CONTENTS

ACKNOWLEDGEMENTS

I wish to thank sincerely my project supervisor Dr. Mazim Mahmoud for his continual help and encouragement throughout the project. I would also like to thank Professor T. Alukaidey and G. Pissanidis for their invaluable help in understanding the inner workings of the DSP used in my project.

ABSTRACT

This report presents an attempt to create an echo canceller using a DSP (Digital Signal Processor) chip. The report goes through the steps taken to design the system and implement it on Analog Devices ADSP ® 21065L EZ-KIT Lite. The filter used in the implementation was a transversal FIR filter and the algorithm used for updating the adaptive filter coefficients was LMS (Least Mean Square). The system performs well for echo delays of up to 10ms with an attenuation of approximately 20dB of echo attenuation. The number of filter taps used was 64.

1.0 INTRODUCTION

1.1 Chapter Overview

This chapter explains the project giving information about its aims and objectives. It also gives some background information on the tools used and the layout of the project.

1.2 Project introduction

The very concept of communication involves the presence of at least one “speaker” and a “listener” that interchange roles. A “speaker’s” task is to transmit to the “listener” some information (data). This data propagate through a medium usually either air or a cable conductor. During this propagation of data part of it are reflected back to the “speaker” thus an echo is created. If the delay between the transmission of the data signal and the echo signal is small and at the same time the amplitude of the echo signal is small compared to the initial transmitted data then the echo signal is not noticeable. If the delay exceeds a few tens of milliseconds and its amplitude is not negligible compared to that of the initial transmitted signal the echo is noticeable and presents a communication problem. In voice data communications is a nuisance and in data communications can cause data corruption or be falsely mistaken as valid data itself.

This report focuses on the elimination of echo in communications by the use of an adaptive filtering technique.

1.3 Project Aims and Objectives

The objective of this project was to create an echo cancelling module for use mainly to deal with acoustic echo, which results from the reflection of sound waves and acoustic coupling between the microphone and loudspeaker as well as with electrical echo, generated the two-to-four wire conversion hybrid transformer due to network impedance mismatch.

The echo canceller would have to utilize a DSP (Digital Signal Processor) for processing the necessary adaptive filtering algorithm. The DSP used was the ADSP-21065L SHARC. The algorithm was the well known and established, LMS (Least Mean Square).
To achieve the objective the work was roughly split into the following three stages:

STAGE 1:
Derive a system design and figure its mathematical modelling. This was also the first aim.

STAGE 2:
Set-up the DSP hardware and start the implementation of the theoretical design.

STAGE 3:
Test and debug the system to achieve real-time echo cancellation.

1.4 Equipment Used

Below is a list of the equpment used to create and test the echo canceller system.

- Signal generator1: LEVELL Function Generator Type TG 303
- Signal generator2: LEVELL Function Generator Type TG 303
- DSP Chip: ADSP© 21065L EZ-LITE Developing Board
- Computer/Software: PC running windows98 with Analog Devices VisualDSP software installed
- Oscilloscope1: HAMEG Storage Oscilloscope HM 205-3
- Oscilloscope2: Digital Oscilloscope (used for generating printout results)
- Set of custom audio Cables
- Noise Generator: H01-3722A Noise Generator

2.0 BASIC THEORY OF ECHO

2.1 Chapter Overview

This chapter describes basic information about the types of echo in communication systems and ways to eliminate the problem caused by the echo effect. It briefly describes the operation of an echo suppressor and an adaptive echo canceller.

2.2 What Is Echo

Echo as mentioned in the introduction is the delayed or distorted signal reflection that returns to its source. There are mainly two types of echo the acoustic echo and the electrical echo.

2.2.1 Acoustic Echo

When we have a microphone in close proximity to a loudspeaker what usually results is acoustic coupling between the two. This problem was first noticed when the first telephones that offered a loudspeaker option for hands-free conversations. The speaker's voice is coupled through the listener's microphone and so an echo is created. The following figure (fig.1) illustrates the procedure.

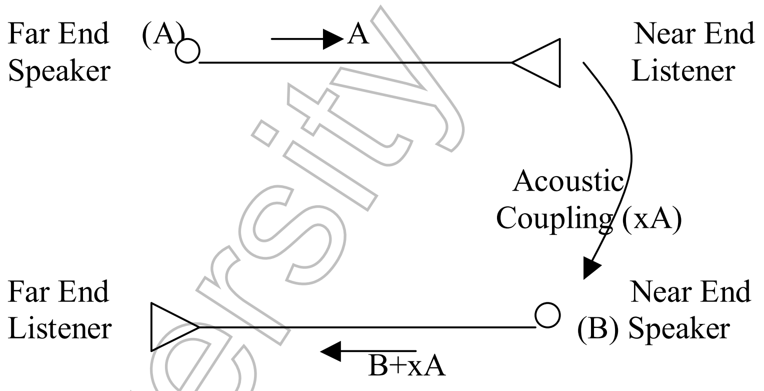


Figure 1 **Echo generation by acoustic coupling at the near-end speaker**

2.2.2 Electrical Echo

Electrical echo is the echo generated whenever a transmission line carrying a signal is not properly terminated. In a telephone network the main source of electrical echo is at the hybrid when connecting a two-wire transmission line (telephone) to a four-wire transmission line to allow for full duplex communication. The following figure (fig.2) illustrates the concept:

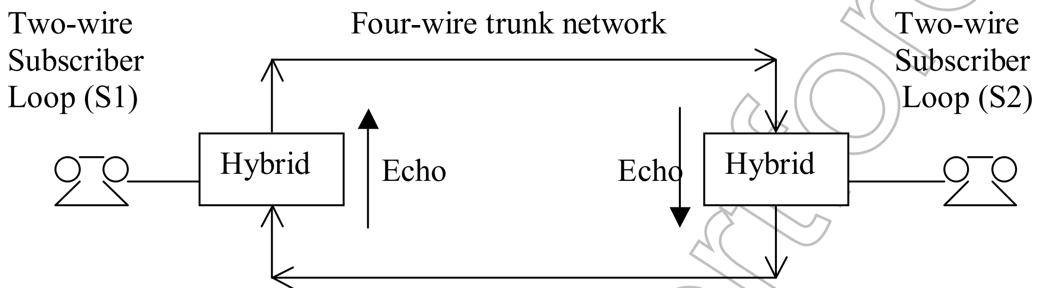


Figure 2 **Echo Transmission within a telephone hybrid network**

The hybrids function is to direct the signal energy from S1 and S2 to the upper and lower paths of the four-wire network circuit without allowing any leakage back to the two sources over the opposite direction line pairs.Ref[1]. Unfortunately due to the impedance mismatching some of the transmitted signal returns to its original source thus an echo occurs. A delay of up to 1-2 milliseconds can be tolerated or even go unnoticed but as the signal round trip increases the echo becomes more noticeable and annoying. Typical values for long distance communication (satellite communication) round trip is 300-600 msec. delay. This proves to be a serious problem.

2.3 Echo Cancellation

There are various techniques for cancelling echo. The two most prevailing in everyday modern communication systems are Echo Suppression and Adaptive Filtering. Echo suppression the oldest technique for echo cancellation dates back to the 1950's. It was used both in terrestrial and non-terrestrial communications but exhibited poor performance in the later. Adaptive filtering was first introduced in 1960's by AT&T Bell laboratories and progressively became more and more popular as digital technology allowed compact, fast and economic real time signal processing units (DSP's).

2.3.1 Echo Suppresser

Echo suppresser is mainly a device that detects the level of speech on an active channel. Depending on the threshold of that level anything above is treated as speech and passes through as opposed to anything below that is treated as noise or echo and is therefore heavily attenuated. Such a scheme may work satisfactorily for terrestrial circuits and echo delays of less than 100ms but for satellite communication were 400ms echo delays are typical, performs poorly. Another problem with such a system is that it is heavily depended on the speech/echo classification system it involves. This sub-system discriminates between forward speech and echo, based mainly on the signal amplitude level, high level for speech, low level for echo and is exactly this feature that poses a problem, when sometimes it allows high level of echo to pass through as speech and low level of speech to be attenuated as echo. Furthermore an echo suppressor cannot operate during double-talk and produces choppy echo. These are also the main reasons why echo suppressors are being replaced by adaptive echo cancellers.

2.3.2 Adaptive Filter Echo Canceller

As mentioned in the introduction adaptive echo cancellers were initially developed during the 1960's by AT&T Bell laboratories. The implementation of an adapting echo canceller system was introduced in a satellite network where echo delays exceeded 600ms. The project was a success. As illustrated in figure 3a,b below the basic function of an adaptive echo canceller is to predict the echo Y(n) added to the receive path signal d(n) and subtract it thus giving a clear output signal e(n). This echo cancelling system is appropriate for both types of echo namely electric (fig.3a) and acoustic (fig.3b).

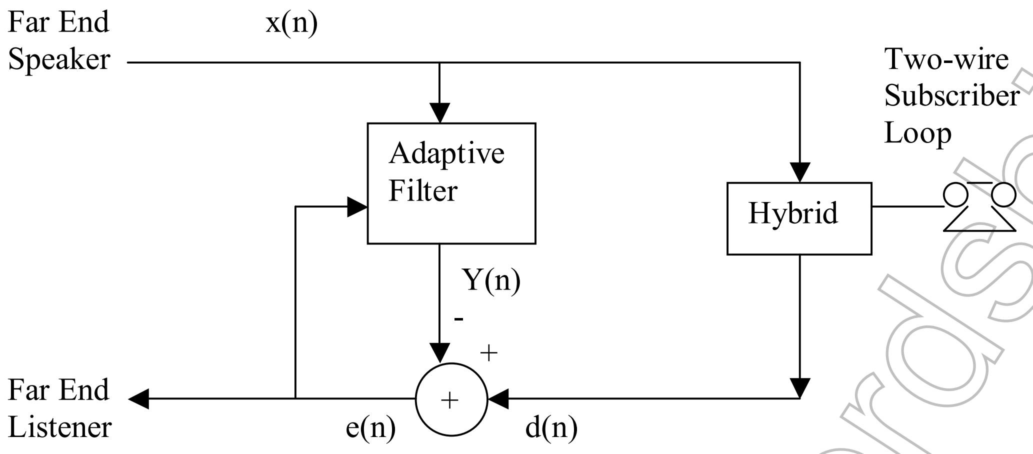


Figure 3a **The adaptive filter removes Far End Speaker echo from hybrid return path d(n). The location of the filter is as close to the Far End Speaker/Listener as possible.**

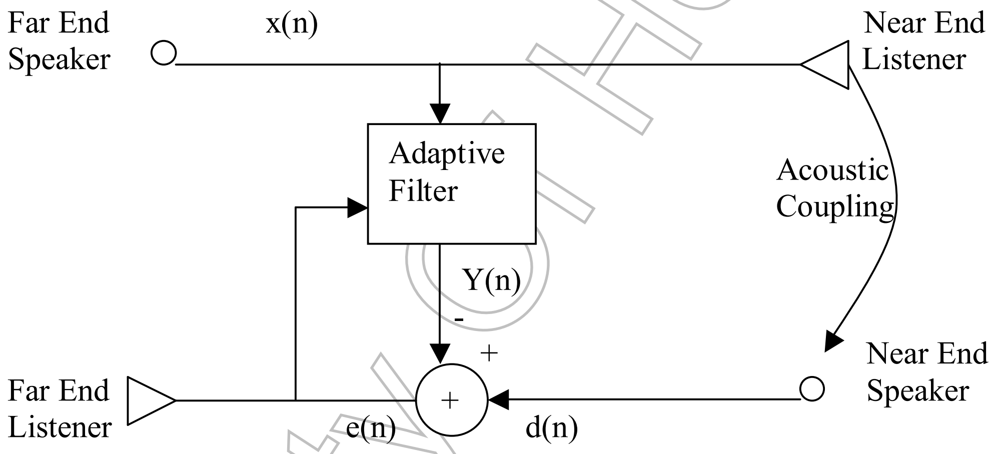


Figure 3b **The adaptive filter removes Far End Speaker echo from the Near End Speaker path d(n). The location of the filter is as close to the Far End Speaker/Listener as possible.**

The adaptive filter is a digital filter with variable coefficients, which means its frequency response can be modified. The coefficients update according to some criterion to give the desired prediction for the echo signal by means of an algorithm that continually monitors the error output of the system. More specifically the predicted echo is generated by the input signal x(n) and the system error e(n),

$$e(n) = d(n) - Y(n) \qquad \textbf{equation 1.}$$

where :

e(n) is the system output
d(n) is the incoming signal with the actual echo and
Y(n) is the predicted echo.

There are many types of adaptive filter designs used in a variety of applications where frequency manipulation of a signal is needed. In this project a transversal FIR (Finite Impulse Response) filter was used and is discussed later on chapter 5.3.

3.0 HARDWARE

3.1 Chapter Overview

This chapter gives information and basically presents the hardware used in this project. Namely the ADSP 21065L SHARC digital signal processor.

3.2 Analog Devices ADSP21065L

Analog Devices is an American company that specializes in designing and manufacturing a variety of linear, mixed-signal and digital signal processing integrated circuits. ADSP 21065L is a member of a large family of DSP chips utilising the Super Harvard Architecture (SHARC). It is a 32-bit general purpose DSP that allows programming in fixed or floating-point arithmetic. The chip comes with a developing board under the package named ADSP 21065L SHARC EZ-LAB.
The following figure (Fig.4) illustrates the general structure of the device.

