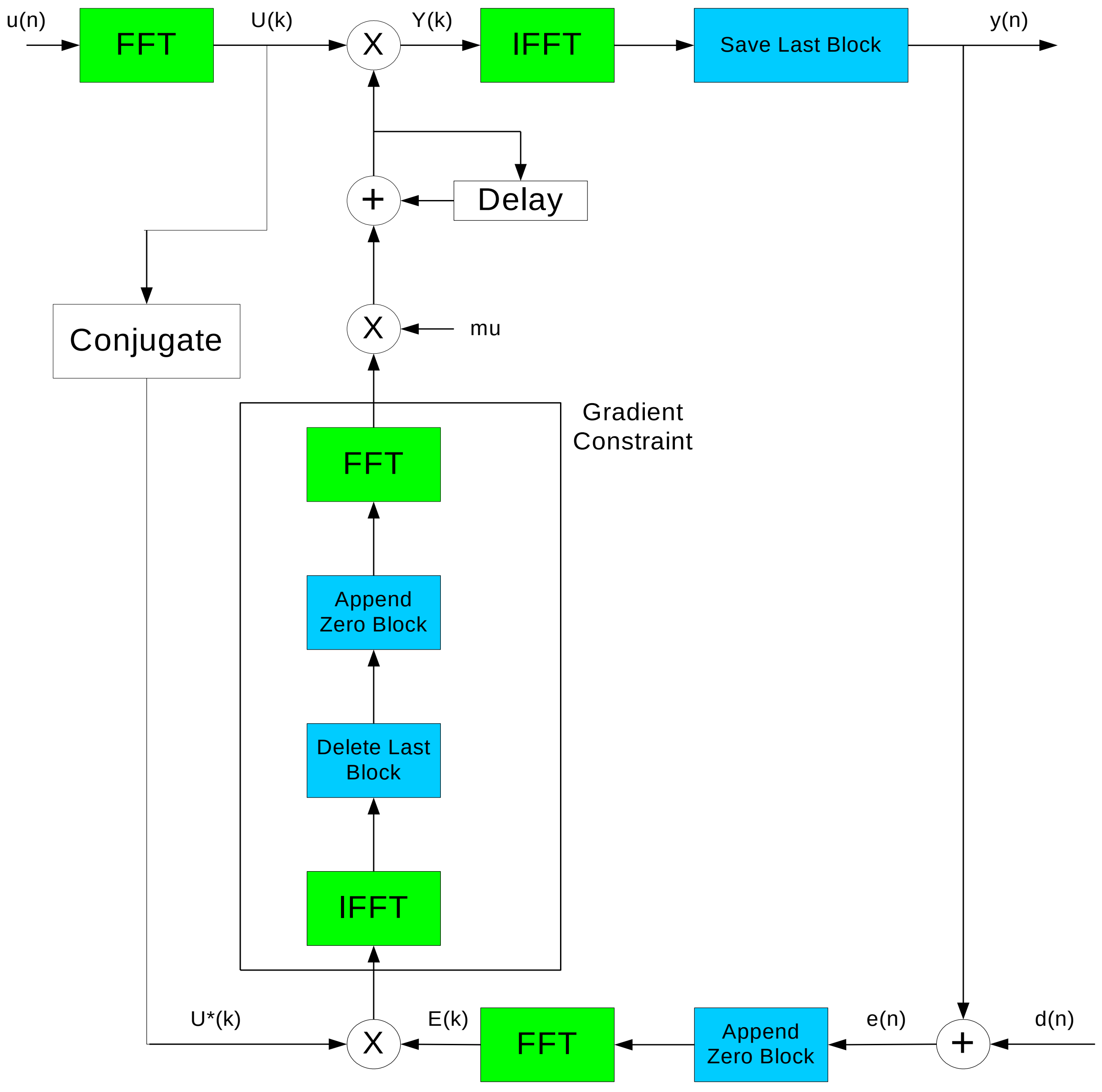

Fig 4.2 Block Diagram for the Fast LMS algorithm[1]

4.4. An Adaptive Frequency Domain Algorithm

4.4.1 Introduction

Some research on this topic quickly identifies rapid LMS algorithm [6] as the main frequency domain based adaptive algorithm and frequency domain adaptive filter based on this algorithm for the purpose of this project. The version of the Fast LMS explored in this project is based on the method of convolution superposition of savings, this makes filtering. Regarding the updating of the filter coefficients fast LMS algorithm is based on their equivalent time domain. A significant difference is that the fast LMS operates on blocks, introducing a latency time for the system. The block diagram of Figure 4.2 illustrates the flow of the algorithm [6]

4.4.2 The Fast LMS Algorithm

As in the overlap-save algorithm N is the length of the impulse response of the unknown system. 2N measurement blocks are taken from the input at a time with 50% overlap as before. W will donate the filter coefficients, which are initialized to zero and updated after each block.
The desired output is obtained by using the filter function in Matlab. The desired response of the adaptive filter is now known and can be used to update the coefficients correctly. Similar to the overlap-save algorithm add N zeros at the beginning of the input array to ensure correct results convolution

