Sudan University of Science and Technology

College of Graduate Studies

كلية الدراسات العليا

Designing of Real Time Voice Scrambler / Descrambler Based on Microcontroller

تصميم مشفر/ مفكك شفرة الصوت في الوقت الحقيقي بإستخدام متحكم دقيق

Thesis Submitted in Partial Fulfillment of the Requirement for the Degree of M.Sc. in Electronics Engineering (Computer and network Engineering)

Prepared By:

MOHAMMED SAAD DAOUD SULIMAN

Supervisor:

DR. AHMED ABDALLA MOHAMMED

May 2021

Dedication

To My Beloved Mother and Father

Acknowledgement

First and Foremost, I have to thank my research supervisor Dr. Ahmed Abdullah. Without his assistance and dedicated involvement in every step throughout the process, this research would have never been accomplished. I would like to thank you very much for your support and understanding

.

Abstract

Most of voice channels used in voice terminals such as mobile, telephone exchange are not ciphered. This makes them vulnerable to eavesdropping and hacking. Present day securing voice communication became an urgent need for civil and military applications and especially for real time communication. This work presents a real time microcontroller-based voice scrambling and descrambling (full duplex) using 16KHz sampling frequency. Voice scrambling / descrambling was done in frequency domain and time domain. In frequency domain, a frequency inversion is implemented using sinusoid wave multiplication and 8-order low pass Finite Impulse Response filter. In time domain, a segment permutation technique was implemented. When inputting, voice samples were stored in memory sequentially. However, when playing the voice, samples were read with a random sequence according to a look-up-table. The scrambling / descrambling algorithms were verified using MATLAB and implemented using ARDUINO DUE microcontroller. The system was tested in real time by scrambling the voice input from a microphone and output it through a speaker. Tests show that the scrambled voice was not recognizable at all. Also, the descrambled voice shows acceptable similarity with the original voice.

المستخلص

معظم القنوات المستخدمة في الاتصالات الصوتية غير مؤمنة مما يجعل الصوت عرضة للاختراق واستراق السمع.في الوقت الحاضر اصبح من الضروري المحافظة على سرية الاتصالات الصوتية في التطبيقات العسكرية والمدنية وخصوصا الاتصالات في الوقت الفعلي.هذا المشروع يقدم آلية لتشفير وفك تشفير الصوت في الوقت الحقيقي باستخدام معالج دقيق بتردد اخذ عينات 16 كيلوهيرتز وبالاتجاهين (الارسال والاستقبال) . في نطاق التردد ، نفذت آلية لعكس الترددات بضرب الاشارة بموجة جيبية وباستخدام مرشح تمرير منخفض ذو استجابة نبضية محددة من الدرجة الثامنة. في نطاق الزمن ، استخدمت تقنية تبديل المقاطع . عند ادخال الصوت تحفظ العينات في الذاكرة تسلسليا. لكن عند عرض الصوت تُقرأ العينات عشوائيا وفقا لجدول بحث. آليات وخوارزميات التشفير وفك التشفير تم التحقق منها باستخدام برنامج الماتلاب . و تم تنفيذها باستخدام معالج دقيق اردينيو ديو.اختبر النظام بإدخال صوت في زمن حقيقي من خلال مايكرفون واخراج الصوت المشفر من خلال سماعة . الاختبارات وضحت ان الصوت المشفر كان غير قابل للتمييز على الاطلاق. كما ان الصوت بعد فك تشفيره يبدو مشابها للصوت الاصلي بشكل مقبول.

Table Of Contetns

List of Figures

List of Abbreviations

Abbreviation	Meaning
ADC	Analog To Digital
BPSK	Binary Phase Shift Keying
BUF	Buffer
DAC	Digital To Analog
DFT	Digital Fourier Transform
DHT	Discrete Hartley Transform
DMA	Direct Memory Access
DSP	Digital Signal Processing
FFT	Fast Fourier Transform
FHSS	Frequency Hopping Spread Spectrum
FIR	Finite Impulse Response
FS	Sampling Frequency
FSK	Frequency Shift Keying
FT	Fourier Transform
GPSK	Generalized Pre-Shared Key
GSM	Global System For Mobile Communications
HT	Hartley Transform
IFFT	Inverse Fast Fourier Transform
IIR	Infinite Impulse Response
ISR	Interrupt service routine
LPF	Low Pass Filter
MIC	Microphone
NVIC	Configurable nested interrupt controller
OFDM	Orthogonal Frequency Division Multiplexing
PDC	Peripheral direct control
PC	Personal Computer
PESQ	Perceptual Evaluation Of Speech Quality
PSTN	Public Switched Telephone Network
RTS	Reversed Time Segmentation
SRAM	Static Random Access Memory
TSP	Time Segment Permutation
VOIP	Voice Over Internet Protocol

CHAPTER ONE

INTRODUCTION

CHAPTER ONE

INTRODUCTION

1.1 Preface

Present day securing voice communication became an urgent need for civil and military application and especially for real time communication, a scrambler is a device that transposes or inverts signals or otherwise encodes a message at the sender's side to make the message unintelligible at a receiver not equipped with an appropriately set descrambling device.

The security could be achieved by analog method referred as scrambling. The characteristics of the original signal are modified in time and frequency domain according to a given code to produce a scrambled channel to be transmitted. While in reception the scrambled channel received have to be descrambled to produce the original channel. The scrambled channel has to be transmitted on voice channel bandwidth (0—4KHz).

The security could be achieved with digital method called encryption which is done on digital value of the signal with different cyphering techniques. Although, very high levels of security could be achieved but a high bandwidth is needed and imposes many restrictions in real time implementation such as stream cipher.

1.2 Problem Statement

Most of voice channels used in voice connections such as mobile, telephone exchange are not ciphered. This Make them vulnerable to eavesdropping and hacking.

There is a need to guarantee end-to-end security for speech in real time communication systems such as GSM, Voice Over Internet Protocol, Telephone and analog Radio.

1.3 Objectives

The main objective of this work is to design, test and implement a real time microcontroller-based voice scrambling / descrambling system in frequency domain and time domain. Keeping the voice band without any change (0-4KHz).

1.4 Methodology

There are two domains for voice scrambling time domain and frequency domain. In frequency domain a frequency inversion with variable modulation frequency while in time domain the voice is divided into frames which are written in the memory sequentially while reading from memory randomly to output the scrambled Voice. The number of frames depends on microcontroller memory. To achieve real time operation a microcontroller used for scrambling and descrambling.

Analysis and feasibility study of chosen algorithm to be implemented in microcontroller. Analysis and estimation of execution time for each algorithm to be executed by the microcontroller taken in to consideration the type, clock of microcontroller, Sampling frequency and the memory needed to be used.

Matlab implementation for the algorithms, Simulation and execution of the algorithms in Matlab or C# to check the validity, Robustness with different parameters used before the decision about the final set of parameters.

Simulation Check for the algorithms with recorded voice and then real voice.

Design and implementation of the algorithms using microcontroller. Verification, Validation of system with testing the system functions against the different conditions.

1.5 Thesis outline

This research consists of six chapters. Chapter One introduces the project, problem statement, solution, objectives, and methodology of the research. In chapter two the common scrambling techniques were introduced with a theoretical background and literature review in the area. In Chapter three the system design and simulation steps were discussed. In Chapter four there is an overview of the implementation and verification steps. Chapter five illustrates and discusses tests, verification results after implementing the system using microcontroller. The last chapter outlines the main conclusions and gives recommendations for future work.

CHAPTER TWO

BACKGROUND AND LITERATURE REVIEW

CHAPTER TWO

BACKGROUND AND LITERATUREREVIEW

2.1 Introduction

This chapter introduces the background theory and literature related to the research conducted as well as the theoretical background of the research.

2.2 Theoretical Background

Distinguishing Analog Scrambling from digital encryption determining whether a specific device uses analog scrambling or digital encryption can be difficult and confusing.

The key to distinguishing between the two is the technique used to secure the voice. If the technique used to secure the voice only involves some manipulation of the time or frequency characteristics of the speech signal, the device uses analog scrambling. If the technique used to secure the voice involves encrypting digitized voice with a conventional cipher algorithm to yield digital cipher text, the device uses digital encryption. [2]

The real time analog scrambling techniques implemented in this work are.

- Frequency inversion
- Time permutation

2.3 Techniques Used in Voice Scrambling

2.3.1 Frequency Scrambling

Most common technique used in frequency scrambling is based on changing the frequencies of the signal in a certain way.

This section provides a review of different design methodologies for voice scrambling in audio band with emphasizing on real time implementation.

2.3.1.1 Fast Fourier Transform

Fast Fourier transform is used to calculate the frequency components of the signal, the analog transform is defined by

$$X(w) = \int_{-\infty}^{\infty} x(t)e^{-jwt}dt \qquad (2.1)$$

Where X (w) is frequency domain of the signal

The digital Fourier transform (DFT) is defined by

$$X(k) = Nc_k = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi kn}{N}} \; , k = 0,1,\dots,N-1 \qquad (2.2)$$

Where X (k) is digital frequency component of the signal

The frequency resolution (the space between X (K) and X (K+1) is given by

$$F_{reslution} = \frac{F_S}{N} \; , F_S = Sampling\ Frequency \qquad (2.3)$$

For voice signal with F_S =8192Hz, N=1024, the frequency resolution = 8Hz

Noting that DFT require N*N complex multiplication (for N=1024, number of multiplication = 1,048,576) which is so difficult to be used in real time implementation

To reduce the number of multiplication fast Fourier transform algorithm (FFT) is used. Which reduce the number of multiplications to

$$\text{Complex multiplication of FFT} = \frac{N}{2}\log_2 N \qquad (2.4)$$

For N=1024, the number of complex multiplications = 3,465. The general block diagram for voice scrambler and descramble as shown in figure (2.1).