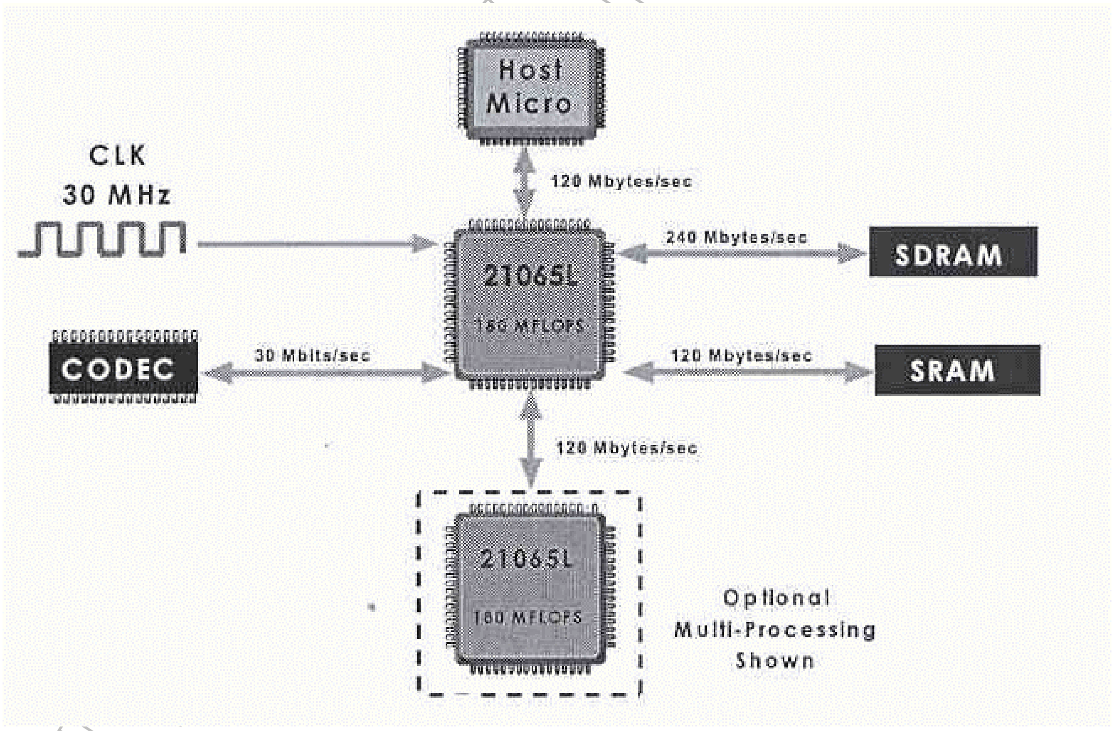


Figure 4 **General Overview of ADSP 21065L**

3.2.1 The Processor

A powerful 32-bit processing 32-bit words essential for processing 20- and 24-bit input signals. It is capable of 60 MIPS (Million Instructions Per Second) with an average 180 MOPS (Million Operations Per Second) and 198 MFLOPS (Million Floating Point operations per Second). It executes one instruction per cycle from a 16K* 32-bit (544 Kbits) of user-configurable on-chip memory that reduces memory access bottlenecks and improves processing speed. It also incorporates two Data Address Generators (DAGS), an advanced program sequencer, one result Bus and three arithmetic units (ALU/MAC/Shifter).It works at 3.3Volts. Figure 5 below illustrates the internal processor architecture for the ADSP 2100 family processor in which ADSP 21065L is included Ref[2,3,4].

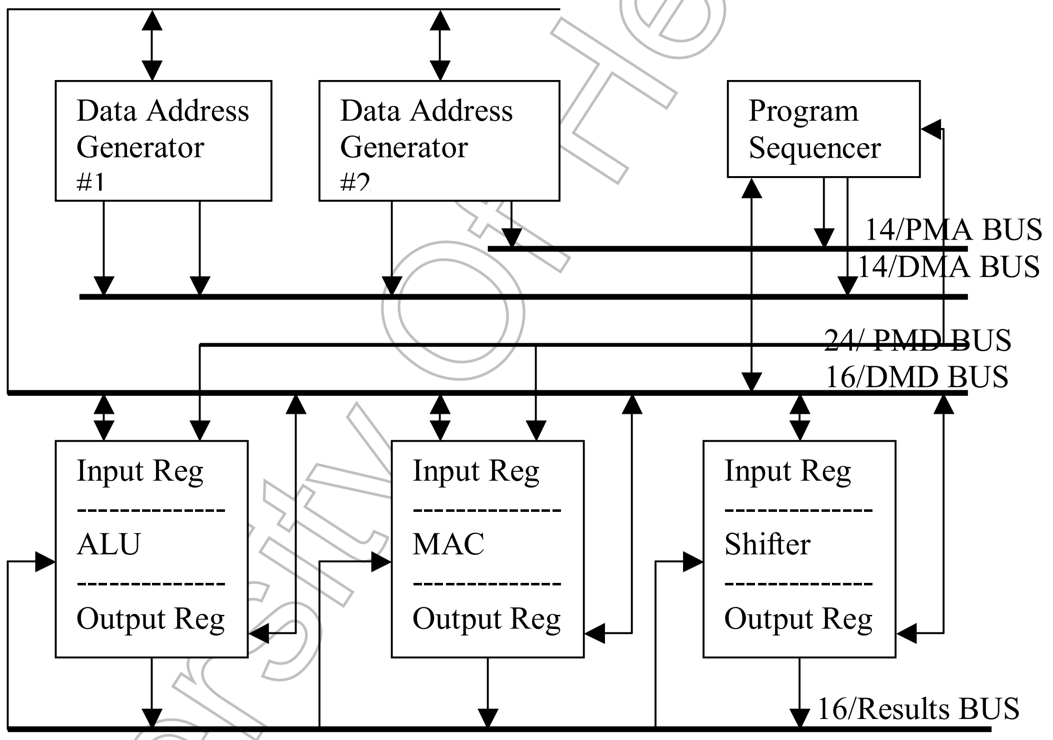


Figure 5 **ADSP-2100 Family Base Internal Architecture**

3.2.2 The Audio CODEC

The developing board includes an onboard full duplex, 16-bit stereo audio CODEC the AD1819A. It supports sampling frequencies of up to 48KHz and has two channel input and two channel output ports with an option at input to choose between Line Input and Microphone Input.

3.2.3 ADSP 21065L SHARC Key Features

The following bullet point sum-up the key features of the hardware used Ref[2,3].

- 198 MFLOPS (32-bit floating-point)
- 180 MOPS (32-bit fixed-point)
- 16K 32-bit Dual-ported on-chip memory (544 KBits configurable)
  64M x 32-bit word external address space
- Glueless SDRAM interface
- 2 serial transmit/receive ports (Tx/Rx) support 32-channel TDM
- I2S mode supports up to 8 channels
- Two timers with event capture and PWM options
- 12 programmable I/O pins
- 10 DMA channels
- Glueless 32-bit SDRAM Interface multiprocessing with 2 ADSP-21065Ls with synchronous data transfer rate
- 3.3Volt, 208-pin PQFP
- TDM interface
- Direct interface to T1/E1 lines
- 240 Mbytes/second external port/ Maximum data throughput
- Non-intrusive DMA Utilize full capabilities of core processor

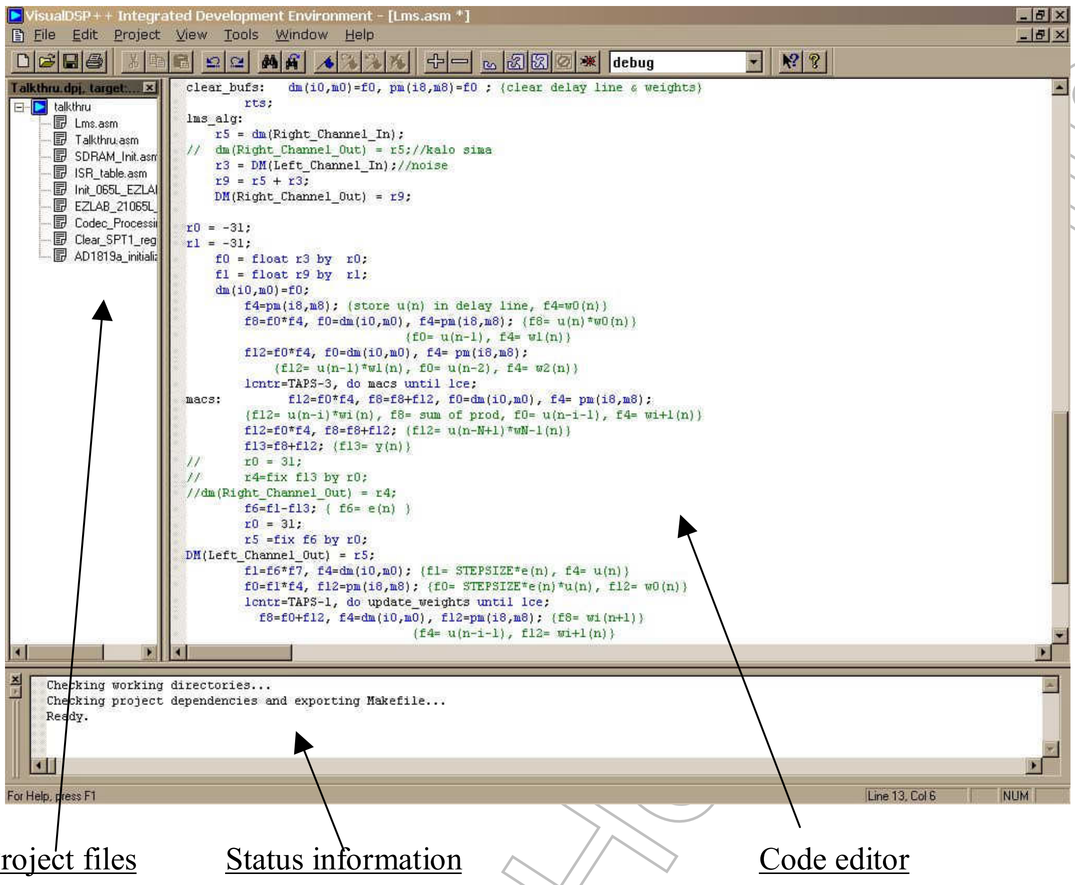
4.0 SOFTWARE

4.1 Chapter Overview

The following paragraphs describe the software used to write the code of the echo canceller. The author believes that a detailed analysis of the steps to install and run the software is beyond the scope of this report and suggests further reading from the installation and software manuals included in the package. Ref.[2,4].

4.2 VisualDSP Environment

In the ADSP 21065L SHARC EZ-LAB a CD-ROM was included containing a software program for developing DSP applications. The program's name was Visual DSP and is comprised of an Integrated Development Environment and a Debugger. The environment provided a project management utility containing some useful tools such as a C compiler, a run time library with over 100 Math, DSP and C runtime library routines, an assembler, linker, loader, simulator and EPROM Splitter. The software had a licence of 45 days after which you could only run it in the limited DEMO edition.