1. An input block of size 2*N* is taken from the input array, ***U*** the FFT of this block is calculated.

$$U(k) = FFT\{u(n)\} \qquad \textbf{4.6}$$

2. Now the Filter output can be computed by multiplying the FFT of the input block, ***U(k)*** by the Filter coefficients as updated by the previous iteration of the algorithm.

$$Y(k) = U(k) \bullet W(k) \qquad \textbf{4.7}$$

   This is transformed to the time domain by computing the IFFT of the above result.

$$y(n) = IFFT\{Y(k)\} \qquad \textbf{4.8}$$

   Due to circular convolution the first half of this result is simply discarded and the second half forms the output of the adaptive filter for the given input block.

$$y(n) = y\left(N + 1 \xrightarrow{upto} 2N\right) \qquad \textbf{4.9}$$

3. The error signal is computed next by means of simple subtraction to calculate the difference between the desired and the actual response.

$$\mathrm{e(n) = d(n) - y(n)} \qquad \mathbf{4.10}$$

where ***d(n)*** is the corresponding section of the desired response.

The error is brought into the frequency domain by adding N zeros to the start of ***e(n)*** and by computing a 2N point FFT and the result is called ***E(k)***.

$$E(k) = FFT\{zeros, e(n)\} \qquad \mathbf{4.11}$$

The Gradient Constraint

4. The conjugate of ***U(k)***,***U'(k)*** is found and this is multiplied by ***E(k)*** and the IFFT of the result is calculated. The second half of this result can be dropped due to circular convolution.

$$g(n) = IFFT\{E(k) \bullet U'(k)\} \qquad \mathbf{4.12}$$

$$g(n) = g\left(1 \xrightarrow{upto} N\right) \qquad \mathbf{4.13}$$

5. *N* zeros are now added to the end of what we are left with and the 2*N* point FFT of the resulting sequence is calculated and the result is multiplied by μ (the step size parameter)

$$g(n) = g(n) \text{ followed by N zeros} \qquad \mathbf{4.14}$$

$$W_1(k) = \mu \bullet FFT\{g(n)\} \qquad \mathbf{4.15}$$

this is the filter coefficient update factor, ***W₁(k)*** and is added to ***W(k)*** and this is how the update of the coefficients is conducted.

$$W(k+1) = W(k) + W_1(k) \qquad \mathbf{4.16}$$

6. This newly updated ***W(k*+1*)*** will now be used as the filter coefficients for the next block of input. An error may exist, however as ***W(k)*** is updated more often this error will diminish as is indicated in Figure 4.3 which shows the convergence of the filter coefficients to near optimum performance.