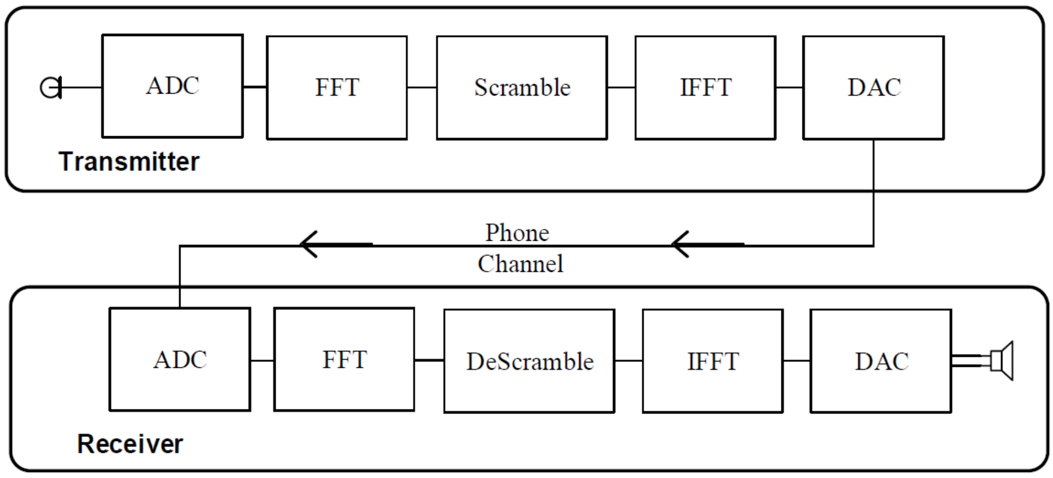


Figure (2.1) Block diagram of FFT voice scrambler and descrambler [5]

The basic idea of scramble block is the permutation of FFT filters randomly or in a given sequence as defined in look up table.The descramble block is the inverse permutation to get back the original sequence for FFT filters. [5]

The main points for the real time implementation are

- The need for complex multiplication (Floating point) which implies the use of dedicated DSP chip.
- The number of FFT points to be used which introduce a time delay in scrambling and descrambling channels in addition of frequency resolution (sampling frequency /number of points).

- For sampling frequency 8000Hz the processing time for scrambling and descrambling have to less than 125 microseconds.

2.3.1.2 Hartley Transform

The Hartley transform (HT) is an integral transform closely related to the Fourier transform (FT), but which transforms real-valued functions to real-valued functions. It was proposed as an alternative to the Fourier transform. The Hartley transform has the advantages of transforming real functions to real functions (as opposed to requiring complex numbers) and of being its own inverse.[11]

The Discrete Hartley Transform (DHT)

$$H_h(k\Omega_n) = \frac{1}{\sqrt{N}} \sum_{n-0}^{N-1} h(nT)\, cas(k\Omega_n nT) \qquad (2.5)$$

$$h(nT) == \frac{1}{\sqrt{N}} \sum_{k-0}^{N-1} H_h(k\Omega_n)\, cas(k\Omega_n nT) \qquad (2.6)$$

$$\Omega_n = \frac{2\pi}{NT} = frequency\ resolution\ (rad\ s^{-1}) \qquad (2.7)$$

Where

$$cas(wt) = cos(wt) + sin(wt) \qquad (2.8)$$

As compared with Fourier transform, DHT avoids complex arithmetic and requires half the memory storage because it requires N $\log_2 N$ real operation instead of complex operation for DFT.

Hartley transform need a real multiplication and not complex multiplication as compared with FFT.

2.3.1.3 Bank of Filters

The basic idea for this technique is to separate the voice frequency band in to different sub bands as shown in figure (2.2) below.

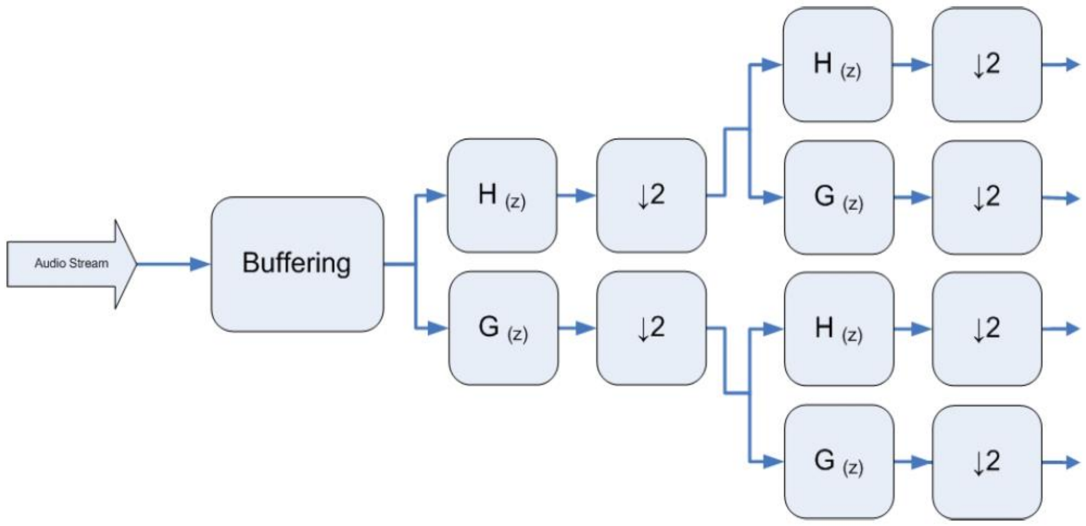


Figure (2.2) Bank of filers technique [7]

Where H (z) high pass filter, G(z) low pass filter, and (/2) down sample by 2 (reduce sample frequency by 2) For voice example with sampling 8000Hz, the first stage low-pass filter.

$F_{cut} = 1800Hz, F_{stop} = 2000Hz$

and high- pass filter

$F_{cut} = 1900Hz, F_{pass} = 2100Hz$

For the second stage sample frequency = 4000Hz

Low pass filter

$F_{cut} = 900Hz, F_{stop} = 1100Hz$

High pass filter

$F_{cut} = 1100Hz, F_{pass} = 1300Hz$

And so on

For scrambling, the position of the filter are changed as in the figure (2.3) below.

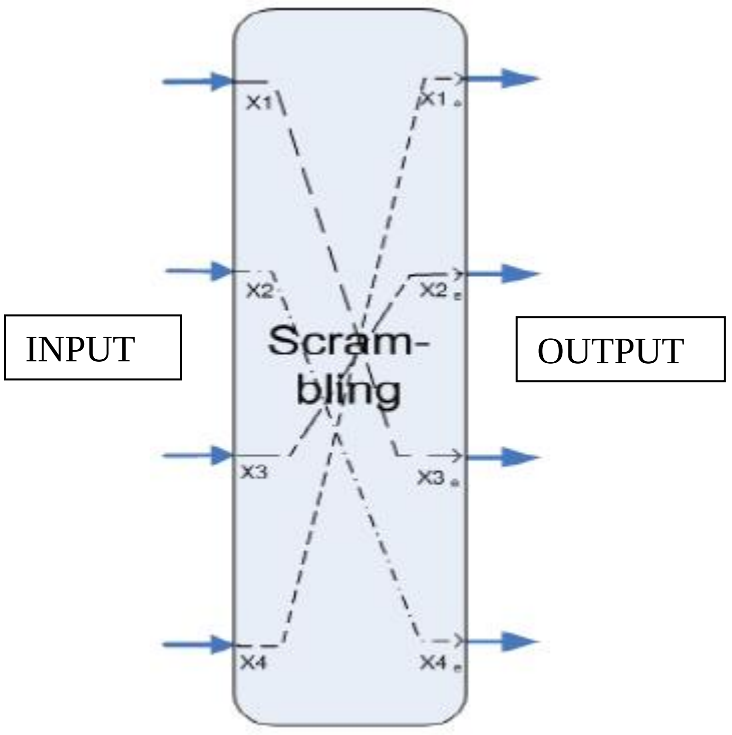


Figure (2.3) Scrambling block

The reconstruction of scrambled channel as in figure (2.4).

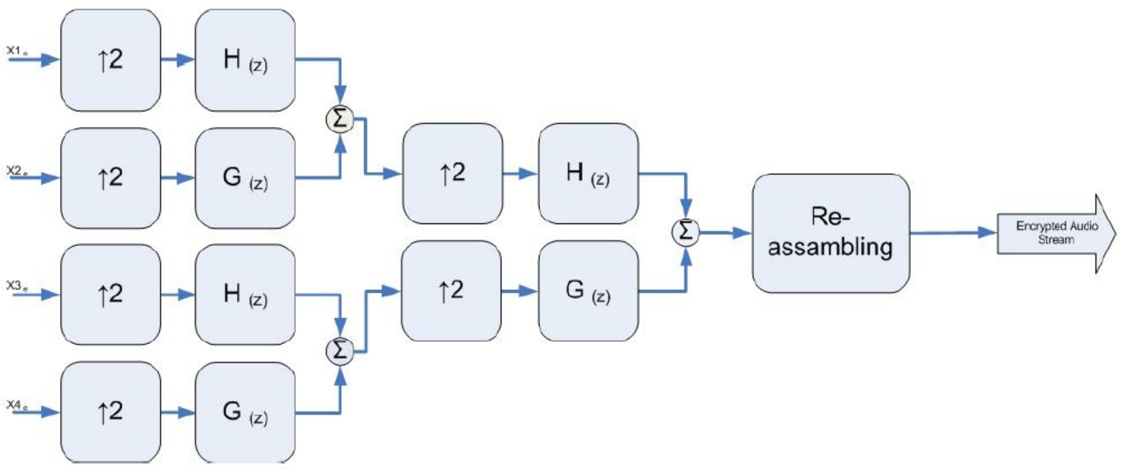


Figure (2.4) Reconstruction of scrambled channel diagram [7]

The complete channel as in figure (2.5).

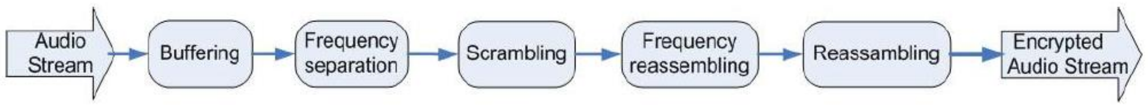


Figure (2.5) Complete channel block diagram

The descrambled channel is the same as scrambled channel and the only difference is the matrix which must be inverted to redistribute the frequency band as in original voice.

The main challenge toward the real time implementation, the buffer length which introduce a delay between the voice input and the output. The time needed for wavelet transformation calculation which need floating multiplications, and the number of level used.