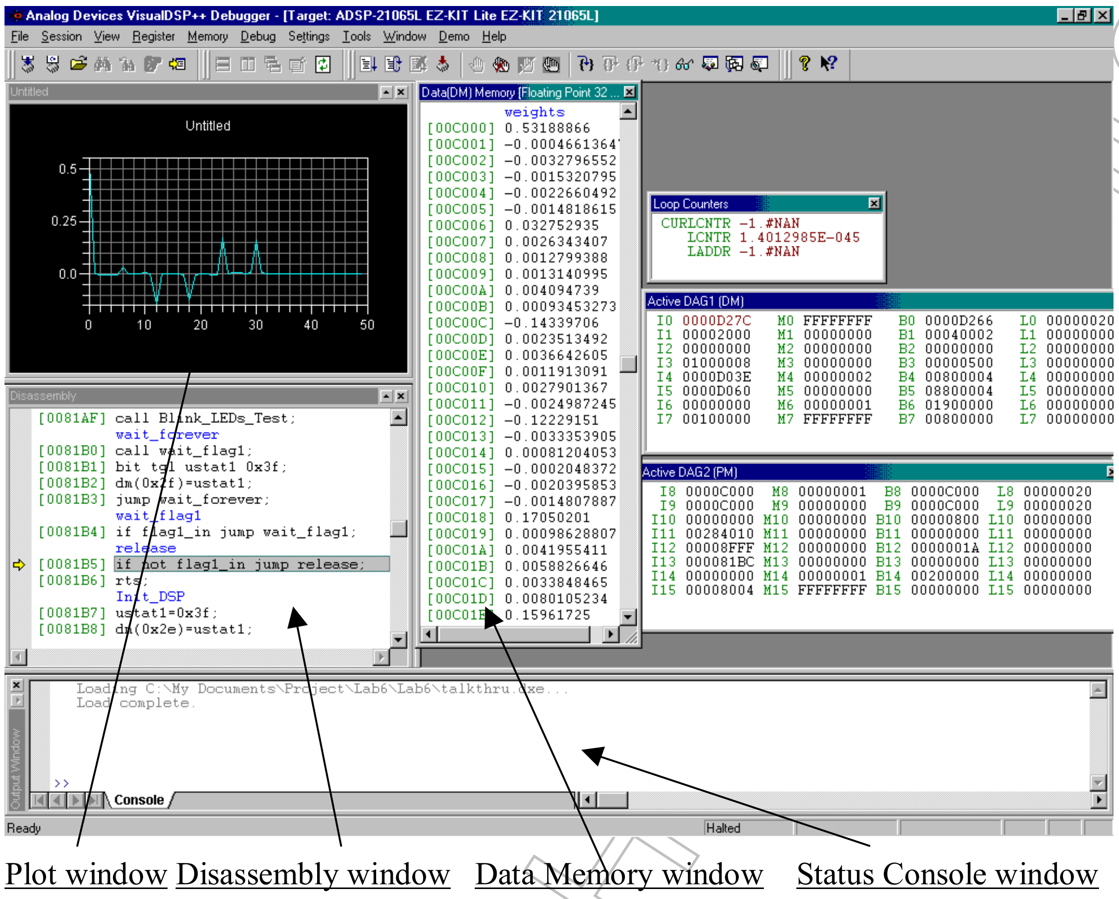
```
clear_bufs:   dm(i0,m0)=f0, pm(i8,m8)=f0 ; {clear delay line & weights}
        rts;
lms_alg:
    r5 = dm(Right_Channel_In);
//  dm(Right_Channel_Out) = r5;//kalo sima
    r3 = DM(Left_Channel_In);//noise
    r9 = r5 + r3;
    DM(Right_Channel_Out) = r9;

r0 = -31;
r1 = -31;
    f0 = float r3 by  r0;
    f1 = float r9 by  r1;
    dm(i0,m0)=f0;
        f4=pm(i8,m8); {store u(n) in delay line, f4=w0(n)}
        f8=f0*f4, f0=dm(i0,m0), f4=pm(i8,m8); {f8= u(n)*w0(n)}
                              {f0= u(n-1), f4= w1(n)}
        f12=f0*f4, f0=dm(i0,m0), f4= pm(i8,m8);
            {f12= u(n-1)*w1(n), f0= u(n-2), f4= w2(n)}
        lcntr=TAPS-3, do macs until lce;
macs:         f12=f0*f4, f8=f8+f12, f0=dm(i0,m0), f4= pm(i8,m8);
        {f12= u(n-i)*wi(n), f8= sum of prod, f0= u(n-i-1), f4= wi+1(n)}
        f12=f0*f4, f8=f8+f12; {f12= u(n-N+1)*wN-1(n)}
        f13=f8+f12; {f13= y(n)}
//      r0 = 31;
//      r4=fix f13 by r0;
//dm(Right_Channel_Out) = r4;
        f6=f1-f13; { f6= e(n) }
        r0 = 31;
        r5 =fix f6 by r0;
DM(Left_Channel_Out) = r5;
        f1=f6*f7, f4=dm(i0,m0); {f1= STEPSIZE*e(n), f4= u(n)}
        f0=f1*f4, f12=pm(i8,m8); {f0= STEPSIZE*e(n)*u(n), f12= w0(n)}
        lcntr=TAPS-1, do update_weights until lce;
          f8=f0+f12, f4=dm(i0,m0), f12=pm(i8,m8); {f8= wi(n+1)}
                                    {f4= u(n-i-1), f12= wi+1(n)}
```

```
Checking working directories...
Checking project dependencies and exporting Makefile...
Ready.
```

Figure 6 **This is a captured image of VisualDSP environment instance just after the talk-through program was opened.**

The above picture shows an instance of the Visual DSP environment. By building the project, VDSP checks all dependencies for the files included in the project and invokes the compiler which checks the code for improper declarations, invalid instructions and syntax errors, if all go well the linker is invoked next. The linker then "links" all the files together to produce a .dxe executable file. This file is loaded on to the DSP processor by the Debugger environment. An instance of which can be seen at the following image (Fig7).


Analog Devices VisualDSP++ Debugger - [Target: ADSP-21065L EZ-KIT Lite EZ-KIT 21065L]
File Session View Register Memory Debug Settings Tools Window Demo Help
Untitled
Untitled
0.5
0.25
0.0
0
10
20
30
40
50
Disassembly
Data(DM) Memory [Floating Point 32 ...
Loop Counters
Active DAG1 (DM)
Active DAG2 (PM)
Output Window
Console
Ready
Halted
Plot window
Disassembly window
Data Memory window
Status Console window

```
[0081AF] call Blink_LEDs_Test;
         wait_forever
[0081B0] call wait_flag1;
[0081B1] bit tgl ustat1 0x3f;
[0081B2] dm(0x2f)=ustat1;
[0081B3] jump wait_forever;
         wait_flag1
[0081B4] if flag1_in jump wait_flag1;
         release
[0081B5] if not flag1_in jump release;
[0081B6] rts;
         Init_DSP
[0081B7] ustat1=0x3f;
[0081B8] dm(0x2e)=ustat1;
```

```
         weights
[00C000] 0.53188866
[00C001] -0.0004661364
[00C002] -0.0032796552
[00C003] -0.0015320795
[00C004] -0.0022660492
[00C005] -0.0014818615
[00C006] 0.032752935
[00C007] 0.0026343407
[00C008] 0.0012799388
[00C009] 0.0013140995
[00C00A] 0.004094739
[00C00B] 0.00093453273
[00C00C] -0.14339706
[00C00D] 0.0023513492
[00C00E] 0.0036642605
[00C00F] 0.0011913091
[00C010] 0.0027901367
[00C011] -0.0024987245
[00C012] -0.12229151
[00C013] -0.0033353905
[00C014] 0.00081204053
[00C015] -0.0002048372
[00C016] -0.0020395853
[00C017] -0.0014807887
[00C018] 0.17050201
[00C019] 0.00098628807
[00C01A] 0.0041955411
[00C01B] 0.0058826646
[00C01C] 0.0033848465
[00C01D] 0.0080105234
[00C01E] 0.15961725
```

```
CURLCNTR -1.#NAN
   LCNTR 1.4012985E-045
   LADDR -1.#NAN
```

```
I0 0000D27C   M0 FFFFFFFF   B0 0000D266   L0 00000020
I1 00002000   M1 00000000   B1 00040002   L1 00000000
I2 00000000   M2 00000000   B2 00000000   L2 00000000
I3 01000008   M3 00000000   B3 00000500   L3 00000000
I4 0000D03E   M4 00000002   B4 00800004   L4 00000000
I5 0000D060   M5 00000000   B5 08800004   L5 00000000
I6 00000000   M6 00000001   B6 01900000   L6 00000000
I7 00100000   M7 FFFFFFFF   B7 00800000   L7 00000000
```

```
 I8 0000C000  M8 00000001  B8 0000C000  L8 00000020
 I9 0000C000  M9 00000000  B9 0000C000  L9 00000020
I10 00000000 M10 00000000 B10 00000800 L10 00000000
I11 00284010 M11 00000000 B11 00000000 L11 00000000
I12 00008FFF M12 00000000 B12 0000001A L12 00000000
I13 000081BC M13 00000000 B13 00000000 L13 00000000
I14 00000000 M14 00000001 B14 00200000 L14 00000000
I15 00008004 M15 FFFFFFFF B15 00000000 L15 00000000
```

```
Loading C:\My Documents\Project\Lab6\Lab6\talkthru.dxe...
Load complete.

>>
```

Figure 7 **This image of the debugger was captured while the echo canceller program was running on the DSP chip.**

This utility (debugger) allows the user to step by step analyse the code that runs on the DSP to find problems in coding logic. The debugger has all the necessary functions, such as step by step instruction execution, watch points, break points, disassembly window and many more useful features to examine the program and make the necessary corrections to make it work properly. If during the debugging procedure the user identifies some error and wants to correct them he has to go back to the editor make the changes in the code and re-compile, re-link and re-debug the program until it works as intended. The following diagram shows the main procedures involved (Fig.8).

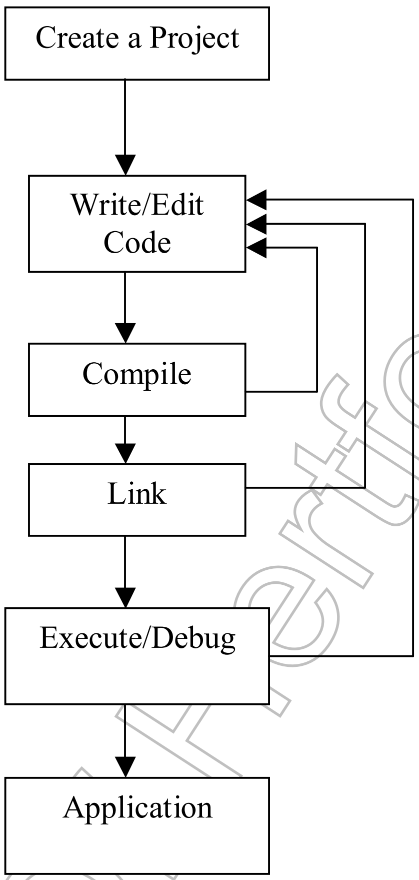


Figure 8 **Basic steps that lead to a working application software**

5.0 Adaptive Filtering

5.1 Chapter Overview

In this chapter the author explains some fundamental concepts governing adapting filters that is relevant to the echo canceller implementation project undertaken.

5.2 Adaptive Filters

As mentioned in the end of chapter 2 an adaptive filter is a digital filter whose characteristics change in an unknown environment input signal. Whether the adaptive filter is used in communication, sonar, radar, or image signal processing the basic feature remains the same. A set of adjustable coefficients, are changed through the error estimation performed by an algorithm, that has an input vector with a desired response. From this we can deduce the two basic parts of an adaptive filter:

- Adjustable coefficients
- Algorithm that updates (adjusts) the coefficients

There are many types of digital filters each performing best on specific tasks yet they are categorized either as Finite Impulse Response (FIR) filters, that as implied by the name have a finite length impulse response, or Infinite Impulse Response (IIR or recursive) filters, that has an impulse response of infinite length. Both types of filters can be used in the realization of the echo canceller but an FIR filter was chosen since it is simple and stable.

5.3 Transversal FIR

In the echo canceller implementation a transversal or tapped-delay line FIR filter was used since it is relatively easy to be implemented and provides stable and accurate performance. The diagram below (Fig.9) shows the realization of the filter:

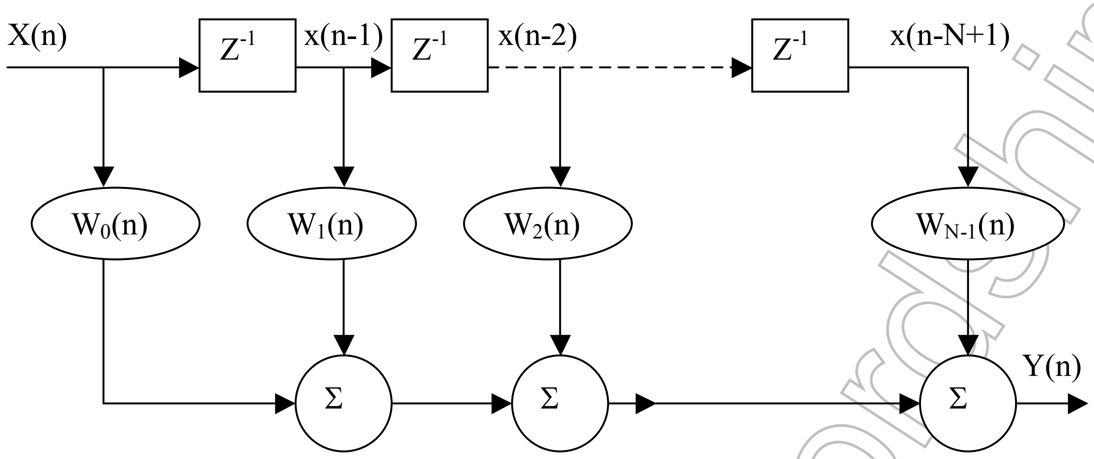


Figure 9 **A Transversal FIR filter implementation**

The squares with Z^{-1} in them denote the delay elements of the filter. The ovals with the W(n) starting at zero up to N-1 correspond to the weights of the filter or else coefficients. The circles are adders. The above diagram corresponds to the following difference equation. Ref.[5]:

$$Y(n) = \underline{W}(n)\underline{X}(n) = \sum_{i=0}^{N-1} W_i(n) X(n-i) \qquad \textbf{equation 2.}$$

Where $\underline{X}(n)$ is the input vector and $\underline{W}(n)$ the weights vector. N is the number of filter taps (coefficients), which give the length of its impulse response. For a sampling frequency of 8kHz the number of filter taps give the total echo path the system can process. Typically for N=64 an echo delay of 8ms can be fully processed. The diagram on the next page (Fig.10), illustrates the position of the echo canceller unit at the far end and near end positions in a communications system. In the project undertaken the number of taps utilized was 64 as a trade of between echo path and rapid filter convergence when turned on. In reality the filter performs satisfactorily in echo paths of up to 20ms.