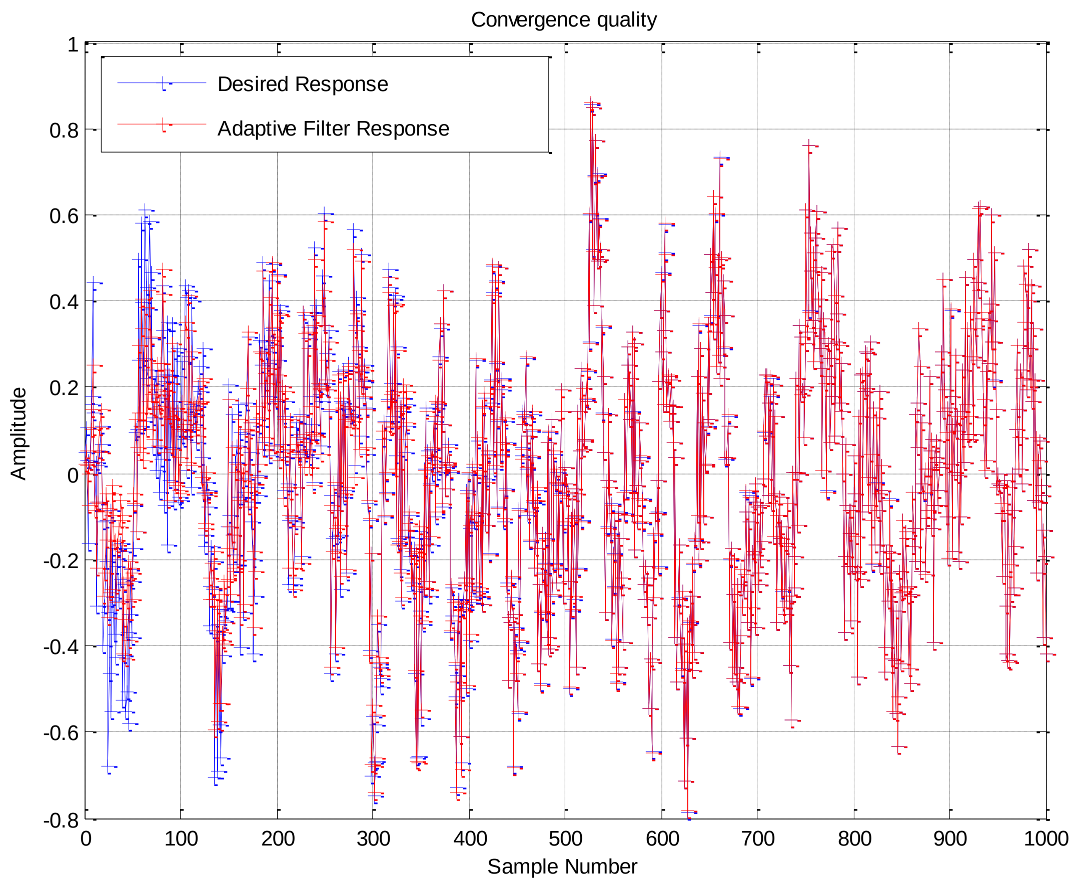

Figure 4.3 The convergence of the adaptive filter coefficients

Note that the desired response and the adaptive filter output produces almost perfectly map on each other after only a few input blocks have been processed. Convergence can be best illustrated in Figure 4.4, where the error signal is the difference between the two previous signals. These figures were obtained with the value of μ adjusted to 2 x 10-3 N set at 50, though this does not implement the efficiency of the FFT which is used to verify fast LMS algorithm convergence. N is set to the length of the unknown system, which in this case represents the echo path modem previously introduced. In the real world would require an engineer to estimate the length of the impulse response of an unknown system and then choose the next power of 2 to use the FFT algorithm efficiency. Again, the input samples would be as illustrated in Figure 3.6.

4.4.3 The First Input Block

Also notice in Figure 4.3 that the results for the first input block are also very good despite being without previous values on which to base the filter coefficients. This is because the special status given to the first block in this version of the algorithm. Initially, the filter coefficients are set to zero and therefore the output for the first block would also be zero. To avoid this problem, the error associated with the first input block is calculated y (n) being set to zero, which means that the error becomes very real answer desired, then this error is used to adjust the coefficients for filter and these are used to input the first block in place of the zeros, the coefficients are updated after that as usual in preparation for the next input block.

4.5 The Cross-Over Point

Now that we have examined both time and frequency domain applications it is time to compare the two domains in terms of computational complexity.

4.5.1 Time Domain Implementation

To calculate an actual output block from a real input block for a given filter we need N multiplied to generate each output value N. N2 also need to update the filter coefficients. This gives a total of 2N2 multiplied by the time domain LMS algorithm

$$\textit{Time Operations} = 2N^2 \qquad \mathbf{4.17}$$

4.5.2 Frequency Domain Implementation

There are 3 FFTs and 2 IFFTs each of length 2N where N is the length of the bock size, since the computation involved in calculating a 2N point IFFT is the same as that involved in calculating the FFT we can group IFFTs and FFTs together. Now we have 5 FFTs requiring 10N log2(2N) real multiplies and 5N log2(2N)+10N real adds [5]. In the rest of the algorithm there are two 2N complex vector products, relating to 16N real multiplies and 8N real adds, there is also one 2N point complex tap vector update requiring 4N real multiplies. Giving multiplies and adds an equal weighting we have an approximate total number of operations for the frequency domain implementation.

$$\text{Frequency Operations} = 15N \log_2(2N)+38N \qquad \mathbf{4.18}$$

Note that this is only an approximation since in reality real multiplies are more computationally Note that this is only an approximation, since in actual fact are more computationally expensive multiplies that really adds, however, the figure can be used for comparison purposes. For 10,000 samples per second, the computational cost of frequency domain and time domain approaches were compared and the results can be seen as operations per second requirements of the two approaches. The number above for operations in the frequency domain depends on N is a power of two and therefore a comparison with the time domain values are compared to N is a power of two. The crossing point was obtained 128, if N is however not allowed to be powers of two for the time domain, the crossover point can be less as shown in Figure 4.5. As N increase the proportion of the time domain operations to the operations in the frequency domain also increases linearly as shown in Figure 4.6