2.3.1.4 Frequency Hopping Inversion

In a frequency hopping spread spectrum (FHSS) system,the transmitted signal is spread across multiple channels as shown in Figure (2.6 a) below . in figure (2.6 b) the full bandwidth is divided into 8 channels , centered at f1 through f8 the signal “hops” between them in the following sequence f5 f8 f3 f6 f1 f7 f4 f2.[13]

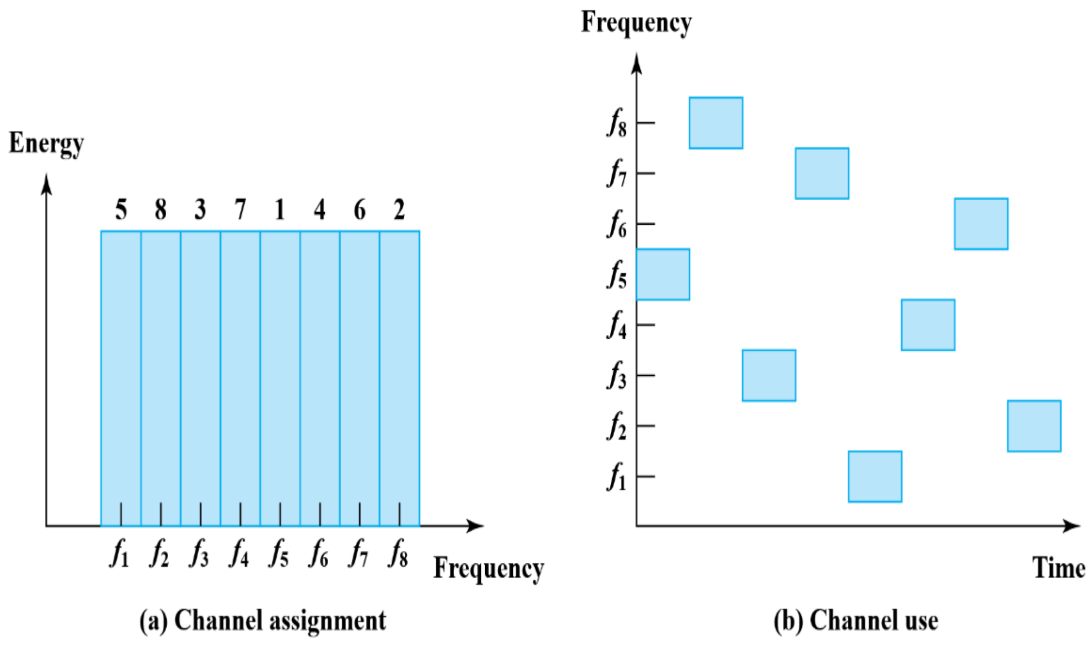

Energy
5 8 3 7 1 4 6 2
f_1 f_2 f_3 f_4 f_5 f_6 f_7 f_8
Frequency
(a) Channel assignment
Frequency
f_8 f_7 f_6 f_5 f_4 f_3 f_2 f_1
Time
(b) Channel use

Figure (2.6) Frequency hopping example [13]

Figure (2.7) shows the block diagram of a typical FHSS transmitter. First digital data is modulated using some digital-to-analog scheme. This base band signal is then modulated onto a carrier.

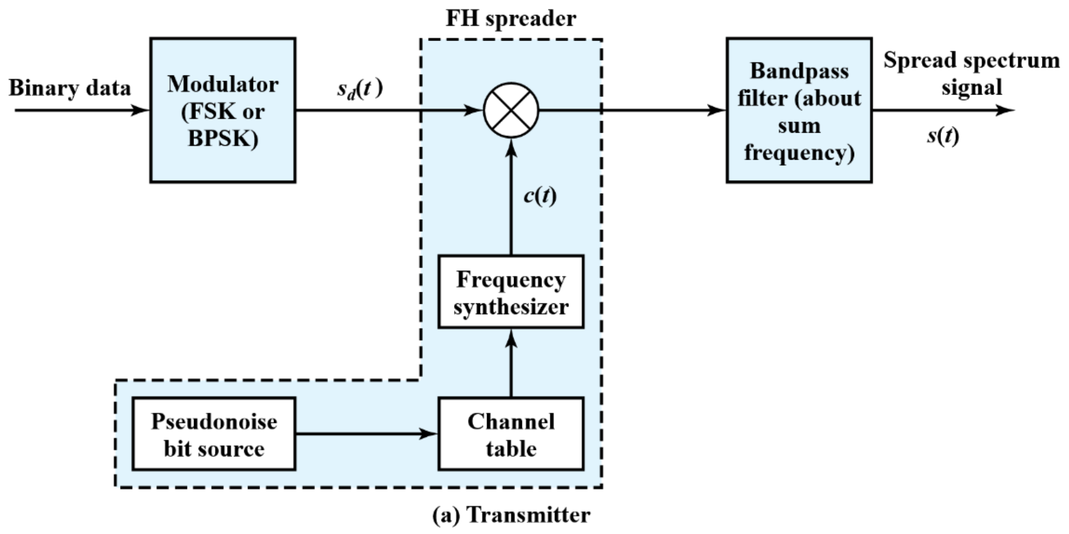


Figure (2.7) FHSS transmitter system

2.3.1.5 Band Splitting

Band splitters divide the typical speech frequency Band into several sub-bands and then rearrange them relative to each other this is illustrated on figure (2.8). Sometimes the band splitting technique is combined with the inversion technique so that sub-bands are not only rearranged but each sub-band is inverted as well. Figure (2.8 b) shows how speech is scrambled by band splitting, with some of the sub- bands inverted.

The fixed-code band splitter rearranges the sub-bands in the same order at all times. Band splitters that continually change the order in which sub-bands are arranged are calling rolling-code band splitters.

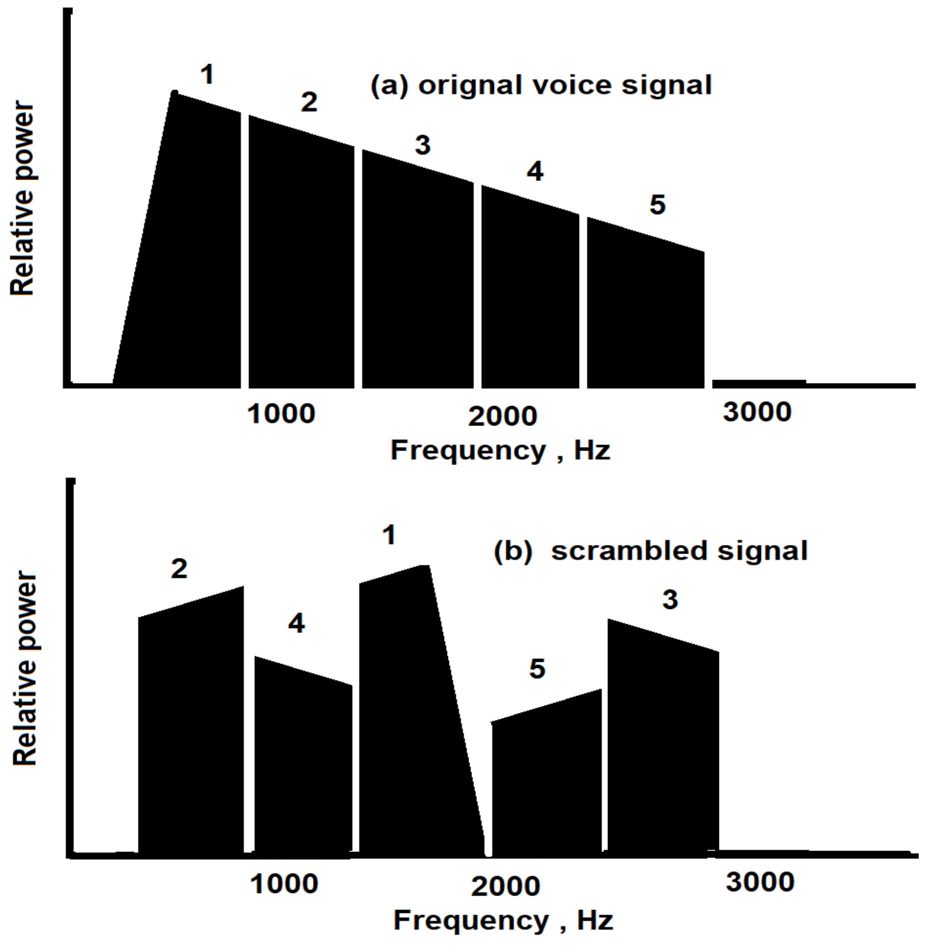


Figure (2.8) Illustrate band splitting process

2.3.1.6 Rolling Code Inversion

Other method is to use a random code to choose the carrier frequency and then change this code in real time. But takes in consideration the synchronization between scrambler and descrambler

2.3.2 Time Domain Scrambling

A common technique is to record voice data for some time and cut it into small frames. These frames can then be transmitted in a different time order depending on a secret code that provides a different not understandable signal. Since the fragments of the spoken words are not in the right order anymore.

In this type of scrambling the voice signal is segmented in time domain. The scrambler changes the order of the segments in time domain using a given sequence (code) while the descramble try to reorder the segments as in original voice signal.

2.3.2.1 Time Segment Permutation (TSP)

Clear voice is first loaded into a memory device and then, upon completion of the storage phase, is read out as a sequence of time-permuted segments of the original “clear voice” signal. The order of the permutation and its rate of change is determined by a preselected Pseudorandom sequence which, in turn, may be changing with time as shown in Figure (2.9) .