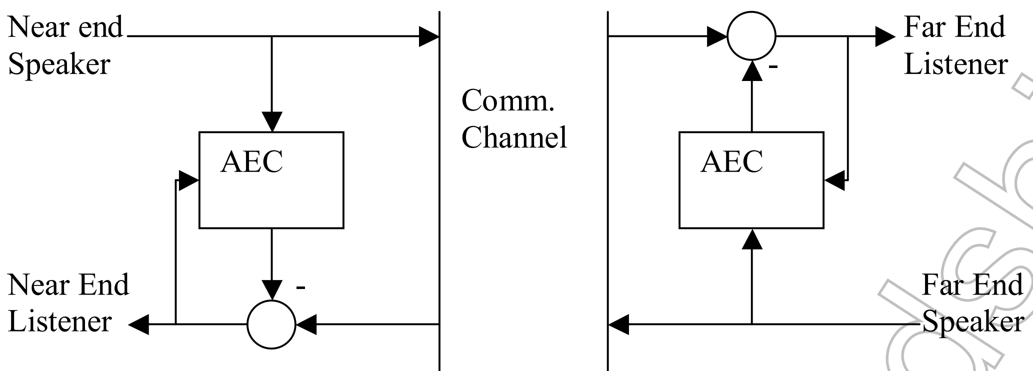


Figure 10 **A basic illustration of echo cancellers within a communications channel, to achive full-duplex echo cancellation.**

5.4 Adaptive Filter's Algorithms

There are many type of algorithms developed for use in adaptive digital filters LMS (Least Mean Square), RLS (Recursive Least Square, NLMS (Normalized Least Mean Square) and many more. Although not the best algorithm to utilize in an echo canceller most commercial applications favour the LMS. This is mainly because it is easy to implement and exhibits stability in performance. In this project the LMS algorithm was therefore implemented based on the same criteria. Although the NLMS was considered as a second algorithm implementation, unfortunately time restrictions did not allow for any implementations other than the LMS.

The LMS algorithm changes the filter coefficients based on the minimization of the square error of the system. As mentioned before the system cancels the echo present in the desired signal d(n) by predicting the echo Y(n) and subtracting it from the actual system output e(n). If the echo prediction Y(n) is not perfect then the subtraction result e(n) also gives the system error an constitutes of the output signal and the residual echo in it .

5.4.1 Mathematical Modelling

So we have:
The filter output:

$$Y(n) = \underline{W}(n)\underline{X}(n) = \sum_{i=0}^{N-1} W_i(n)X(n-i) \qquad \textbf{equation 3.}$$

the system output:

$e(n) = d(n) - Y(n)$ **equation 1.**

And the algorithm output

$\in = E[e^2(n)]$ **equation 4.**

which is the mean square error where E is the expectation operation.

By successive substitution of **equation 3.** to **1.** and **1.** to **4.** we yield the following equation:

$$\in = E\left[d^2(n)\right] + \underline{w}^T(n) R\, \underline{w}(n) - 2\, \underline{w}^T(n)\, \underline{p} \qquad \textbf{equation 5.}$$

Where, $R = E\left[x(n) X^T(n)\right]$, is the auto correlation matrix and

$\underline{p} = E\left[d(n)\underline{x}(n)\right]$, which is the cross correlation vector

By solving the following equation Ref[, which minimizes the mean square error, we can acquire the desired values for the filter coefficients.

$$\frac{\delta \in}{\delta\, \underline{w}(n)} = -2\,\underline{p} + 2RW = 0 =>$$

$$=> RW = \underline{p}$$

$W = [\ w_0\ w_1\ w_2 \ldots\ldots w_{N-1}]$

The above lead to the Wiener-Hopf equation

$\underline{w} = R^{-1}\, \underline{P}$ **equation 6.**

that gives the optimum weights for the filter.

Obtaining the filter weights this way is time consuming and unrealistic due to the calculation of the auto correlation matrix and the cross correlation vector. This is why he practical LMS used in implementations adapts the weights on a sample-by-sample basis. The formula involved is :

$\underline{w}(n+1) = \underline{w}(n) - u\, \underline{\nabla}(n)$ **equation 7.**

$\underline{w}(n)$ is the weight vector, u the adaptation step-size and, $\underline{\nabla}(n)$ the true gradient vectors at the nth sample. u the adaptation step-size also controls the rate of convergence and reliability of the system. By increasing the value of u the algorithm updates the filter weights by larger 'steps' which improves the system's response time but at the same time makes the weight values not so close to the optimal ones, therefore the system performs fast but not accurately. By decreasing the value of u the weights change slower but their values get closer to the optimal ones. This way the system converges slowly but is accurate. Typically 0<u<1 and for this implementation was 0.0025.

The true gradient $\underline{\nabla}(n)$ can be estimated by the $e^2(n)$ to be an estimate of the mean square error.

Thus

$$\underline{\nabla}(n) = \frac{\delta\left[e^2(n)\right]}{\delta\, \underline{w}(n)} = -2e(n)\underline{x}(n)$$

which if substituted in **equation 7.** yields the LMS algorithm

$$\underline{w}(n+1) = \underline{w}(n) + 2ue(n)\underline{x}(n)$$ **equation 8.**

6.0 The Adaptive Echo Canceller

6.1 Chapter Overview

In this chapter the actual implementation of the system is described by presenting the steps followed to achieve the end result of a working application. The code is not explained in detail since such a thing would be pointless. Only key points are addressed. The code provided in the appendix is full and the comments within are mostly self-explanatory.

6.2 First To Final Step

6.2.1 The Talk Through Example

Due to the authors lack of previous DSP application developments he deemed necessary to make a start from the very basic demo programs included in the Visual DSP software package. So a ready written project called Talk-Through was used as the basis for the development of the echo canceller. This project involved some files and a very simple program that initialised the sound CODEC on the developing board and enabled the user to input a voice signal by means of a microphone to the stereo input port of the developing board and listen to an output on speakers connected to the stereo line out port of the developing board. After a lot of experimentation with changing the code a better understanding of the procedures involved gradually was developed.

6.2.2 Fixed Coefficients FIR Filter

As a first target the implementation of a simple low FIR digital filter with fixed coefficients was set. Below is part of the implementation of the filter in assembly code given here as a code example. Ref.[6,9]. The filter loaded the coefficients from a text file into a circular buffer, took the samples of the input signal form the CODEC via a register (r2) and execute the filtering routine to provide an output of the form:

$$Y(n) = \underline{W}(n)\underline{X}(n) = \sum_{i=0}^{N-1} W_i(n)X(n-i)$$

Finite Impulse Response Filter

```
FIR:
        r12 = r12 xor r12, dm(i1,0) = r2;              //set r12=0,store input
                                                        //sample in line circular buffer
        r8 = r8 xor r8, f0 = dm(i0,0), f4 = pm(i8,m8);//set r8=0,get data,get
                                                        //coefficients from buffer
        lcntr = FIRLen-1, do macloop until lce;
macloop:
        f12 = f0*f4, f8 = f8+f12, f0 = dm(i1,m1), f4 = pm(i8,m8); //MAC
        rts (db);                                       //return from delayed subroutine
        f12 = f0*f4, f8 = f8+f12;                       //last multiplication
        f0=f8+f12;                                      //last accumulation
```

The coefficients for the filter were calculated with the help of a program (EZ-FIR) that automatically generated the desired filter weights by choosing from the following information:

- Filter Type
  Low pass, High pass, Band pass, Band stop, Hilbert transformer, Differentiator
- Window Type
  Rectangular, Triangular, Hanning, Hamming, Generalized Hamming, Kaiser-Bessel
- Filter Length
  1 to 256
- Sampling Frequency
- Cut-off Frequency
- Filter Gain

By using the EZ-FIR program and entering the following specifications:
Low pass filter,
Rectangular window,
64 taps filter length,
8000Hz sampling frequency,
3.5KHz cut-off frequency,
and a gain of 1,
a set of coefficients was generated and loaded to the buffer with index i8. The program was compiled, linked, loaded, run and monitored from the debugger. To see though if it really worked a test had to be performed.
The filter was tested using a signal generator providing the input signal at the line in mode of the board and an oscilloscope monitoring the line out of the board. The filter performed as expected with a deviation of approximately 400Hz below the specified cut-off value. A number of other filter types and specifications were also designed and implemented with the same FIR structure for the author to be even more familiarized with the code and DSP board.

6.2.3 The Noise Canceller

As the next stage of a successfully working fixed coefficients FIR filter it was decided to move on and implement a transversal FIR filter with LMS algorithm updating the coefficients. The algorithm, although not the most efficient of those existing nowadays, is widely used basically due to the fact that needs a small amount of memory, is relatively simple to implement and appears to be stable without providing any care. The core assembly for the task was found after some research in a file at the site of Analog Devices under the name lms_rls.zip (Ref[7]) . This file contained several assembly codes for testing well known adaptive filter algorithms. The initial file used in the project was called lms.asm and was written by Wassim G. Najm , from Analog Devices, DSP division. This code though needed modification to be incorporated to the already changed talk-through example program.
Since echo could be regarded as a case of having noise in a signal the system was biased towards an exhibition of noise cancelling properties. For this purpose the set-up of the equipment was as illustrated in figure 11:

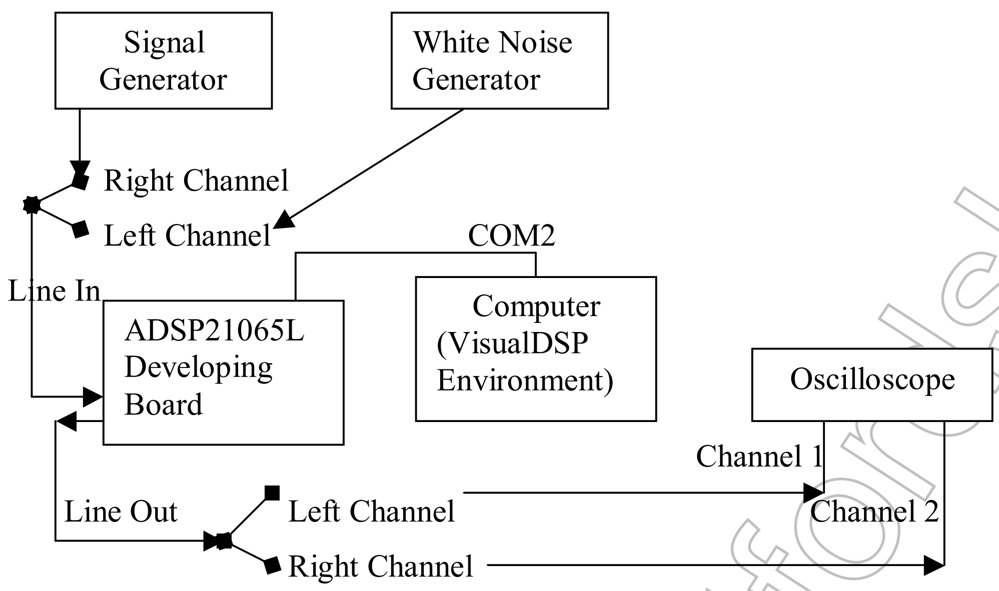


Figure 11 **General overview of instruments and connections between them while testing the system**

A signal generator was used to provide an input signal, with a sinusoidal frequency of 2KHz and peak-to-peak amplitude of 1Volt, at the Right Channel of the CODEC input.

A white noise signal generator was connected to the Left Channel of the CODEC input providing the noise into the system with maximum amplitude of 0.3Volts. A second signal generator instead of the noise generator was used while testing the system for echo cancellation response.

An oscilloscope was connected to the CODEC Line out port to monitor for the two (2) outputs.

The talk-through example program used 48KHz-sampling frequency. This frequency was changed to 8kHz by modifying the necessary variable in the AD1819a_initialization.asm file. This change in sampling frequency was done because the maximum frequency transmitted in voice telephone networks is 3.5KHz so a sampling frequency of 8Khz is adequate.

The Line-In port was preferred to that of the Microphone-In by also changing the necessary value in the AD1819a_initialization.asm file and also by setting the necessary jumpers on the developing board. The jumper pins are a set for each channel (Left, Right) as shown in figure 12 below:

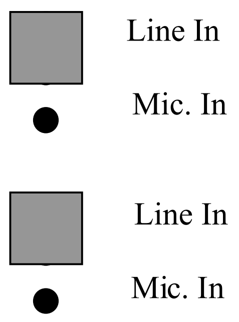


Figure 12 **The schematic above gives a picture of how the jumpers and pins look when selecting the Line Input. The square boxes above depict the plastic, sorting jumpers provided with the board.**

The task was to display the noisy input signal on one channel and the cleared output signal on the other. So the Left Channel of the Line Out was to output the system output and the Right channel of the Line Out was to output the input signal with the noise so that a comparison of those signals on the display of the oscilloscope could give an immediate impression whether the system works or not.