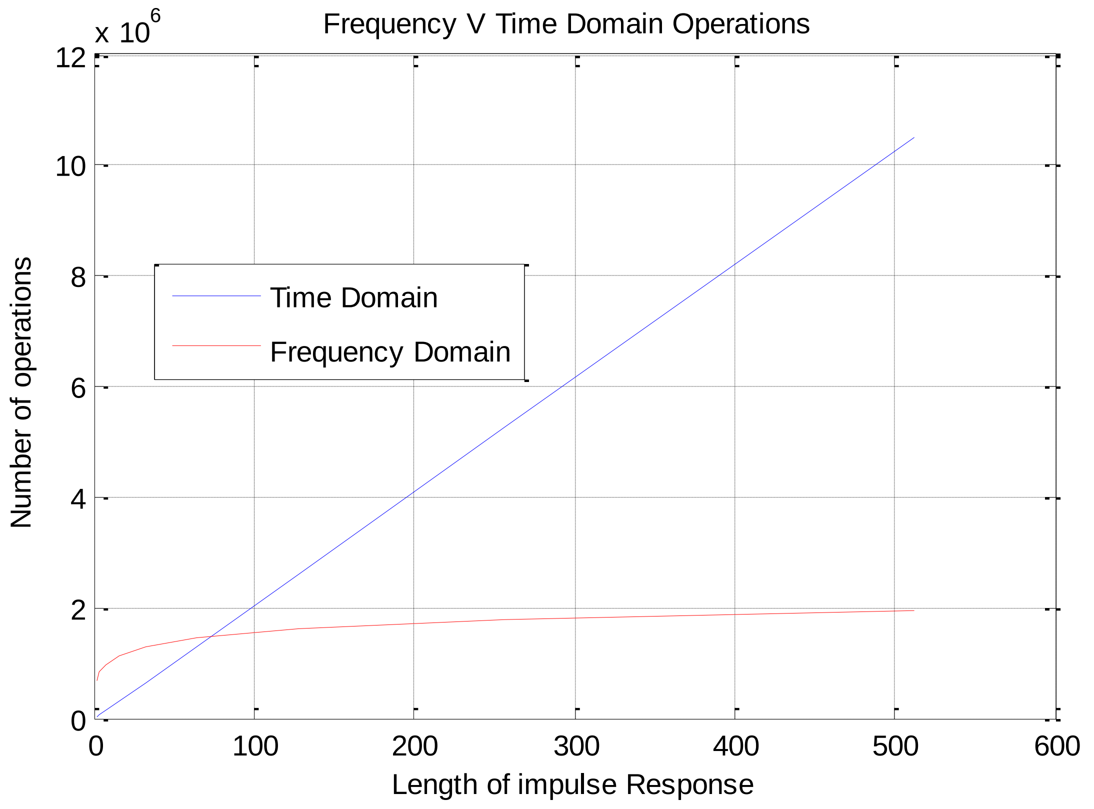

Figure 4.5 Illustration of the Crossover Point

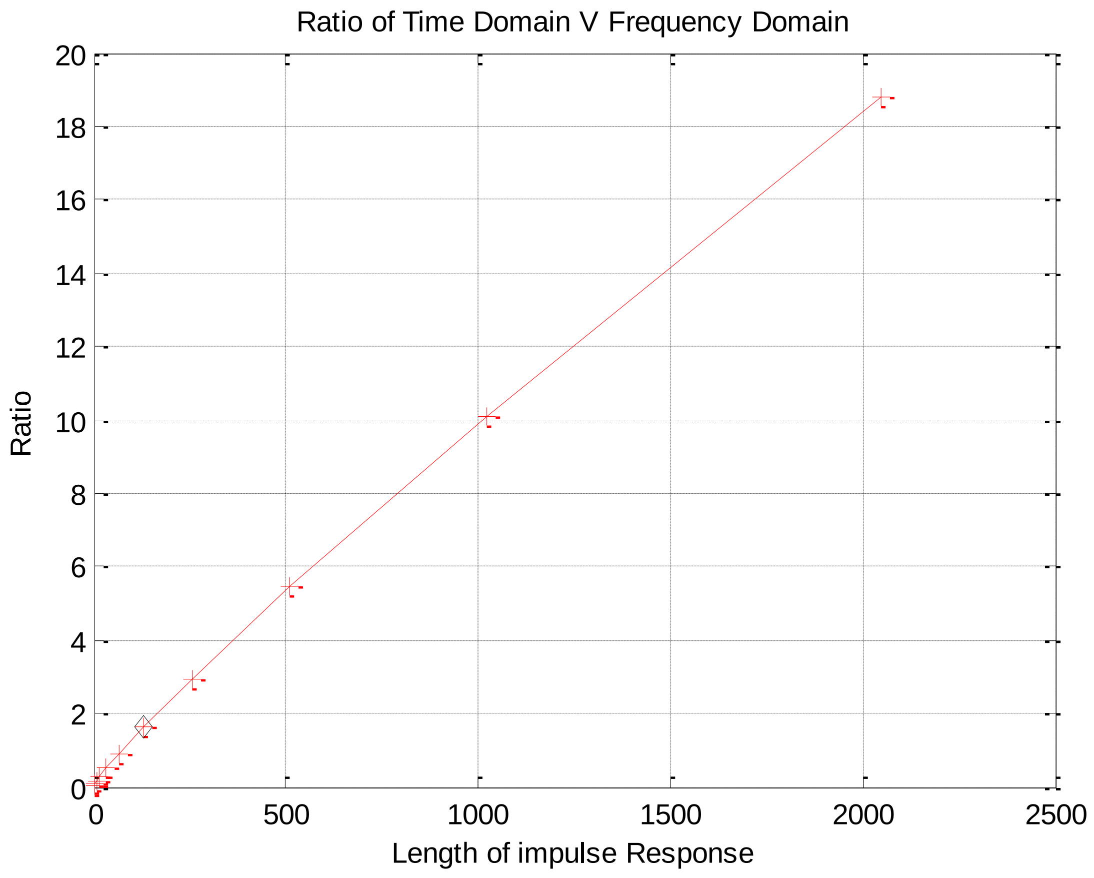

Figure 4.6 The ratio between the time and frequency domain operation count

Chapter 5 Implementation of the Fast LMS

5.1 Introduction

The goal is to have a system that will then process values while enjoying the next set of values as in Figure 5.1. This system would not be enough to be classified as real-time, but rather as close to real time. To test real-world values of the system are read from the data acquisition card from National Instruments (NIDAQ). Unknown system used as a test system is merely a lowpass filter with a cutoff frequency of 140 Hz output of this filter is read through NIDAQ card and a block stored in the same memory, and this sends the adaptive filter algorithm as the desired response. The input to the filter is of the form shown in Figure 3.6. This will be sent to the filter, the unknown system through NIDAQ card. To carry out the task of reading in values, while values previously read are going to use wire processing. Topics that are not quite something that is true in the programming of every day and we are going to introduce the concept of topics in the next section. For the purpose of this project was considered to write a function to implement the FFT would take too long and so a function version is available online at [7].