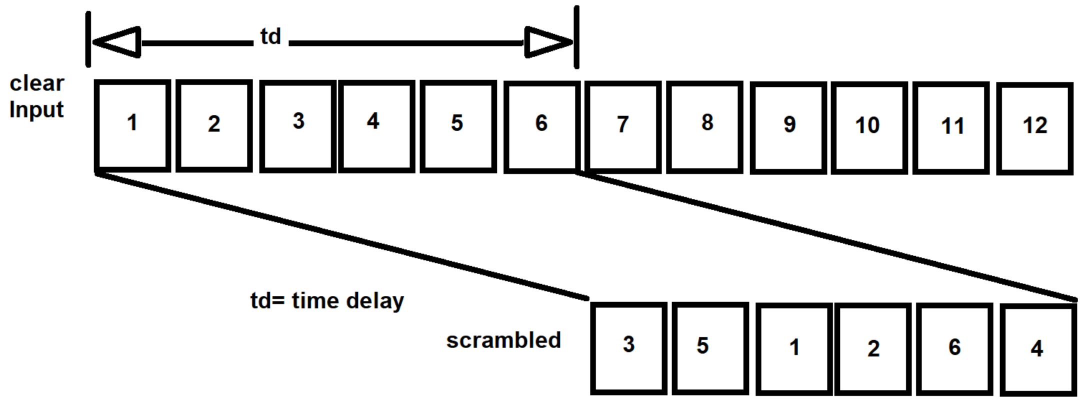


Figure (2.9) Time segment permutation (TSP)

2.3.2.2 Reversed Time Segmentation (RTS)

One of the most efficient methods for destroying the intelligibility of speech segments is to reverse their delivery in time.

Pattern obtained with such a reversed time segment encoding method. Part (a) of the figure (2.10) shows a portion of a speech signal which is progressing in a normal direction, with its segments being generated in the 1, 2, 3, 4, 5... sequence. Part (b) of the figure (2.10) shows how the direction of each of the segments (time intervals) has been reversed, yet the order of their delivery, outputting to the channel, has not been altered. The typical duration of these reversed segments can be in the 50-400ms range, depending on the memory capacity of the system and other implementation considerations.

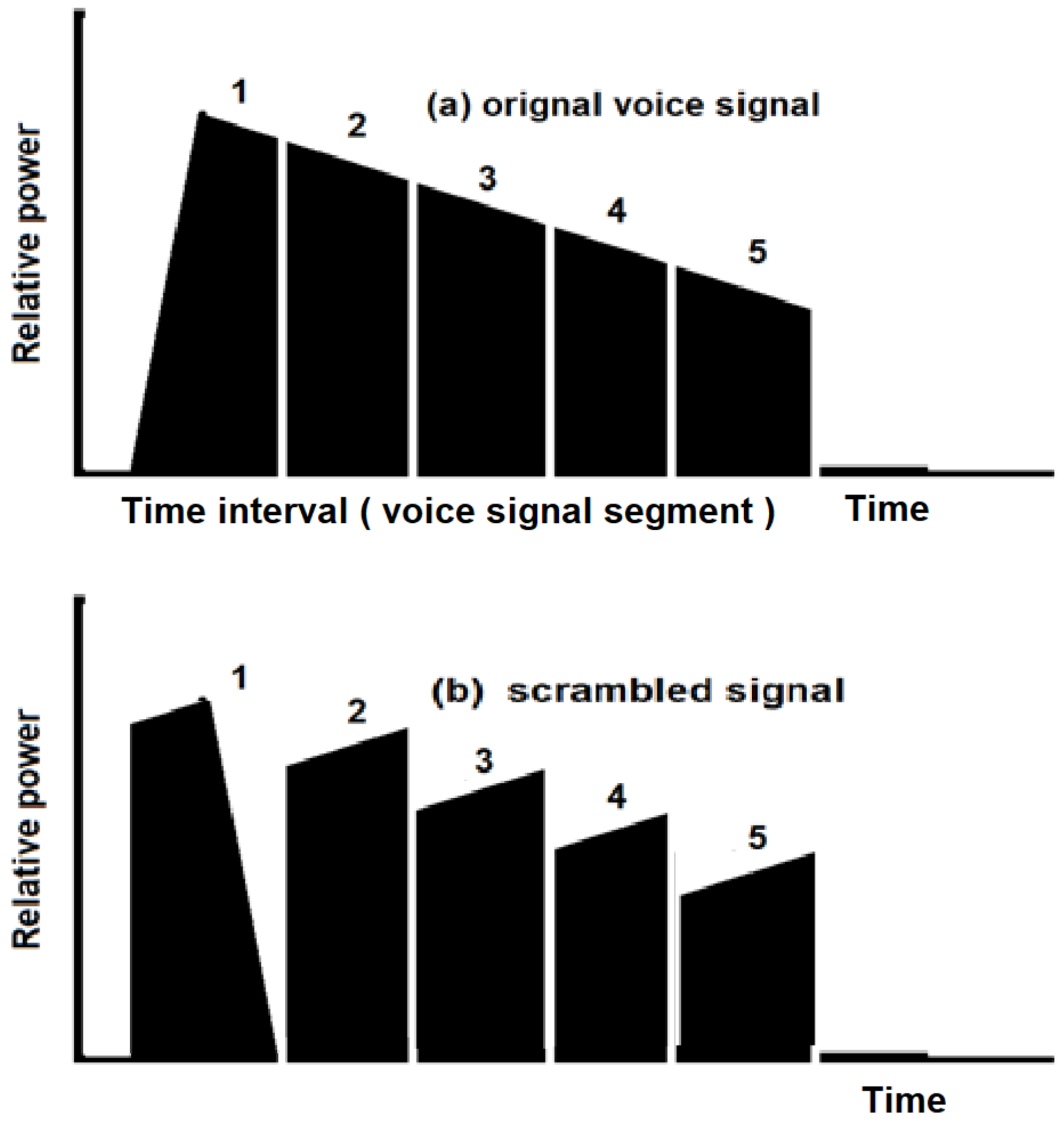


Figure (2.10) Reversed time segmentation

The above techniques used in time domain scrambling require frame synchronization this not only complicates the implementation but make the transmission very sensitive to channel condition. To overcome this a new technique are used for time domain scrambling without the need for synchronization which based on implementation of narrow base band filter in time domain with variable frequency center [9].

But these techniques process could not be implemented in microcontroller in real time operation.

2.4 Frequency Inversion

In frequency inversion,the frequency components of the speech signals are simply flips to up or down,thereby moving the lower part frequency components to the upper part and moving the upper part components to the lower part.

Figure (2.11) shows the frequency inversion technique for the transmitter and the receiver.The frequency range 0.3 to 3khz is the audio range taken by the system and convolute it with a carrier signal thereby achieving a frequency inversion.The modulation of the audio signal with the carrier signal results in shifting the frequency spectrum of upper and lower side bands causing frequency inversion.The audible speech lies on the lower sub band. Figure (2.12 a) and (2.12 b) illustrates the implementation of the frequency inversion for scrambler and descrambler.[8]

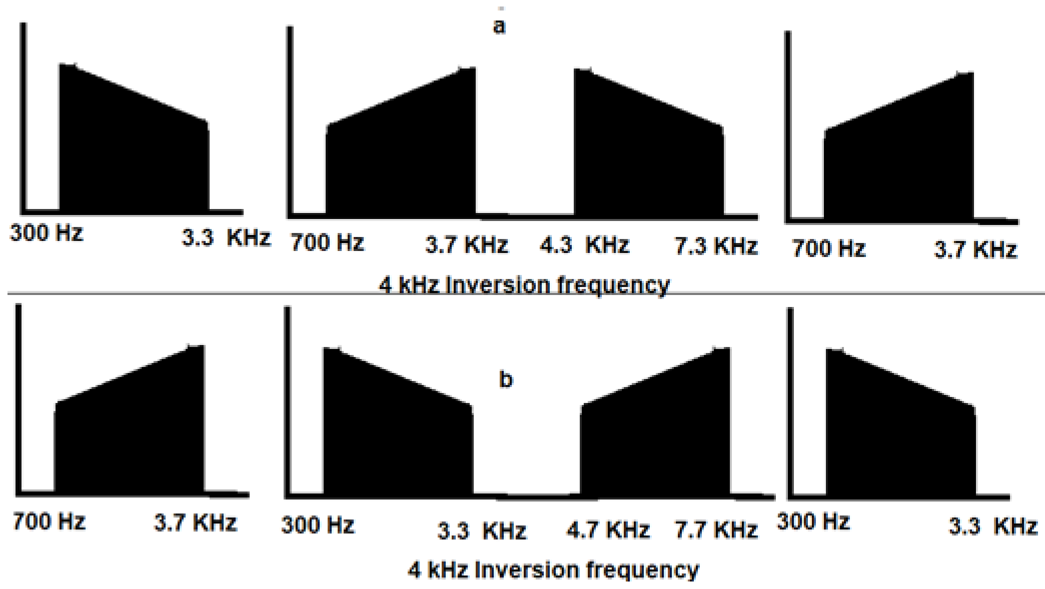
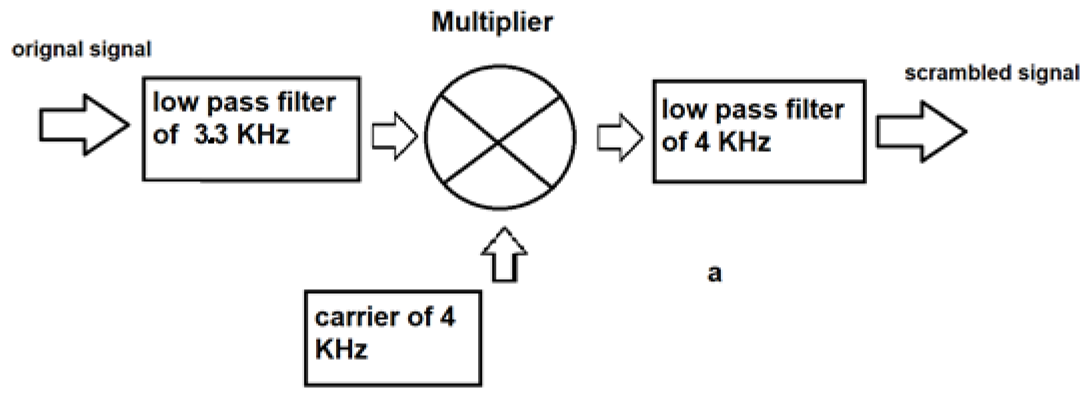


Figure (2.11) Frequency inversion technique[8]