For easy access to the input and output channels and for provision to microphone use the use of six male and four female stereo jacks was devised. A cable used in stereo audio equipment such as the one used for stereo headphones was split in half so as to produce the following accessory illustrated in figure 13:

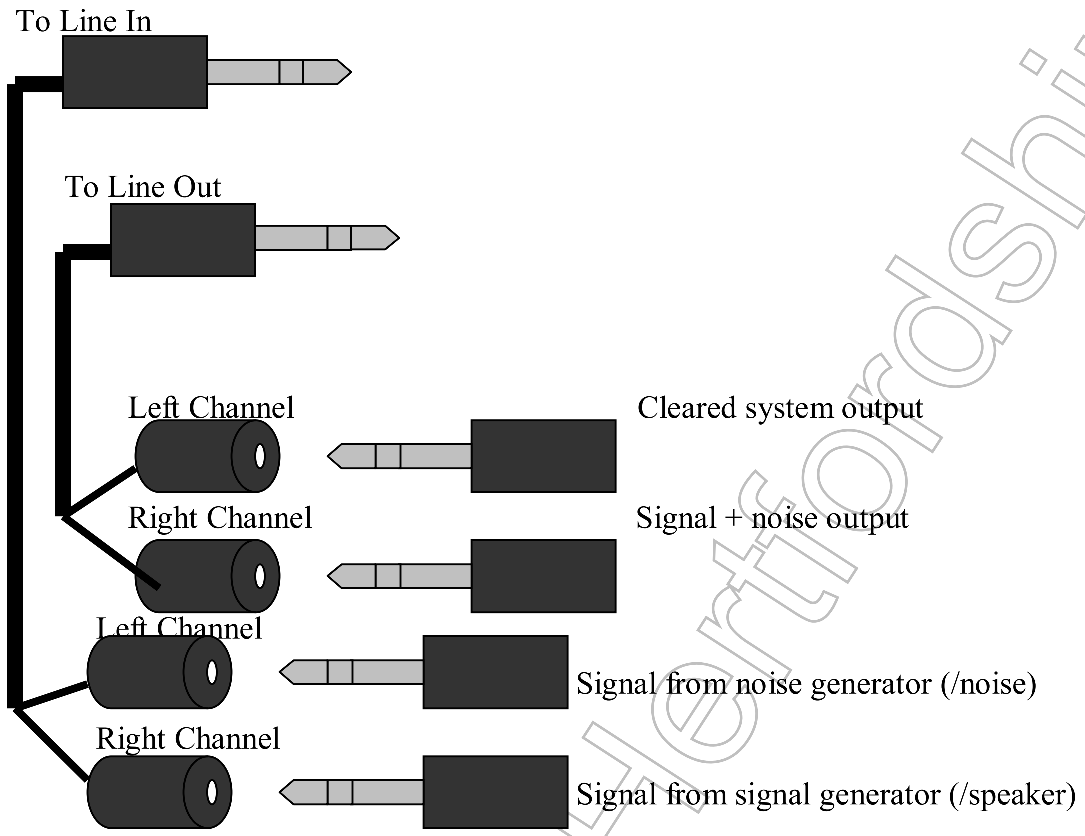


Figure 13 **An illustration of the custom made cables used in the project**

The noise (x(n)) plus the signal was generated by adding the clear signal to the noise signal with code:

```
r5 = dm(Right_Channel_In);
r3 = DM(Left_Channel_In);//noise
r9 = r5 + r3;
DM(Right_Channel_Out) = r9;
```

Register 5 holds the sample coming in from the CODEC's right channel
Register 3 holds the sample coming in from the CODEC's left channel
Register 9 holds the noisy sample
CODEC's right channel output is the contents of register 9
This way the oscilloscope displays the noisy signal we want to filter (d(n)).
At the same time the filter takes just the noise (x(n)) and by performing the computations gives its prediction Y(n).

```
r0 = -31;
r1 = -31;
          f0 = float r3 by  r0;
          f1 = float r9 by  r1;
          dm(i0,m0)=f0;
                    f4=pm(i8,m8); {store x(n) in delay line, f4=w0(n)}
                    f8=f0*f4, f0=dm(i0,m0), f4=pm(i8,m8); {f8= x(n)*w0(n)}
                                                          {f0= x(n-1), f4= w1(n)}
                    f12=f0*f4, f0=dm(i0,m0), f4= pm(i8,m8);
                              {f12= x(n-1)*w1(n), f0= x(n-2), f4= w2(n)}
                    lcntr=TAPS-3, do macs until lce;
macs:                 f12=f0*f4, f8=f8+f12, f0=dm(i0,m0), f4= pm(i8,m8);
                    {f12= x(n-i)*wi(n), f8= sum of prod, f0= x(n-i-1), f4= wi+1(n)}
                    f12=f0*f4, f8=f8+f12; {f12= x(n-N+1)*wN-1(n)}
                    f13=f8+f12; {f13= y(n)}
```

(The brackets contain the comments that explain the individual code line.)

This prediction is subtracted by the noisy signal d(n) and this yields the prediction error and system output e(n)

e(n)=d(n)-y(n) **equation 1.**

```
f6=f1-f13; { f6= e(n) }
```

Where floating-point register f1 holds the noisy signal sample from register r9

This error is processed by the LMS routine to do the necessary updating of the coefficients (Wi(n)).

Wi(n+1)=Wi(n)+2ue(n)x(n-i)

Where 2u is the value of the STEPSIZE variable.

```
#defineSTEPSIZE      0.005
```

```
            f1=f6*f7, f4=dm(i0,m0); {f1= STEPSIZE*e(n), f4= x(n)}
                  f0=f1*f4, f12=pm(i8,m8); {f0= STEPSIZE*e(n)*x(n), f12= w0(n)}
                  lcntr=TAPS-1, do update_weights until lce;
                    f8=f0+f12, f4=dm(i0,m0), f12=pm(i8,m8); {f8= wi(n+1)}
                                                    {f4= x(n-i-1), f12= wi+1(n)}
update_weights:   f0=f1*f4, pm(i9,m8)=f8; {f0= STEPSIZE*e(n)*x(n-i-1)}
                                              {store wi(n+1)}
                  rts(db);
                  f8=f0+f12, f0=dm(i0,1); {f8= wN-1(n+1)}
                                              {i0 -> x(n+1) location in delay line}
                  pm(i9,m8)=f8; {store wN-1(n+1)}
```

The whole procedure, as obvious from studying the code, repeats itself every time a new set of samples arrives from the receive port of the CODEC.

Circular Buffering

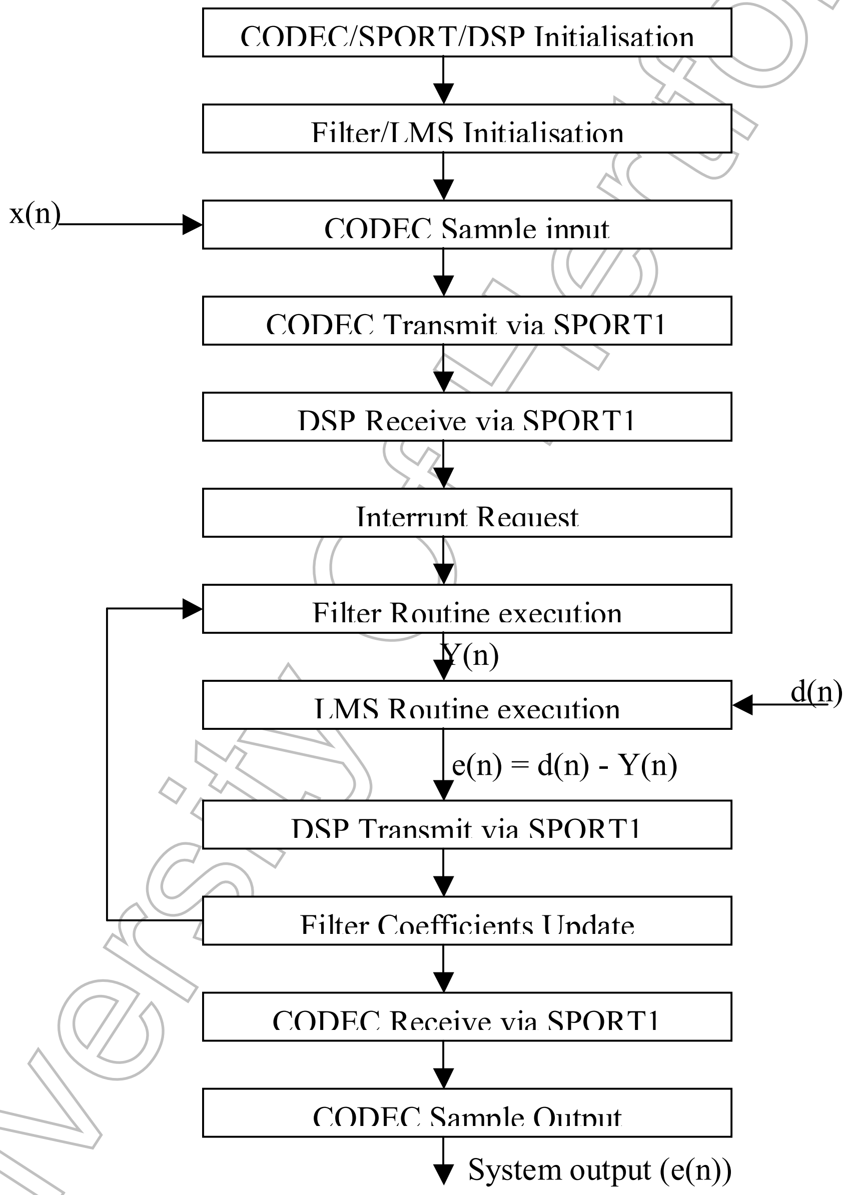
Circular buffers are nothing more than normal buffers with the advantage that after reaching the end of its length it jumps back to the beginning holding one by one the new values overwriting the existing ones automatically. This kind of buffer was used to hold the input data as well as the filter coefficients. The data buffer used was **deline_data.** The coefficient buffer was **weights.** DAG 1 was used to generate the delay line data buffer and DAG 2 to create the filter weights buffer.

The initialisation of the filter and algorithm was as follows:

```
lms_init:
                  b0=deline_data;
                  m0=-1;
                  l0=TAPS; {circular delay line buffer}
                  b8=weights;
                  b9=b8;
                  m8=1;
                  l8=TAPS; {circular weight buffer}
```

```
        l9=l8;
                f7=STEPSIZE;
                f0=0.0;
                lcntr=TAPS, do clear_bufs until lce;
clear_bufs:      dm(i0,m0)=f0, pm(i8,m8)=f0 ; {clear delay line & weights}
                rts;
```

System Process Overview



Further details can be found in APPENDIX II in the program source code. The author believes that the general concept of the design is more important than the code since somebody else might try to implement this system with another language. As it is known, for example, the ADSP 21065L in combination with VisualDSP support the development of applications written in ANSI C.

7.0 CONCLUSIONS

The echo canceller system developed in this project and presented in this report, proved to work satisfactorily and exceptionally good as a noise canceller as well. The noise canceller was the first system developed, since the echo can be considered as a case of noise with spectrum characteristics close to those of the echo creator signal. The system utilized the Analog Devices ADSP-21065L EZ-LAB development board with the ADSP-21065L DSP processor on board and the AD1819a sound CODEC. The CODEC provided for dual channel input and output at 16-bits precision. The echo canceller was based on a simple and reliable system design of an adaptive filter that predicted the echo in the input signal so as to perform a subtraction of the prediction from the input signal to provide a clear output signal. The adaptive filter that was used in this project was a transversal FIR (Finite Impulse Response) filter with the LMS Least Mean Square) algorithm handling the adaptation of the filter coefficients. This type of popular filter was chosen for its simplicity and stability and because it is widely used on adaptive filter designs for sound applications world-wide. The algorithm was chosen to be the LMS since it is popular among the choices of 'entry-level' adaptive filter designers and because it is easy to implement and monitor. Its performance may not be the best but is satisfactory. The NLMS (Normalized Least Mean Square) was considered as a second algorithm implementation, but due to time restrictions the thought was discarded.

One aspect of the echo canceller that failed to reach expectations was its use in satellite communications. This is due to the echo path length in such a system that exceeds 400ms. Some solutions to this problem are mentioned in chapter 8. The echo canceller system in its current form, cancels noise exceptionally good and echo with up to 10ms delay.

APPENDIX I includes the result graphs obtained while monitoring the performance of the system and APPENDIX II the program source code documented.

8.0 FURTHER DEVELOPMENT

There are numerous activities that could better the performance of the current echo canceller. The most important in the authors opinion are:

- Code optimization

Testing its efficiency and omitting parts that may not be necessary could optimise the code of the system. That is because many core part were imported from examples and applications of different nature.

- Active channel detection

A key feature of an AEC algorithm is active-channel detection. Indeed, when the far end operator is silent and the near-end operator is talking, the filter must not be adapted because the near-end operator is no longer an echo.

- Double-Talk Detection

During a DT condition, the near-end signal on the microphone contains echo and near-end speech**.** The residual error used to update the filter coefficients includes the near-end speech, and if the algorithm is still adapting, the algorithm may start to diverge. To cure this problem the filter coefficients must not be updated during that time.

- Use of another more efficient algorithm

The use of an algorithm such as the normalized least mean square NLMS for example could improve the system.

- Echo Delay Estimator

The echo canceller developed in this project task was not tested to verify its ability to cancel echo that results from long distance communications such as in a satellite system. In such a system echo paths of 500ms are typical and some times exceed even 800ms. For an echo path of 500ms to be represented by the echo path length of an adaptive filter 4000 taps are needed if sampling is performed at 8KHz. This large number of filter coefficients takes up a lot of time to calculate and update for a DSP such as the one utilized in this project. To overcome such a problem an echo delay estimator may prove critical since it could start the process of adaptation and echo cancellation from the moment the echo signal has almost reached the near end so that it is cancelled, as if its delay was smaller.

- An amplifier for use of microphones with the board

An amplifier with a gain of 10 is necessary for using microphones with the board, since an odd thing happened when the on board amplification capability of the board was used. Only the right input channel was amplified. So a simple circuit is provided as an external microphone amplifier. The below schematic was designed and tested in PSPICE and is an operational amplifier with a gain of 10.