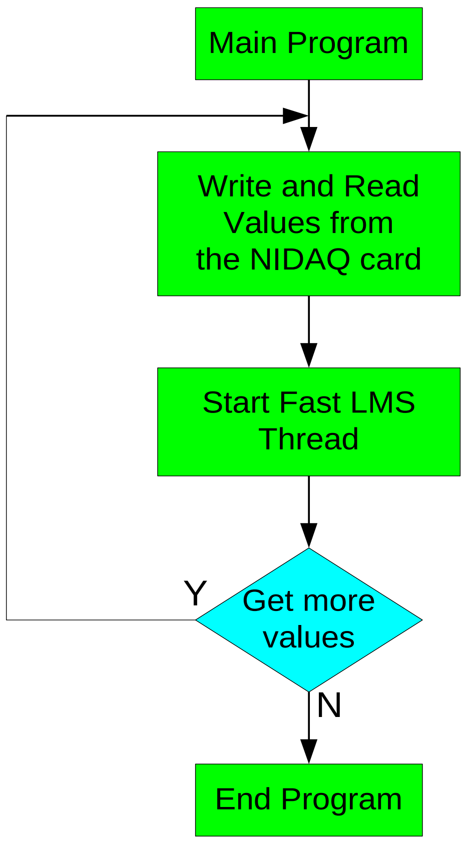

Figure 5.1 Block Diagram Representing the two tier Process

5.2 The NIDAQ Card

The National Instruments Data Acquisition (NIDAQ) card is a powerful tool that allows the computer to communicate with the outside world in a manner simple for the user to follow and to use. The NIDAQ means that we can both write data to and read data from the outside world. The card contains analogue to digital and digital to analogue converters so that the user can input and output analogue signals to the specified pins on the connector. National Instruments provide some sample programs to illustrate the use of common NIDAQ functions. The two functions that we are interested in here are *AO_Write* and *AI_Read*, since these are the two used in this project. *AI_Read*, reads an analogue input voltage, converts it to its digital equivalent and returns the result as an integer code, this code can then be converted to the correct voltage value. AI_VRead would do the above conversion for us, however it was felt that this code may contain some unnecessary extras and so *AI_Read* was used. *AO_Write* takes an integer code representing a voltage and outputs this value to the specified channel. Specification sheets and other information including pin assignments and more detail on the functions can be found at [8].

5.3 Threads

5.3.1 Introduction

A thread is a specified execution path in a process. Most processes consist of only one thread, the primary thread, which is created when the process is begun. Processes can, however consist of many threads. The primary thread can create another thread to take over some of its workload, indeed this thread and all other threads can do the same. The idea of threading is to utilize as much of the processing power of the processor or processors as possible. Generally we should consider creating a new thread when the program encounters asynchronous activity. Background tasks that the user needs to know nothing about can be carried out with efficiency by using threads [9].

5.3.2 Thread Attributes

- Each thread is allocated its own stack from the owning process's address space.
- Each thread has its own thread context, reflecting the state of its processor registers when it was last executed.
- Every thread has a priority associated with it in order for the system scheduler to identify the next thread that it should execute. The scheduler gives priority to threads which have the highest priority associated with them. A thread inherits its priority according to the

process that creates it. [9]

5.3.3 Thread Synchronization

Threads cannot always just be created and run in a random manner with everything expected to run smoothly. Threads must be able to work together and interact, to do this we may have to work on the timing control. We can do this in one of two ways by using priority or synchronization. Priority levels should always be set sensibly, for example simply assigning a high priority to all threads defeats the purpose. Synchronization means that the threads are coordinated and that their sequences of tasks happen in a required manner. To allow efficient synchronization of threads Win32 supports synchronization objects, such as *mutexes*, *critical sections*, *semaphores* and *events* [9]. Now if a thread is to perform a coordinated task it may be forced to await a response from one of the aforementioned synchronization objects, once this is received it is free to continue at least until its next synchronized task. If a thread is waiting for a signal from a synchronization object it is removed from the system's queue to avoid wasting precious processor time. It was thought that *mutexes* could be used as a synchronization tool in this project, however it was found that they were not suitable. The next subsections describe the *mutex* object and explains why it could not be used and how the problem was resolved [9].

5.3.4 The *Mutex*

In essence a *mutex* is a narrow gate that lets threads through one at a time. Generally this gate leads to a small section of code that needs exclusive access to shared data before the code can be executed. The *CreateMutex* function creates a *mutex* and returns a handle to it. [9]

A *mutex* can be used to synchronize threads running from multiple processes. The threads however must have process relative handles to the same *mutex* object. This can be achieved using the aforementioned *CreateMutex* function. If a second call is made to this function with the same *mutex* name specified the system will recognize this and simply return a handle to the previously created mutex [9].