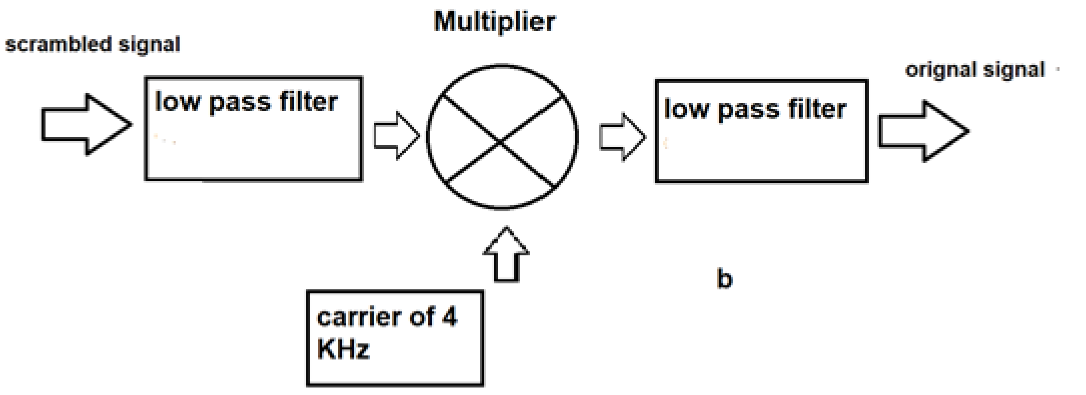


Figure (2.12) illustrates the implementation of the frequency inversion (a) for scrambling and (b) for descrambling

2.4.1 Frequency Inversion Theory

Starting with Fourier transform frequency shift property

$$e^{jw_0t}x(t) \leftrightarrow X(w - w_0) \tag{2.9}$$

This states that if a signal is multiplied by a complex exponential at some Frequency ω_0 , all that happens to the Fourier transform is that it gets shifted by the same frequency ω_0 which can be easily proof starting from the definition of FT.

$$X(w) = \int_{-\infty}^{\infty} x(t)e^{-jwt}\, dt \tag{2.10}$$

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(w)e^{jwt}\, dw \tag{2.11}$$

$$f\{e^{jw_0t}x(t)\} = \int_{-\infty}^{\infty} \left(e^{jw_0t}x(t)\right) e^{-jwt} dt \tag{2.12}$$

$$= \int_{-\infty}^{\infty} x(t)\, e^{-j(w-w_0)t} dt \quad (2.13)$$

$$= X(w - w_0) \quad (2.14)$$

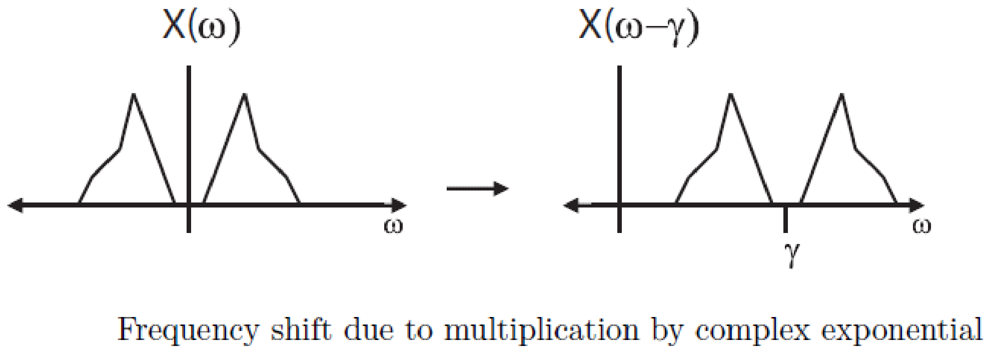

X(ω)
X(ω−γ)
ω
γ
ω
Frequency shift due to multiplication by complex exponential

Figure (2.13) Frequency shift due to multiplication by complex exponential

Since

$$cosx = \frac{e^{ix}+e^{-ix}}{2} \quad (2.15)$$

So the FT of $x(t)cos(2\pi f_1 t)$ will give two bands centered at the frequencyf_1. Two sinusoid are multiplied one with frequency 2000Hz and other with 3500 HZ multiplied, the output frequencies are

$$f_1 = 3500 - 2000 = 1500Hz$$

$$f_2 = 3500 + 2000 = 5500Hz$$

Using low pass filter to pass low band only (1500Hz), if this signal is multiplied by the same sinusoid of 3500Hz the output frequencies will be

$$f_3 = 3500 - 1500 = 2000Hz$$

$$f_4 = 3500 + 1500 = 5000Hz$$

As in Figure (2.14) and (2.15).

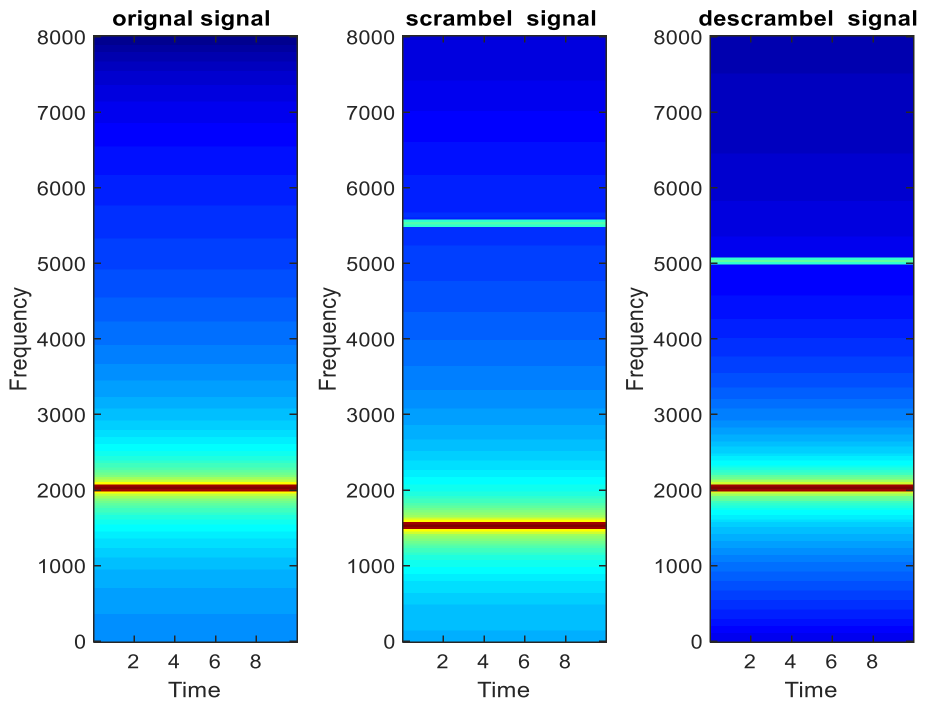


Figure (2.14) Spectrogram scrambling and descrambling using two sine multiplications

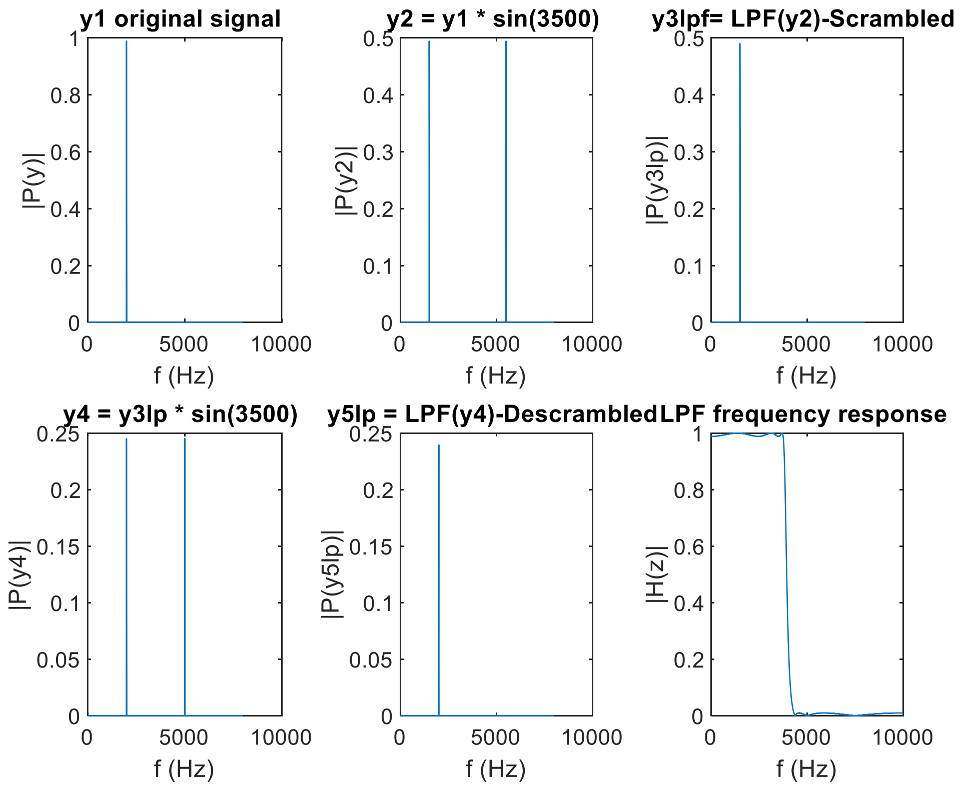


Figure (2.15) FFT scrambling and descrambling using two sine multiplication

2.5 Time Permutation

The technique implemented in this work based on controlling the read and write buffer addresses only without any processing needed. The voice signal is segmented in to 16 buffers each of 1024 samples from ADC; the output from ADC is stored in circular buffers sequentially (Buf0, Buf1, Buf2………. Buf15, Buf0…). By controlling the buffer write address for read operation, the buffers to DAC the controlled by permutation code table as shown in Figure (2.15).