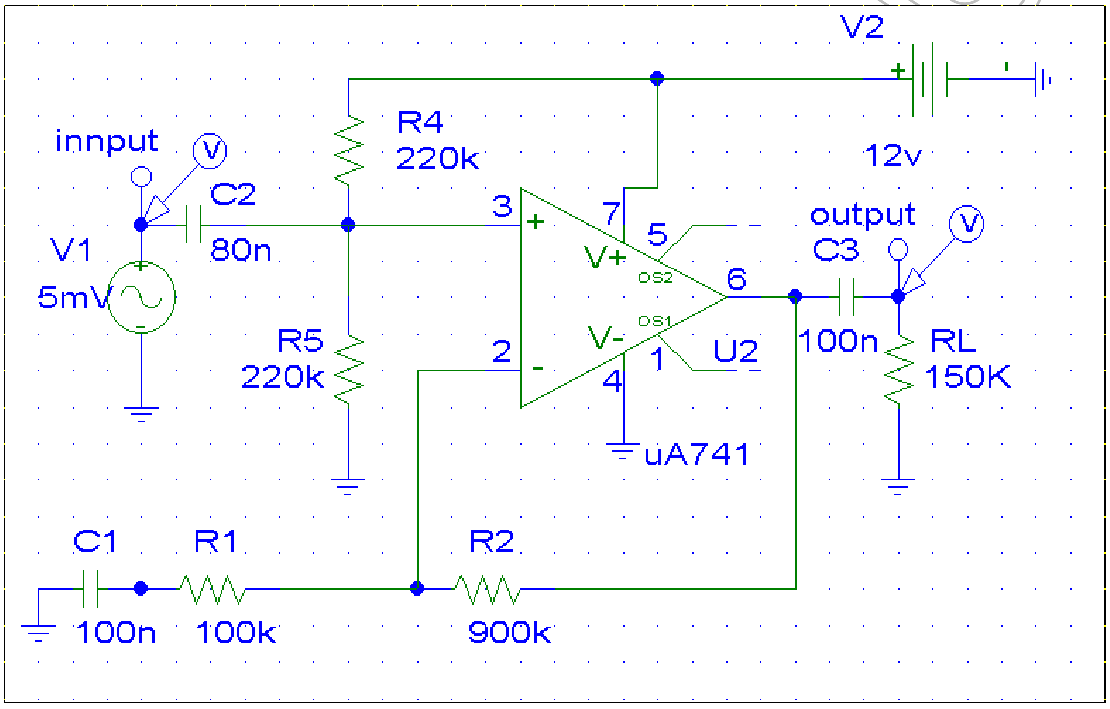


Figure 14 **Schematic of an operational amplifier with a gain of 10**

Op-amp circuit:

We want a gain of 10 therefore we have

10 = Vo/Vi = 1+ R2/R1 =>

⇨ R2 = 9R1

so if we choose R1 to be 100K then R2 must be 900K.

The values of R4 and R5 = 220K to split the voltage from 12 to 6 Volts .

For calculating the values of the capacitors we use the following formula

f = (2RC)^-1

Total offset voltage:

Vo(offset) = VIO(R1+R2)/R1 + IIOR2 = 6x10^-3(1000x10^3)/100x10^3 + 200x10^-9x900x10^3 = 0.06 + 0.18 = 0.24 Volts

The transient analysis was carried out only for the frequency of 1KHz. The characteristic graph is shown in figure 15 below

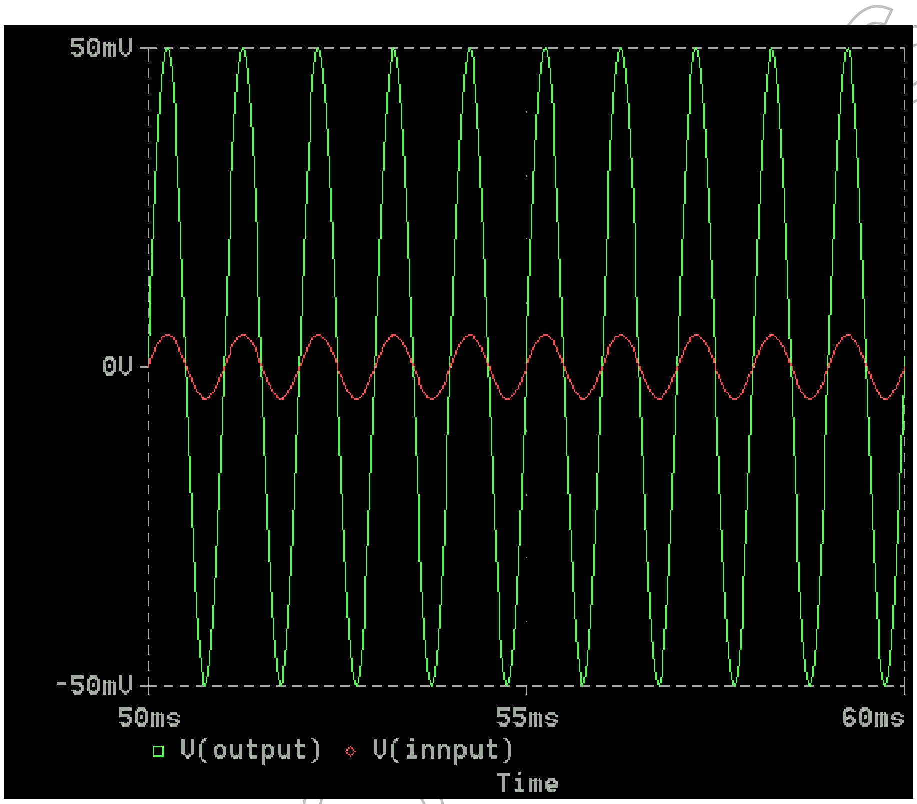


Figure 15 **This is the characteristic graph generated by PSPICE transient analysis of the op-amp circuit.The green line corresponts to the output signal while the red to the input signal.**

9.0 References

[1] Kondoz, M. A. , (1994) Digital Speech, Coding for Low Bit Rate Communication Systems (page 330-335). Surrey: John Wiley & Sons

[2] Owens J. F., (1993) Signal Processing of Speech. London: The Macmillan Press LTD

[3] Analog Devices (1999) ADSP-21065L EZ-LAB Development System Manual (page 1-6). Shrivenham: technology International (Europe)Limited

[4] http://products.analog.com/products_html/list_gen_157_2_1.html

[5] Analog Devices (1999) Getting Started guide. Norwood,USA: Analog Devices, Inc. Computer Products Division One technology Way

[6] http://www.analog.com/support/Design_Support.html

[7] http://www.analog.com/techsupt/application_notes/AD1819A_21065L.pdf

[8] http://www.analog.com/techsupt/software/dsp/app_note/21k_books.html

[9] Installation path of VisualDSP\21k\EZ-KITs\21065L\Demos\Tt.dpj

[10] Installation path of VisualDSP\21k\EZ-KITs\21065L\Demos

10.0 Bibliography

- Kondoz, M. A. , (1994) Digital Speech, Coding for Low Bit Rate Communication Systems. Surrey: John Wiley & Sons

- Owens J. F., (1993) Signal Processing of Speech. London: The Macmillan Press LTD

- Pree H. William, Teukolsky A. Saul, Vetterling T. William, Flannery P. Brian, (1992) Numerical Recipes in C, The art of scientific computing, Second Edition. Cambridge: Cambridge University Press

- Analog Devices (1999) Getting Started guide. Norwood, USA: Analog Devices, Inc. Computer Products Division One technology Way

- Analog Devices (1999) ADSP-21065L EZ-LAB Development System Manual (page 1-6). Shrivenham: technology International (Europe)Limited

- Analog Devices (1998) ADSP-21065L SHARC Technical Reference. Norwood, USA: Analog Devices, Inc. Computer Products Division One technology Way

- Analog Devices (1998) ADSP-21065L SHARC User’s Manual. Norwood, USA: Analog Devices, Inc. Computer Products Division One technology Way

APPENDIX I

Result graphs

The result graphs where obtained from a digital oscilloscope with persistence of 1 second. The grey areas show the various positions the signals took within a 1 second time period. The bold black lines depict the signals positions at the moment of print request. The part of the graph with the 1 at the left side denotes channel 1 of the oscilloscope, while with the 2 at the left side denotes channel 2 of the oscilloscope.

The following graph displays the predicted noise signal corresponding to the adaptive filter output Y(n) on channel 1 of the oscilloscope and the noise signal x(n) from the noise generator on channel 2 of the oscilloscope.

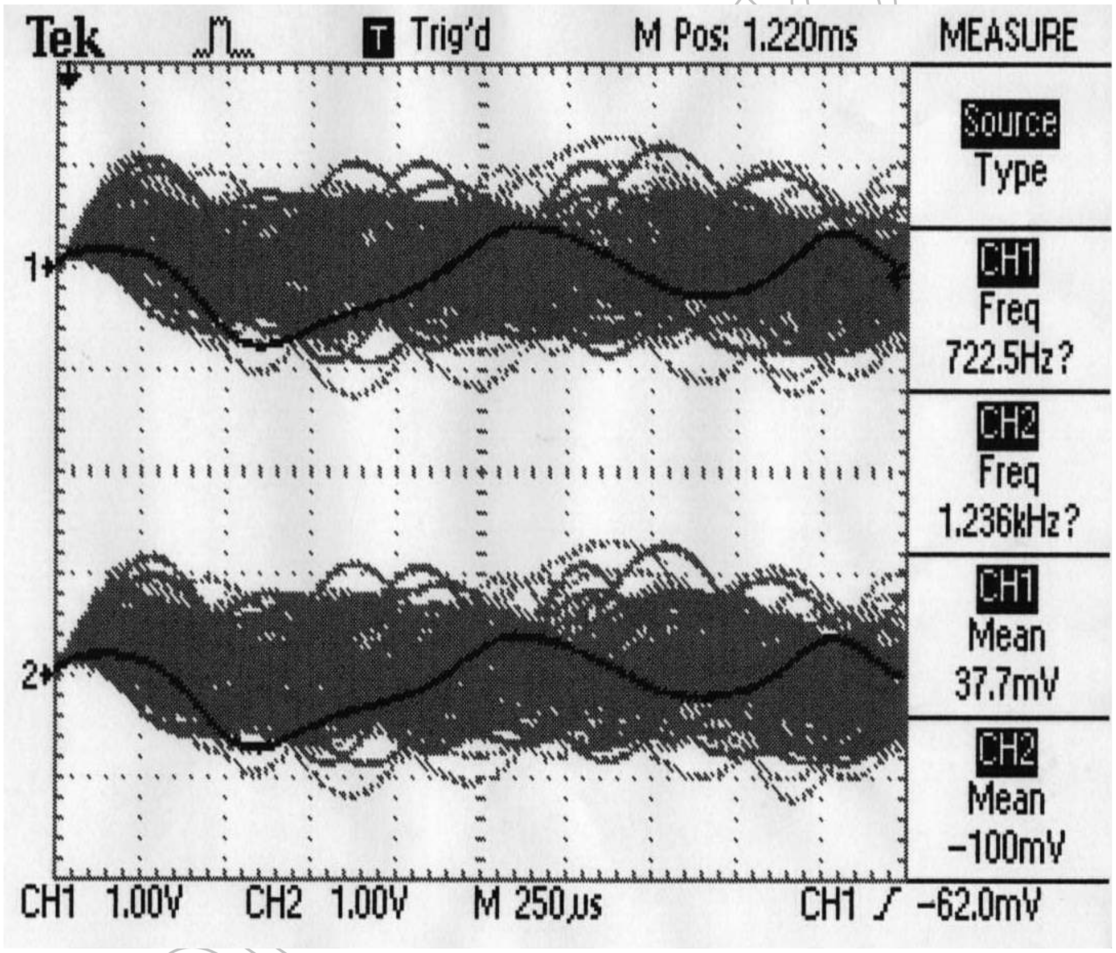


Figure 16 **Channel 1 displays the predicted noise Y(n). Channel 2 displays the actual noise x(n).**

The following graph displays the pure sinusoidal signal from the signal generator on channel 1 of the oscilloscope, while on channel 2 of the oscilloscope the noisy (noise + sine wave) system input d(n).