5.3.5 Problem Encountered

In the case of this project exclusive access to the input data (read in from the NIDAQ card) was required to avoid the data being processed being overwritten prematurely. Synchronization is obviously crucial here to ensure that data read in from the NIDAQ is processed before the *mutex* is passed back to the thread reading in from the NIDAQ card. This is where the difficulties were encountered. Both threads contained *for loops* and each *for loop* contained a call to

WaitforSingleObject [9], which means wait for the mutex handle. The problem was that the *same thread was grabbing the mutex* before the second thread got a chance. To solve the problem we found that we could only remove the *mutexes* and simply begin a new thread each time data was read in, using the *_beginthread* [9] function. Efficiency now became the main concern and to remove some of the inefficiencies of starting a new thread each time global variables were used.

To verify this C implementation of the adaptive filter it and the Matlab version were run using the same input and desired signals. The outputs of both implementations were compared and the error was found to be of the order of 10^{-16}. Having verified the operation of the Matlab version in chapter 4 it can be said that the C implementation is now also verified. The code is included on the disk accompanying this report.

In the next chapter we will explore possible applications of the adaptive frequency domain filter.

Chapter 6 Further Applications

6.1 Introduction

As part of this project we investigated the possible applications of the LMS adaptive filter Fast mode system identification. One possible application was investigated particularly adaptive equalization. Essentially, implying adaptive equalization is compensating for unwanted linear channel (such as telephone lines) additional filtering features. Inter Symbol Interference (ISI), which is the interference caused by the received symbols overlap requires rather precise equalization or compensation to reduce ISI caused by the channel. [3] In the identification system generally depends on the availability of a desired response. This is not always practical, so instead of looking for the real answer simply desired signal can generate the same preset (the training sequence) at both the transmitter and receiver. Such transmission sequence and application version generated in the receiver after a time equal to the channel transmission delay as the desired response means that an appropriate input signal and the corresponding desired response is available to establish the filter coefficients . This is done before the data transmission and depends to some extent on the characteristics of the channel does not change dramatically during the duration of the call in the case of application to the telephone network. The training sequence must be at least as long as the impulse response of the equalizer to ensure the necessary density in the channel bandwidth to be matched. [3] After this training period the training sequence generator is turned off and the channel is now ready for data transmission as illustrated in Figure 6.1. During transmission of data decisions are used to determine which symbol was transmitted. A decision maker is simply a device that decides which symbol was transmitted. Since the release of this decision-making is reliable, we can use it as the desired response for data transmission that enables adaptive equalizer to track small changes in the channel. [3]

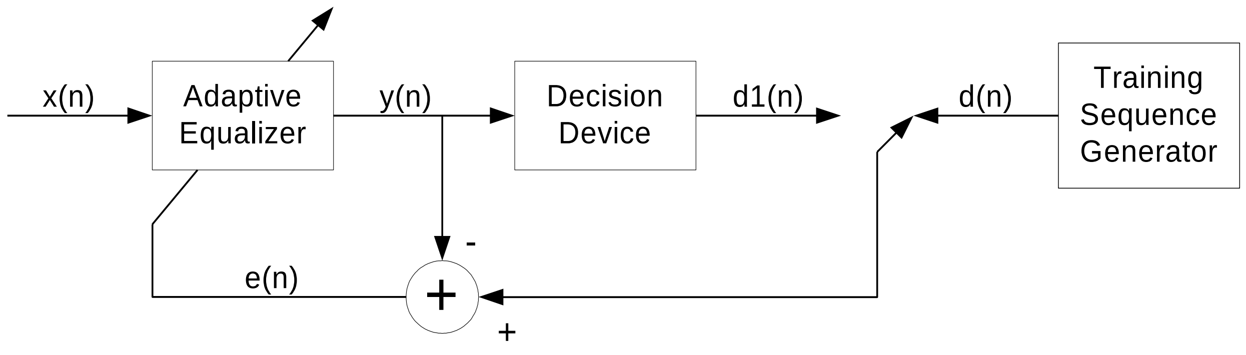

Figure 6.1 A block diagram of the adaptive Equalizer

6.2 Time Domain Adaptive Equalizer

The time-domain adaptive equalizer is investigated based on the LMS algorithm. The training sequence is a randomly generated sequence of 1 and -1, this sequence is used as the input also delayed by ten samples and is applied as the desired signal for the duration of the training period. Although the exact delay is unknown, as long as the chosen value is sufficient adaptive filter will adapt to introduce further delay if necessary. After the training period decision making can be used to identify which symbol is transmitted and provide the filter coefficients have converged properly during the training sequence can be used as the desired response. Figure 6.2 shows the convergence of the filter for a training period of length 200 with a value of μ set to 0.8 for the training period and reduced to 0.5 when applying the decision maker. The mean square error is calculated as the difference between the desired response and the actual response. Suitable values of μ chosen simply on the basis of trial and error in order to investigate this application. M value is highest during the training period to provide fast convergence and reduced to the period of data transmission for providing finer adjustment of the filter coefficients during this period. Unknown system to be identified is the echo path modem used in chapter 3 and therefore the length of the equalizer chosen was 25.