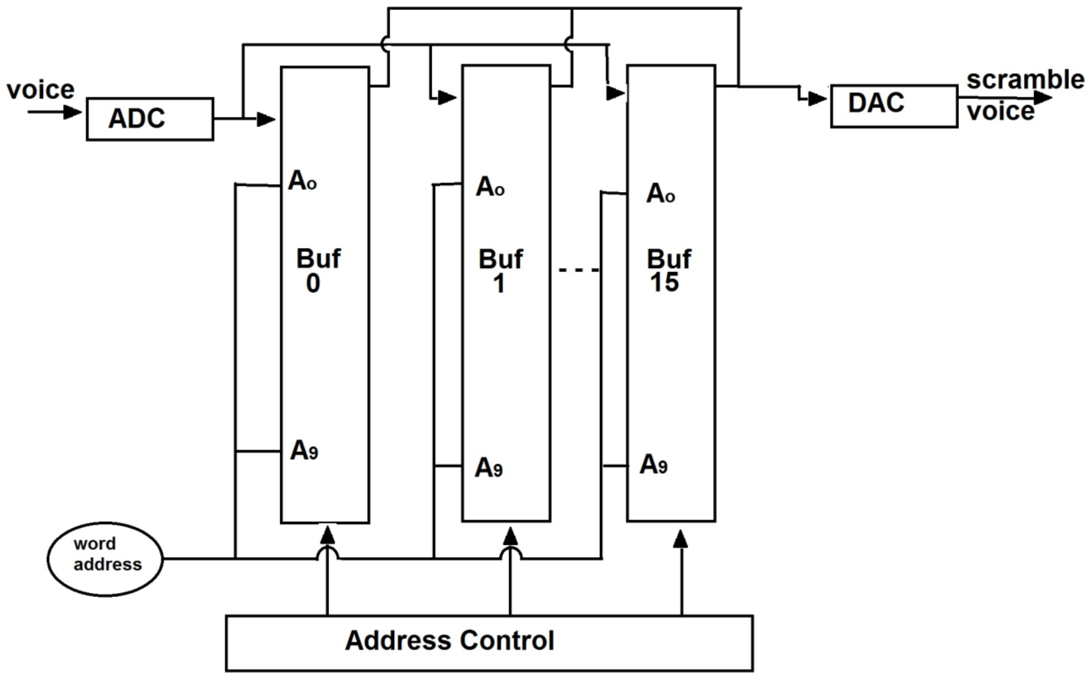


Figure (2.16) Time permutation process

Noting that the word address is controlled by a counter counting from 0 to 1023 with sampling frequency clock.

2.6 Software and Tools

The software and tools used in this research will be explained in the following subsection.

2.6.1 Matlab V.2018

MATLAB Simulink is used as step one for verification and implementation.

2.6.2 Arduino IDE 1.8.7

ARDUINO IDE 1.8.7 is used with c language for the programming and downloading of the modules to microcontroller on ARDUINO DUE as shown in more details in chapter 4.

2.7 Hardware Used

2.7.1 ARDUINO Due

Figure (2.17) Arduino Due microcontroller

ARDUINO Due is chosen for the following

- The ARDUINO Due is a microcontroller board based on the Atmel SAM3X8E ARM Cortex-M3 CPU. It is the first ARDUINO board based on a 32-bit ARM core microcontroller. It has 54 digital input/output pins (of which 12 can be used as PWM outputs), 12 analog inputs (12 Bits), 4 UARTs (hardware serial ports), an 84 MHz clock, an USB OTG capable connection, 2 DAC (12-Bits digital to analog), 2 TWI, a power jack, an SPI header.

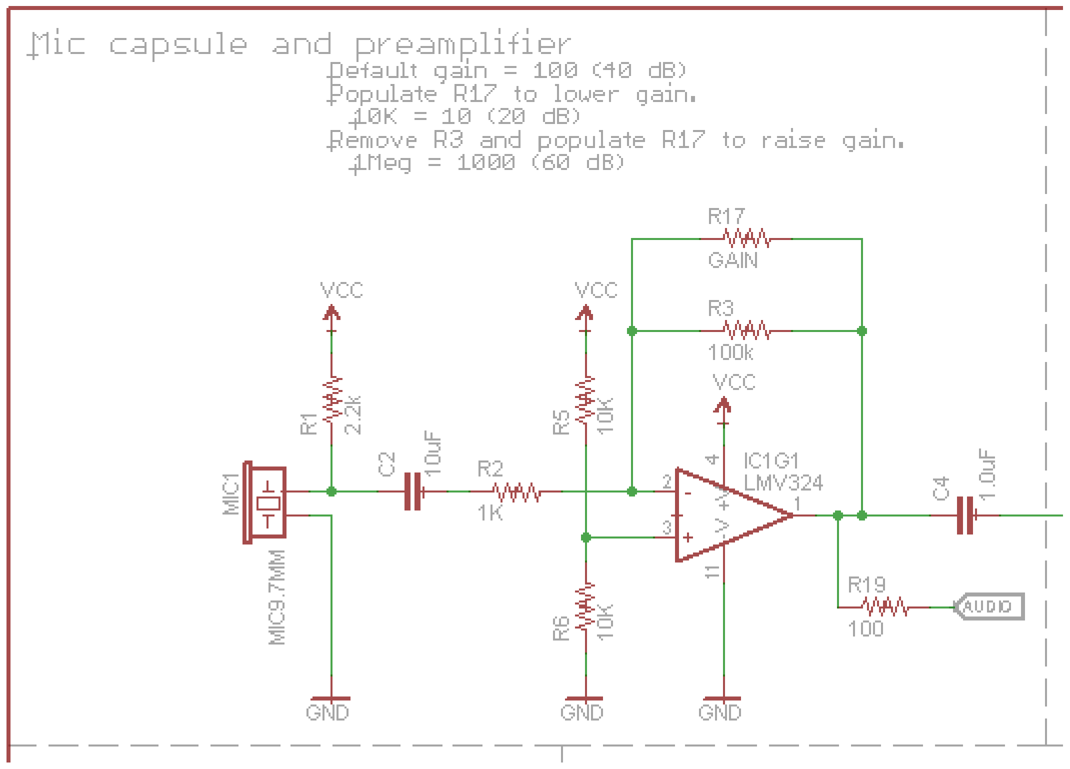
- Peripheral DMA Controller (PDC) to handles data transfer between peripherals and memory without the intervention of the processor.
- Three timers counters modules and each of these has three channels giving nine timer counter channels in all. In this implementation both ADC and DAC are triggered by channel 0 and channel 1 respectively.
- A configurable nested interrupt controller (NVIC) to deliver industry-leading interrupt performance. The NVIC provides up to 16 interrupt priority levels, dramatically reducing the interrupt latency, fast execution of interrupt service routine (ISR).
- 256 to 512 Kbytes embed Flash with 32 to 100Kbytes SRAM with dual banks.
- 16-channel 12-bit ,1 MBPS ADC with differential input mode and programmable gain stage.
- 2-channel 12-bit 1 MBps DAC.
- Rich libraries for programming.

2.7.2 MIC Interface and Amplifier

A ready card Sparkfun Sound Detector is used as shown in figure (3.5).

Figure (2.18) Sparkfun card

With MIC interface and amplification as shown in Figure (3.6).



Microphone and Preamplifier

Figure (2.19) Microphone and preamplifier

2.7.3 Speaker Interface and Amplification

A circuit from data sheet of LM386 is chosen and converted to printed board as shown in figure (3.7).

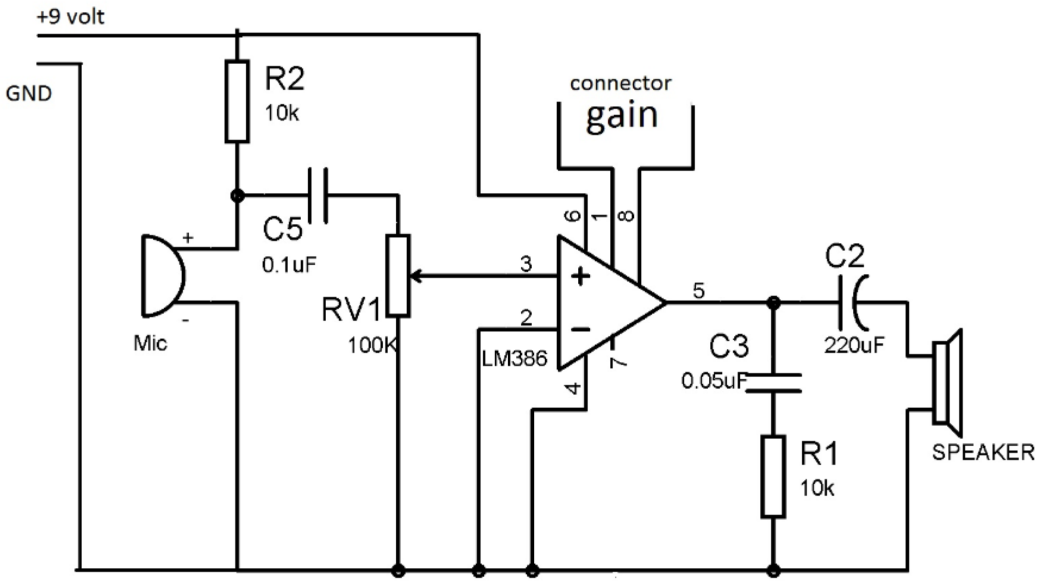


Figure (2.20) LM386 circuit diagram

2.7.4 Speaker

Speaker 8 ohm used to listen to voice output as shown in figure (3.8) .

Figure (2.21) 8-ohm speaker

2.8 Literature Review

This section introduces some of related and previous studied and works in this topic then it shows the final results of their work.

In **[**1**]** a modified approach for frequency permutation based on FFT of speech signal has been designed implemented and verified based on MATLAB Simulink. A separate permutation of real and imaginary parts of the frequency component has been used. A Short Time Objective Intelligibility has been used for checking the performance and residual intelligibility of this method. The result shows a positive indication as compared with classical permutation. But no indication for real time implementation.

In [2] the authors propose a platform to secure voice data over wireless mobile communication using multi-circular permutations rotated by dynamic keys to scramble the data. However, the pc is used for data scrambling descrambling before sending to Motorola using two ways low speed audio frequency shift keying, (AFSK) so the proposed system could be considered as data encryption not a voice scrambling.

In [3] the author uses an orthogonal frequency division multiplexing (OFDM) scheme, to scramble speech without residual intelligibility by permuting several frequency components. The results and simulations show that the original speech signal could be scrambled into unintelligible signal and it is also being confirmed that the scrambled signal could be recovered into the original speech signal. Also, the residual intelligibility of the scrambled speech is very low, since the intelligibility of the speech is excellent and the recovered speech files are identical to the original ones. However, The OFDM scrambler is suitable for frequency selective fading channels and it is a best technique for providing high security in the next generation mobile communication system

The authors in [4] provides an idea for encryption of voice signal in wired public switched telephone network (PSTN) system as it has better connectivity, and is noise free. The idea of the paper is to ensure advance level of audio security using rapid time domain scrambling. The 256kbit of ram can store 4 seconds of data considering delta modulation is done at 64kbits of data rate the double buffering initially in the transmission side and then in the receiving device and together with additional synchronization signals needed, however the overall signal delay will accumulate up to 2.6 seconds. This is indeed to slow for the Full Duplex period. Therefore, the communication used in the crypto mode is a Half-Duplex. This operating standard of encryption mode is similar to the walky talky with PTT (push-to-talk) button. The hardware platform is constructed using standard AVR microcontroller (8 bits) Atmega8515 with added SRAM for temporary storage of voice.