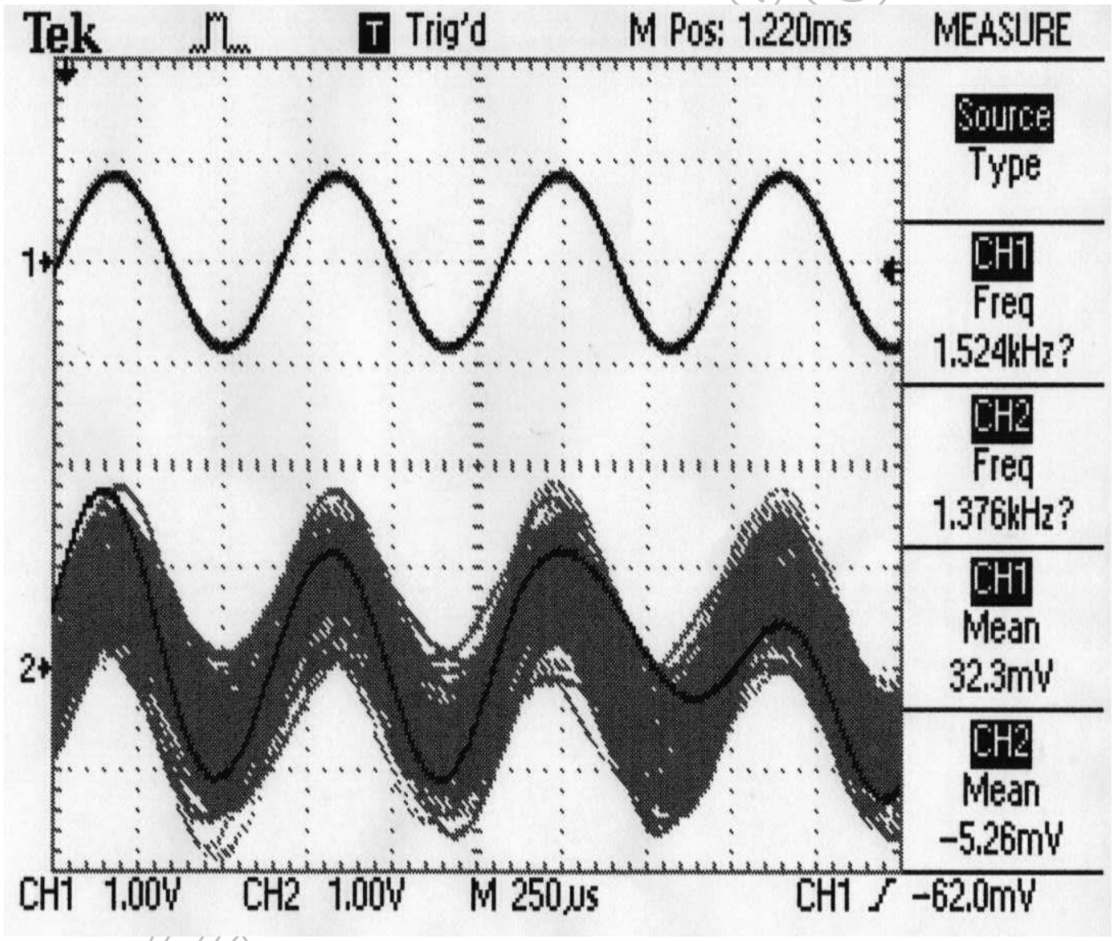


Figure 17 **Channel 1 displays the sine-wave from the signal generator and channel 2 the same sine-wave with the added noise (d(n)). This signal was used as the system input d(n).**

The following two graphs show the system output in contrast with the system input. The noisy system input is a perfect sinusoidal waveform signal from a signal generator plus a white (Gaussian) noise signal from a noise generator. Channel 1 (top part of graph) shows the noisy input signal d(n) while channel 2 (bottom part of graph) shows the system output e(n) = d(n) - Y(n). The cancellation of noise is obvious. Ideally the system output should have been a perfect sinusoidal wave. The difference between the graphs is in the time scale used, 250μs for the first 2.50ms for the second.

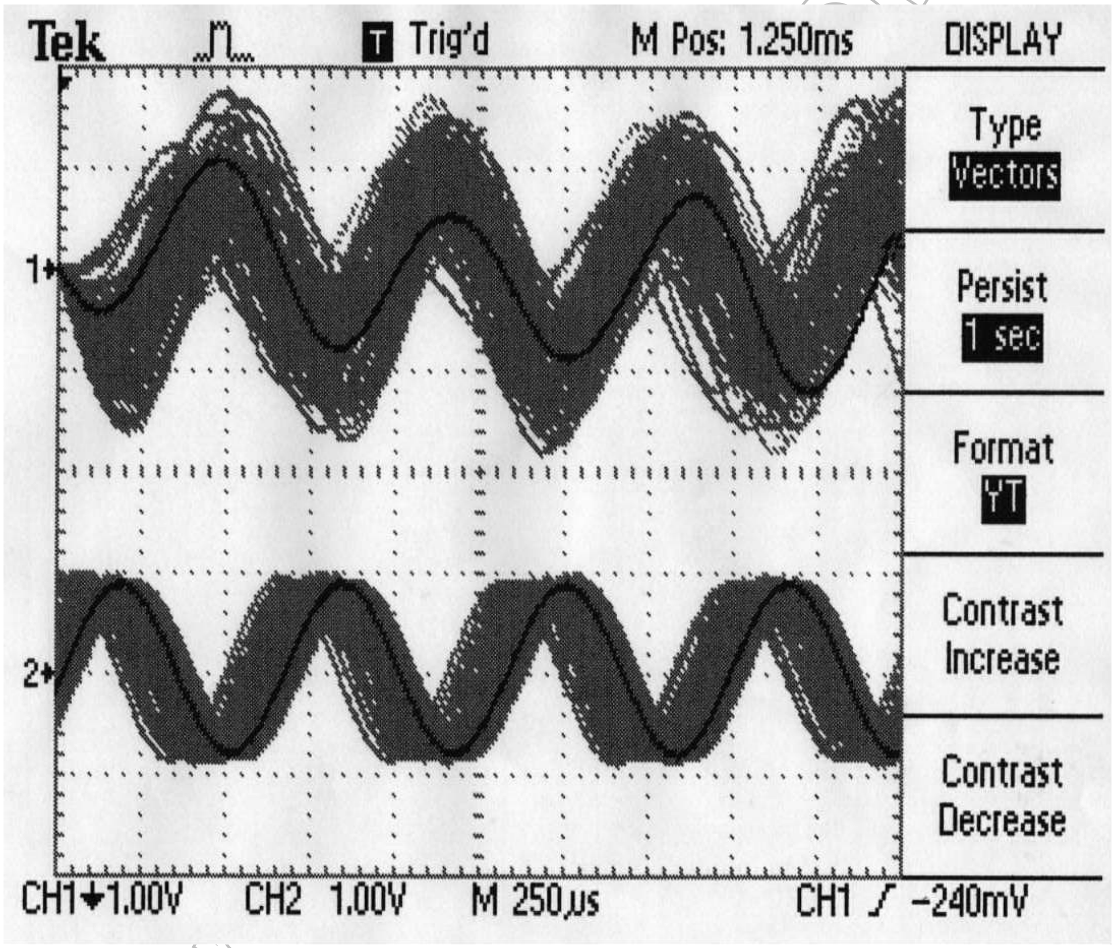


Figure 18 **Channel 1 displays the noisy system input (d(n)), while channel 2 displays the system output e(n). Ideally this should be a pure sine wave.**

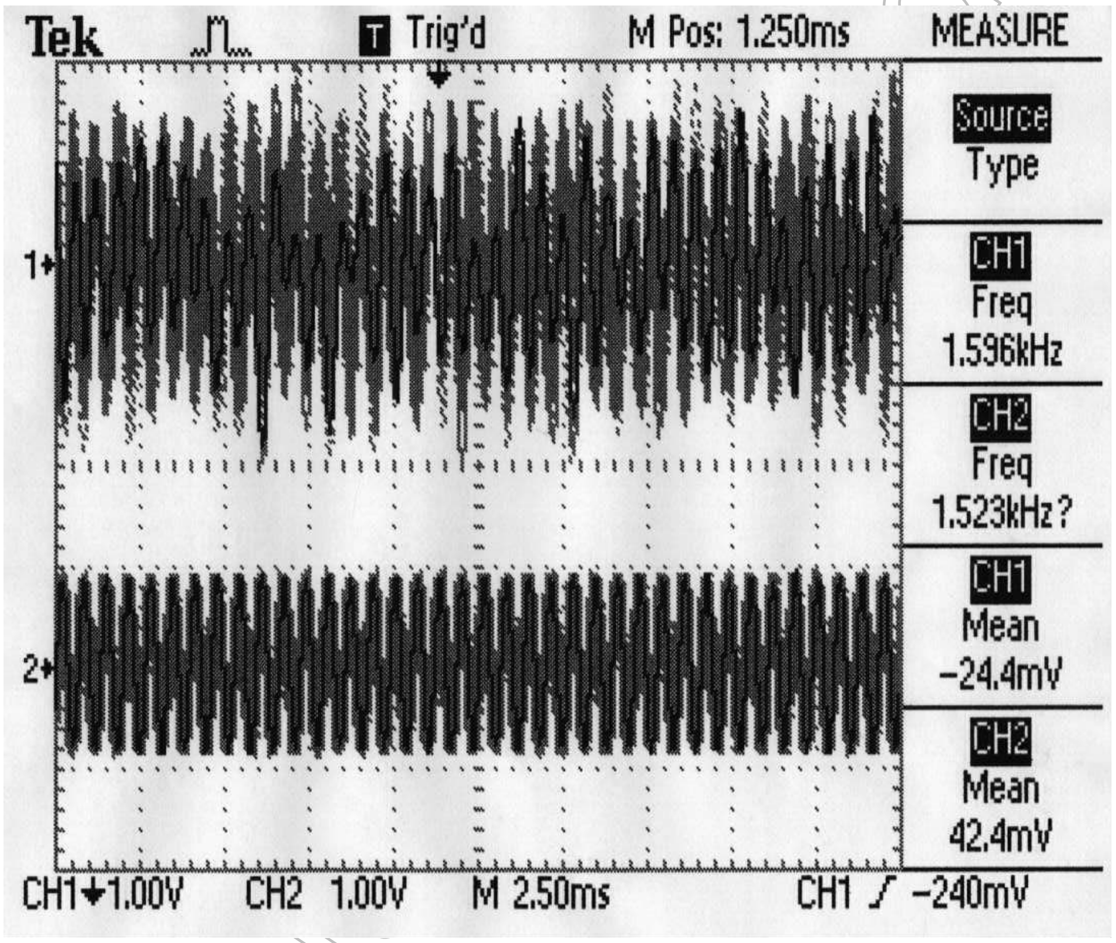


Figure 19 **Channel 1 displays the noisy system input (d(n)), while channel 2 displays the system output e(n). Ideally this should be a pure sine wave.**

The following two graphs display on channel 1 of the oscilloscope the ideal sinusoidal output of the system, while on channel 2 the actual system output. It must be noted that the first graph printout was generated while the coefficients start to adapt from their initial values (zero), while the second graph printout was generated after allowing some time for the coefficients to adapt.

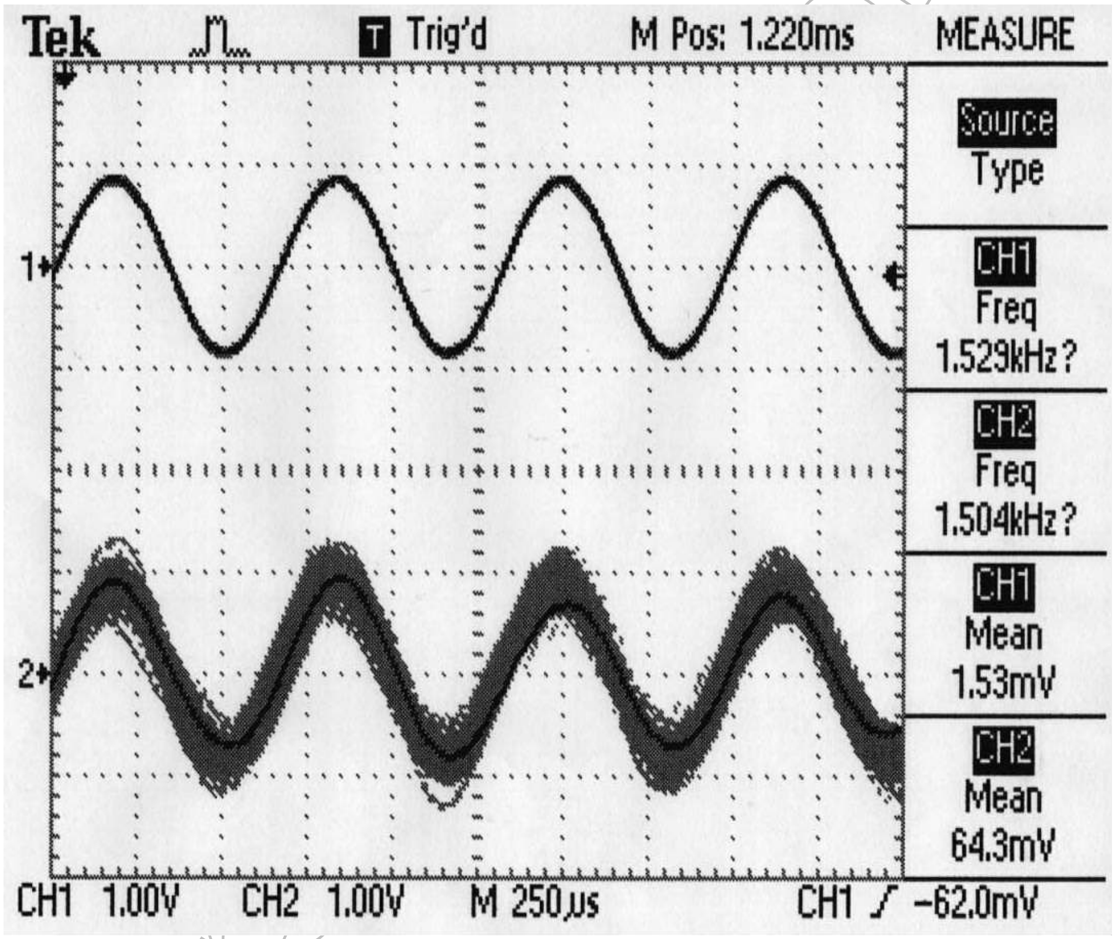


Figure 20 **Channel 1 displays the ideal system output and channel 2 the actual system output just at the beginning of the adaptation procedure.**