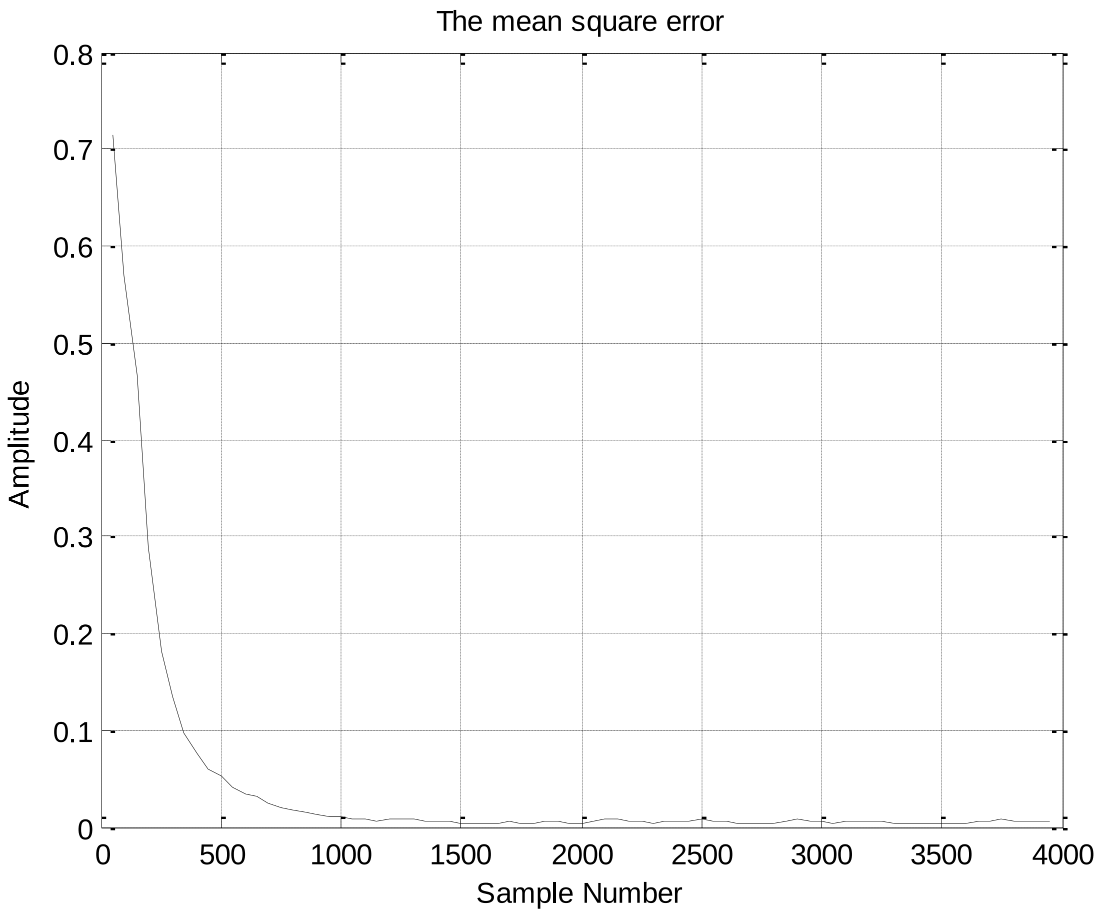

Figure 6.2 The convergence of the time domain adaptive Equalizer

6.3 Frequency Domain Adaptive Equalizer

This project investigates a frequency domain based adaptive equalizer LMS algorithm fast. The training sequence is the same as that used for the time domain equalizer, except that this time has to be divided into blocks of size N, where N is the length of the impulse response selected to be a power of 2. N was chosen to be 32 according to the discussion in Chapter 4. The desired response is delayed again for ten samples. The training sequence has a duration of 20 input blocks of size N and after using the device of choice. Although convergence was not achieved great figure 6.3 shows that the tie will certainly not converge.