In this paper [5] three scrambling techniques are presented and their performances are analyzed. These methods were tested based on some real audio signals that they were recorded in an anechoic chamber, shifted spectrum, sub bands and sub bands and shift spectrum. However it’s not a real time operation.

In [6] the authors propose wavelet transform scrambling system, speech has been recorded with sampling frequency 11.025 KHz as wav files. At the scrambler, the sample speech signal is converted into frames with each frame containing 64 samples and then the Wavelet Transformation is performed on each frame. After that, the transform coefficients are permuted. The resulting scrambled speech signal is saved in a wave file. At the descrambler, frame by frame of length 64 samples are descrambled and saved in wave file. The proposed system investigates one type of Wavelet (Daubechies 4), with one level and not a real time system operation.

In [7] fast hartly transform is used. The hartley transformed coefficients are then given to the scrambler which scrambles the coefficients according to the OFDM scrambling keys generated by the algorithm process. After scrambling, inverse hartley transform is applied to the frequency scrambled hartley coefficients, which is same as the direct transform. The IFHT output is converted to analog by DAC and after multiplexing with synchronization signal and training sequence it is released to the channel. The speech signal is sampled at a rate of 8000 samples per sec. The input signal is processed over a band width of 3.1 kHz, ranges from 300-3400 Hz. The transmission channel band limited to the range of 0-4000Hz.The 256 samples frame size results in 32ms frame time. However, the proposed system was simulated in matlab.

In [8] the authors Presents speech security method which enables variety of encryption operations in the combined time and frequency spaces. The analog speech encryption operations are performed by an important Bi-orthogonal transformation technique in order to give the communication systems that based on using a speech signal high degree of the security. The noisy communication channel usually accompanying the encryption processes is studied. Different case studies were taken into consideration, through using different types of Biorthogonal transforms. The performance of the proposed scheme is examined through the calculation of evaluation measure (the segmental signal to noise ratio). The results are promising for analog speech encryption systems.

In [9] the authors present an overview of the main analog scramblers which are, time domain, frequency domain and Encryption by using Pseudo-Noise Sequences. Using XOR operation of these sequences with the speech samples makes the speech signal a noise-like signal, and the encrypted speech signal sounds like a random noise signal. Although The intangibility of speech signal can be reduced by removing the correlation among the speech samples by using XOR with Pseudo-Noise sequences, this operation will affect the voice channel properties and could be used for data communication techniques like voice over internet protocol.

In [10] the authors Presents a real-time frequency voice scrambler algorithm and its Implementation in TMS 320C6X - TEXAS Instruments DSP (Digital Signal Processor) architecture. The voice channel cryptography system is based on a scrambled discrete Hartley Transform. It includes large energy random components inserted into the voice channel bandwidth. As result it's not possible to identify pauses between words, and it is also observed that the speaker could not be identified. The proposal uses low cost hardware architecture adapted for tactical level security in network voice communications. Real-time experimental results have been performing well in a variety of analog voice channels.

In [11] describes the real time implementation of frequency domain scrambling and descrambling techniques on Texas TMS320C6711 DSP, three techniques are implemented. frequency inversion, frequency hopping inversion and rolling code inversion.

In [12] a band scrambler which processes only time domain Samples are described. The band scrambler has the effect of dividing the input signal spectrum into N sub-bands. The N sub-bands are permuted such that the rth band is mapped onto the k.rth band modulo N, where N is a constant of the scrambler and k is the key which is variable in the range 2<k.<N-l.

The authors in [13] propose time domain scrambling which does not need synchronization. The scrambler permutes individual speech samples rather speech segments with band pass filter process with variable center frequency implemented in time domain the process. Made to be equivalent to a frequency-band scrambler.

The authors in [14] describe the hardware architecture and the signal processing algorithm of a secure telephone which can be used over the public switched network. Using time-varying frequency-domain scrambling, a high level of security can be achieved, with no residual intelligibility. By means of the DFT filter banks implemented by the FFT and polyphase networks; the entire signal processing for a full duplex conversation can be performed with a single TMS320C25 digital signal processor. The complete scrambler fits inside a standard telephone set unit.

The authors in [15] Present FFT scrambling method based on the manipulations of the Fast Fourier Transform (FFT) coefficients of speech is one of the analog speech scramblers which realize high-level security and excellent descrambled speech quality. It is not clear, however, how the descrambled speech quality and synchronization performance are affected by transmission impairments. This paper analyzes the effects of transmission impairments on the descrambled speech quality by computer simulations and provides a guide for the system design. Then practical methods to cope with the effect of transmission impairments are proposed. Based on the results of the simulations, experimental scramblers are developed using general-purpose DSP's (Digital Signal Processors). The system configuration and the result of the performance evaluation tests are shown. The experimental scramblers are equipped with an automatic equalizer for group delay and amplitude distortions, as well as the Automatic Frequency Control (AFC). An echo canceller is also provided so that the full-duplex operation is achieved using two-wire lines. As a result of the test under the simulated environments, it is proved that the descrambled speech quality is satisfactory and the proposed method is effective in practice.

The authors in [16] Propose microprocessor based analog voice scrambling system with frequency domain and time domain implementation

CHAPTER THREE

SYSTEM DESIGN AND SIMULATION

CHAPTER THREE

SYSTEM DESIGN AND SIMULATION

3.1 Introduction

This chapter introduces the system design and simulation steps with software tools related to the research.

3.2 Research Design

Review of different techniques used in voice scrambling mainly for Real time applications and Select frequency inversion with programmable center frequency inversion and select random time permutation for time domain scrambling.

Review the different microcontroller available with free development tools and choosing ARDUINO DUE for its specifications oriented to real time applications.

Verifications of the designed algorithms using MATLAB SIMULINK and MATLB .m files. Implement the scrambling and descrambling algorithms using ARDUINO DUE with sampling frequency 16 KHz. Implement the random time permutation based on manipulating the read addresses of the memory inside ARDUINO DUE.

testing the operation in real time by scrambling the input voice from MIC and output the descramble voice through speaker .

3.3 Proposed System

The proposed system is designed to scramble and descramble voice signal in real time using two methods frequency scrambling and time permutation as shown in figure (3.1) and figure (3.2).

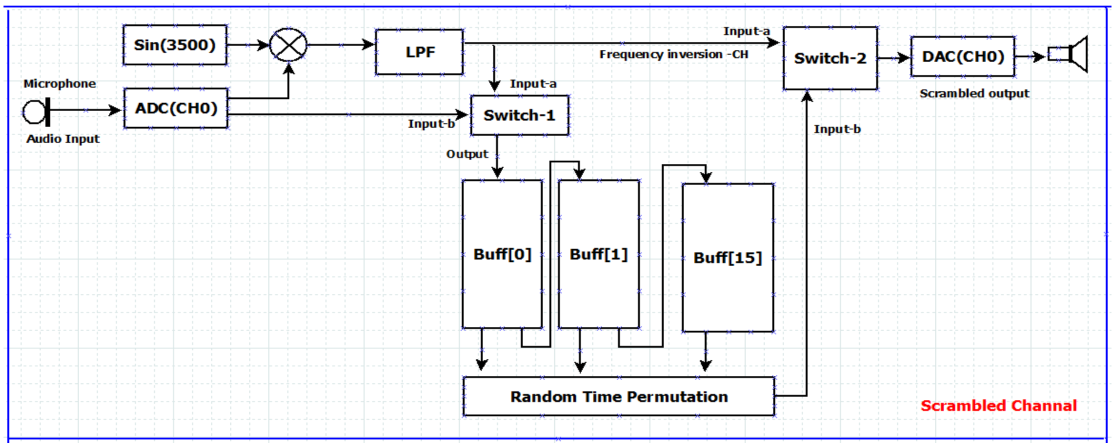


Figure (3.1) Propose system block diagram for scrambling

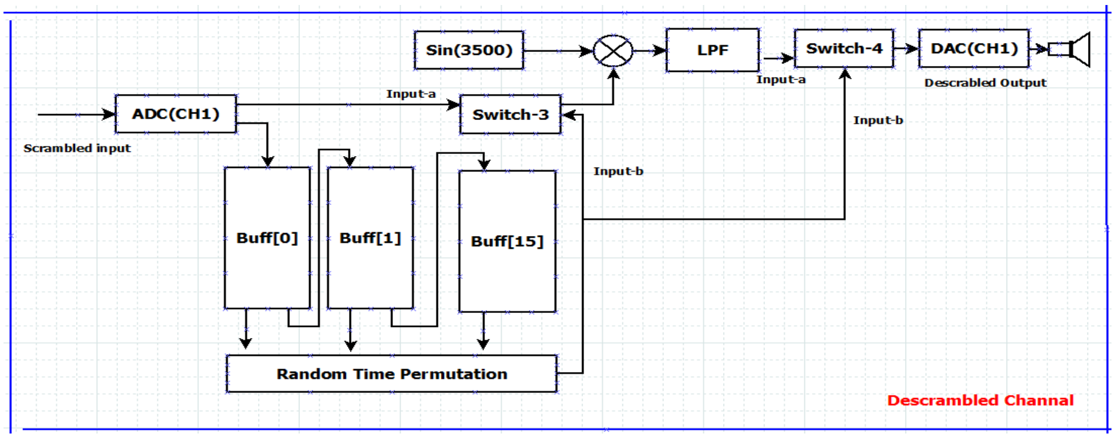


Figure (3.2) Propose system block diagram for descrambling

The switches in the above diagrams control the scrambling types used. For example, if frequency inversion only is used. Swith-2 in scrambled channel have to pass input-a to DAC.If we want to use frequency inversion and time scrambling, switch-1 has to pass input-a to time scrambling module and switch-2 have to pass input-b.