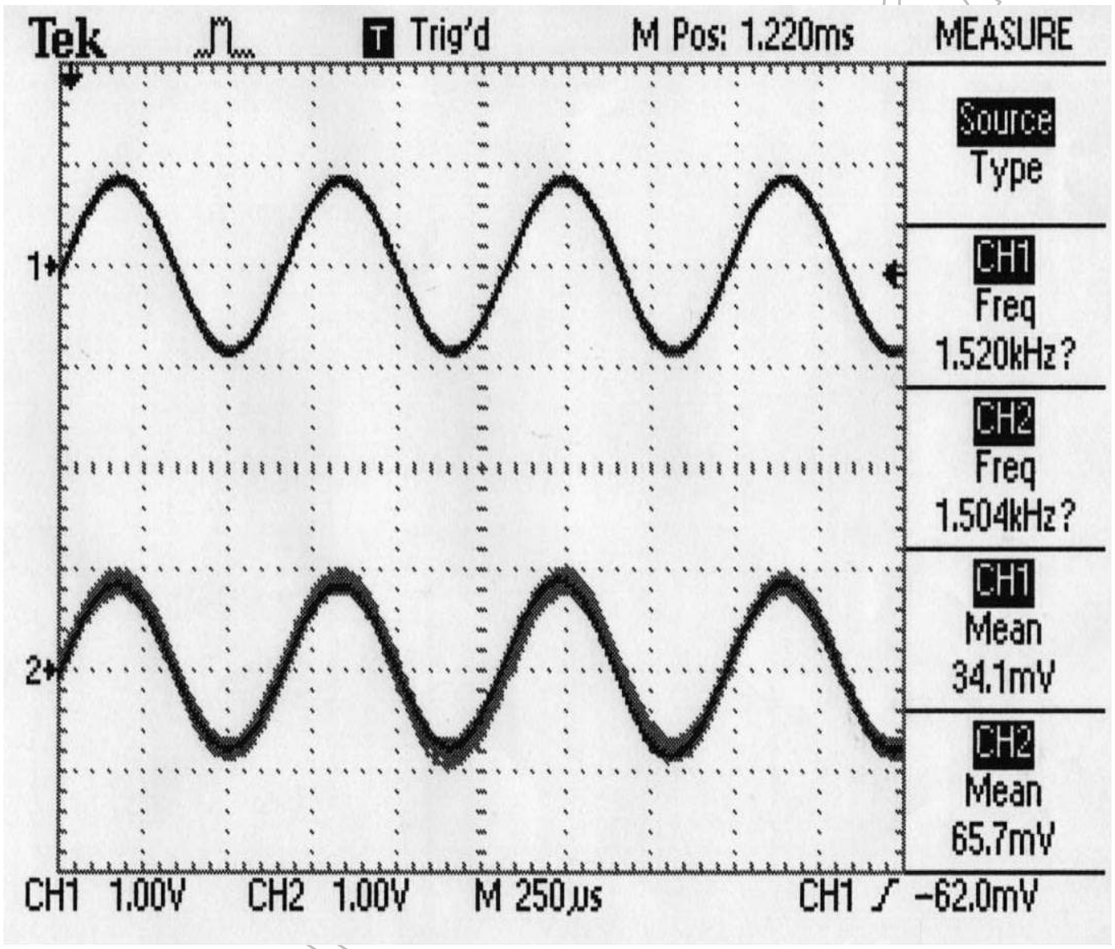


Figure 21 **Channel 1 displays the ideal system output and channel 2 the actual system output after the adaptation procedure advanced for some time (approximately 5 seconds).**

The following two graphs display the ideal output signal subtracted from the actual system output, thus giving us the residual noise. Ideally if the system adapted perfectly this should be a flat line. The difference between the graphs is in the time scale used, 250μs for the first 2.50ms for the second.

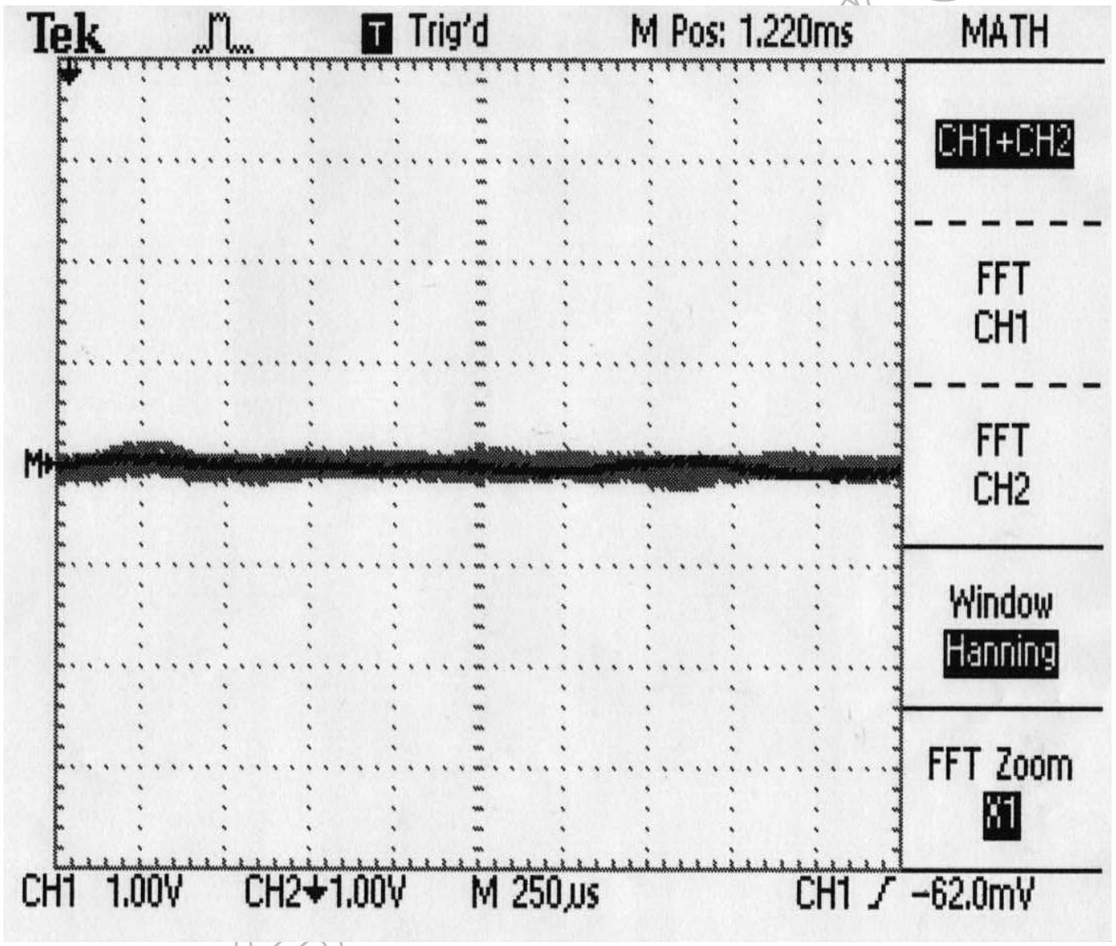


Figure 22 **Actual system output minus ideal system output.**

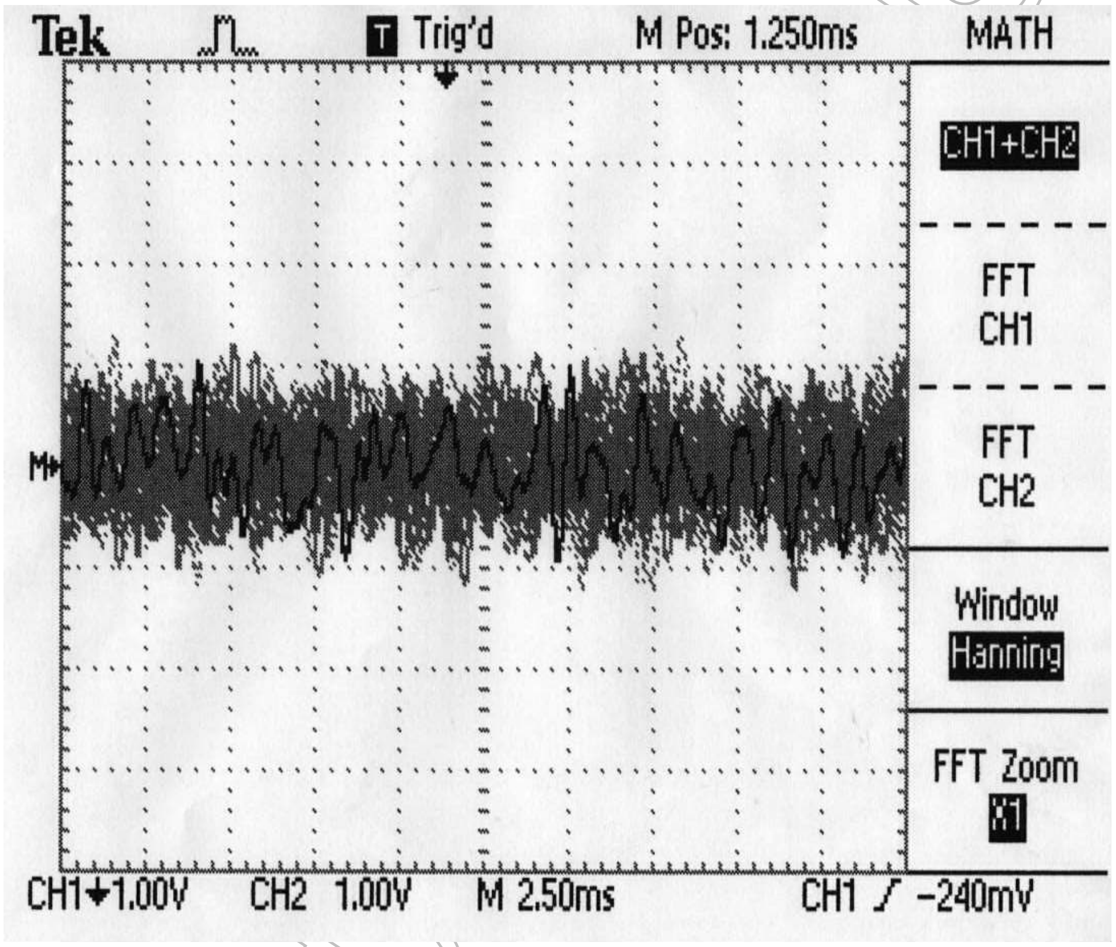


Figure 23 **Actual system output minus ideal system output.**

APPENDIX II

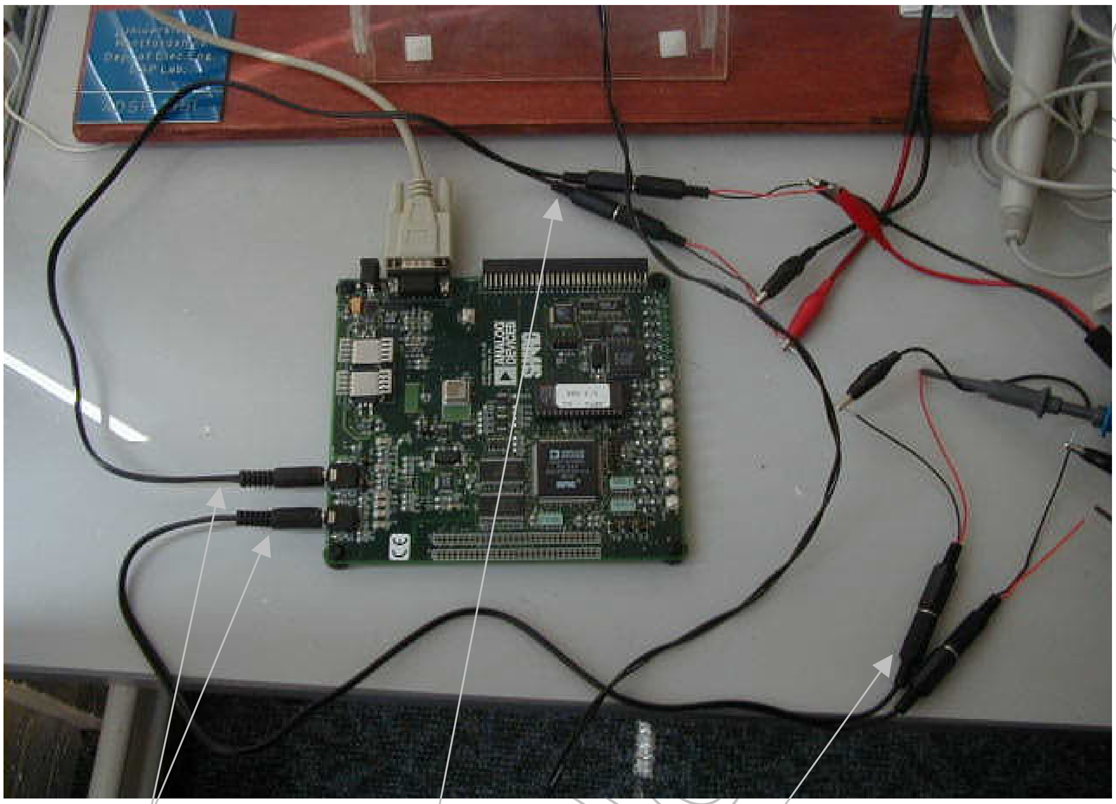
Program Code

This appendix contains the assembly code necessary to create an echo canceller using the Analog Devices ADSP 21065L Digital Signal Processor. It was written compiled and executed with VisualDSP software under windows 98 environment.

The files included in the order presented are:

- new65Ldefs.h
- def21065l.h
- EZLAB_21065L_debugger.ldf
- Lms.asm
- Talkthru.asm
- SDRAM_Init.asm
- Clear_SPT1_regs.asm
- AD1819a_initialization.asm
- Codec_Processing_ISR.asm
- Init_065L_EZLAB.asm
- ISR_table.asm

These files along with examples used and the report itself are included in the CD-ROM provided with the report. The following page has a couple of pictures of the DSP board used in the project.



Board under test!