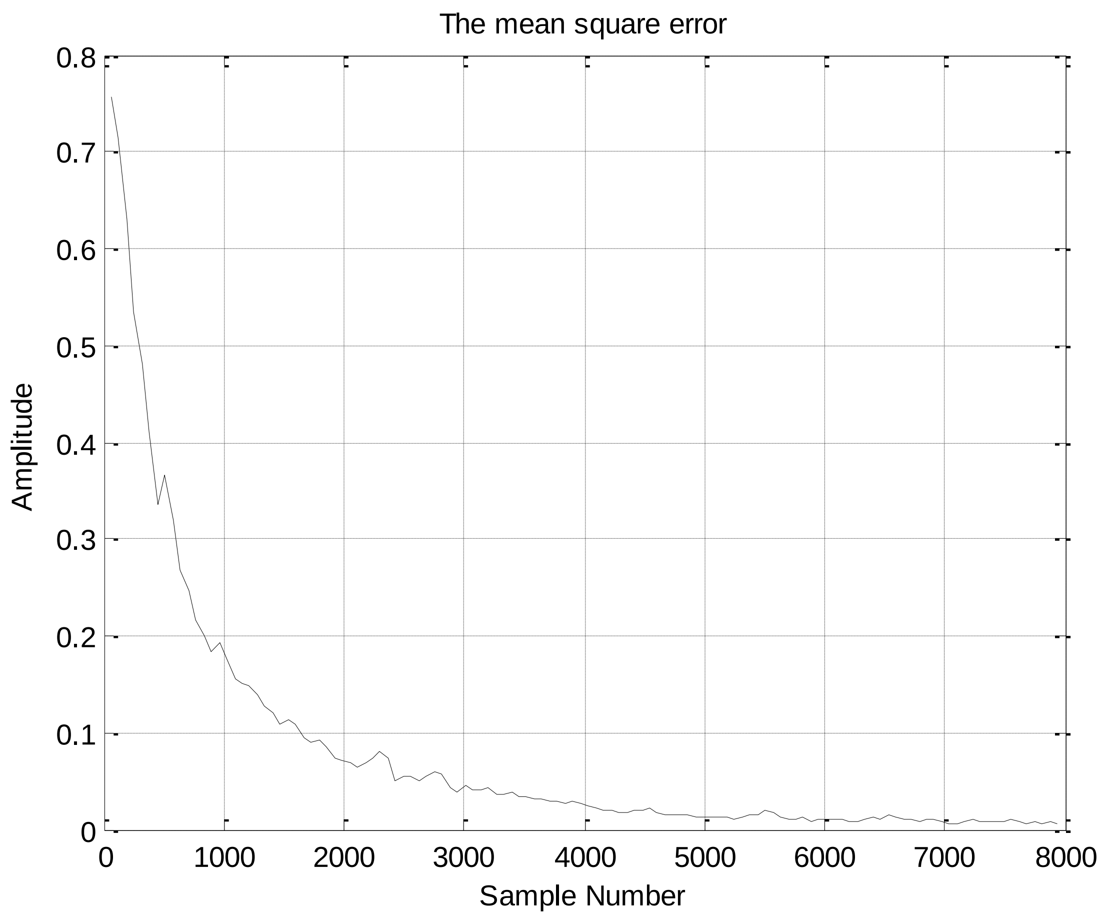

Figure 6.3 The convergence of the frequency domain adaptive Equalizer

Chapter 7 Conclusion

7.1 In Conclusion

In this study adaptive signal procesing is introduced by the transmission of a daily application in echo cancellation in the telephone system. An introduction to digital filtering was introduced then to give some background on the basic idea of digital filters and why so much work is put into them instead of analog filters. Convolution concept is introduced, which helps depict digital filtering as a mathematical process. Chapter 3 explains adaptive filtering, in particular mode system identification. The LMS algorithm is introduced as the main adaptive algorithm in the time domain and its operation is discussed. An alternative representation of signals in the frequency domain is then introduced, which allows the convolution of two signals is calculated in a much more efficient. The cost of converting the signals to and from the frequency domain should be noted however, and filter impulse responses short that is too high to allow frequency domain filter replace filter time domain. Substantial savings can be made as though the impulse response increases, a crossing point was portrayed approximate. This report presents a possible implementation of the fast LMS in the C programming language and a possible application as an adaptive equalizer explored.

7.2 Achievements

It was expected that the crossover point between the time domain and frequency domain implementation is well defined but difficulties in identifying the suitable coefficients for multiplication and prevented it adds, however, is considered to be given a good guideline figure.
The Matlab simulation of a time domain adaptive filter worked well, the investigation of the effects of varying N, the number of filter coefficients and □, the step size parameter followed and the results are portrayed. Simulation was achieved adaptive filter frequency domain in Matlab, the appropriate value of □ chosen in this case was in a trial and error basis. The value of N, for this filter is intuitively needs to be at least equal to the length of the impulse response of the unknown system similar to the methodology in the domain of time and then go to the next power of two to allow the use of FFT efficiency.
The translation of the fast LMS algorithm to the programming language C was achieved with the development of a real-time architecture suitable for real time operation using a low pass filter of the first order simple as a test system.

References

[1] "Communication Systems" 4th edition, Simon Haykin

[2] On the Complexity of Frequency Domain Adaptive Filtering, Neil K. Jablon

[3] "Digital Signal Processing" by A Vallavaraj & C Gnanapriya

[4] "Adaptive Equalization" by L.Hanzo, C.H. Wong, M.S Yee

[5] IEEE Transactions on Signal Processing, Vol. 39,No. 10, October 1991

[6] www.music.miami.edu/programs/Mue/mue2003/research/jvandekieft/jvchapter2.

[7] http://www.ee.calpoly.edu/~jbreiten/C/

[8] http://www.interex.org/pubcontent/enterprise/jul01/13parksh.html,

[9] "Adaptive Filter Theory " by Simon Haykin.

[10] en.wikipedia.org/wiki/adaptive filter

Bibliography

[1] "Signal Processing Algorithms in Matlab"
S.Stearns and R.David, Chapter 9