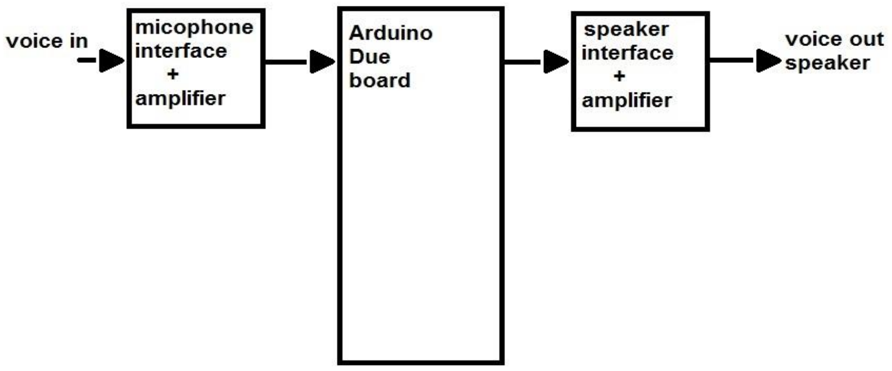

voice in
micophone interface + amplifier
Arduino Due board
speaker interface + amplifier
voice out speaker

Figure (3.3) System block diagram

In frequency system the system has to be capable of implementing frequency inversion. The core module for frequency inversion used as a base for implementing the different types of frequency domain scrambling. As shown in figure (3.4)

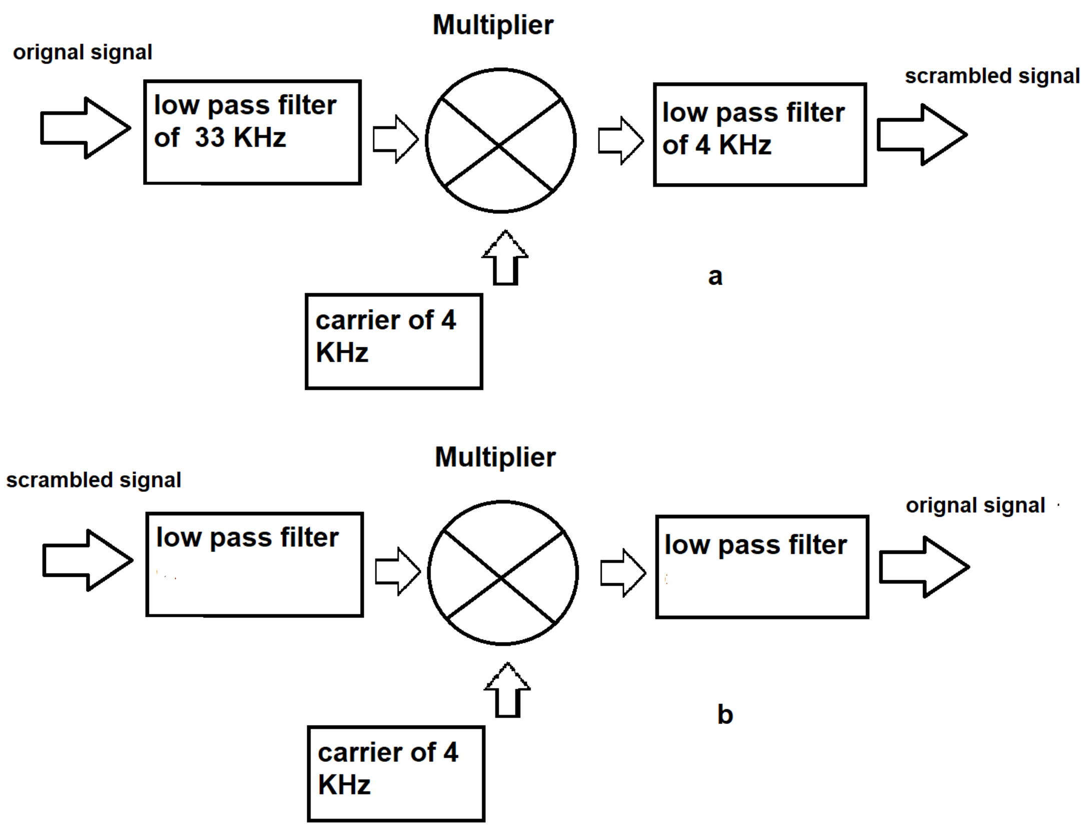


Figure (3.4) Frequency inversion system (a) for scrambling (b) for descrambling

In time domain the system has to be capable for implementing time segment permutation with the possibility of sampling inversion in each sequence as explained in chapter two. The scrambling in time domain have to be implemented by controlling the read addresses of the memory without any additional hardware.

The system design is divided into two phases, the simulation and test phase as in this chapter and the implementation phase as in chapter four.

3.4 Step-1 Matlab Simulink Simulation

MATLAB Simulink is used as step one for verification and implementation as in the following tests.

3.4.1 Test-1

For the test and verification of two sinusoids multiplications. Two sinusoid sine wave, first one with 2000Hz is multiplied by second one with 3500 Hz, the spectrum of multiplication is shown in Spectrum Analyzer 3 in figure (3.6). Spectrum Analyzer 1 show the spectrum of multiplication after passing through low pass (scramble channel). While Spectrum Analyzer 2 show the spectrum of scramble channel after passing through the second low pass Filter (descramble channel) as shown in figure (3.5) and figure (3.6).

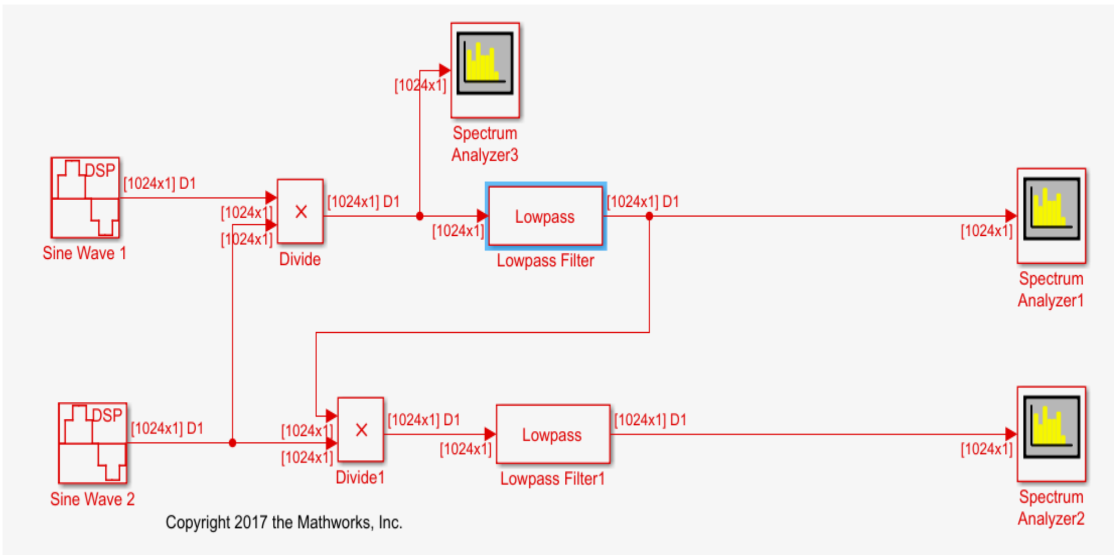


Figure (3.5) Matalb simlink (multiplication of two sines)

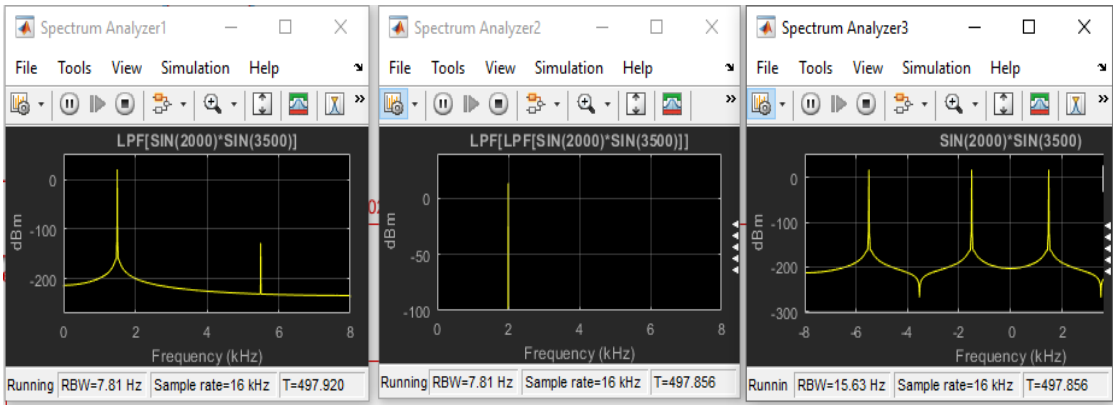
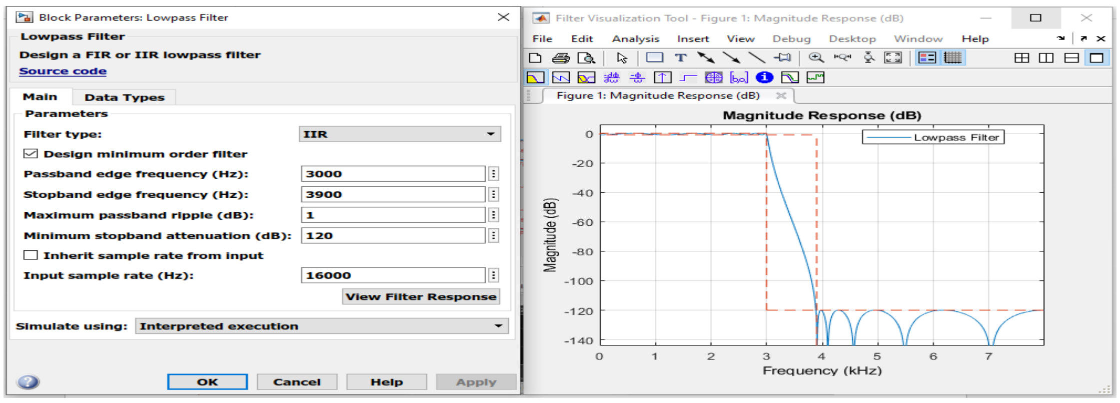

Spectrum Analyzer1
LPF[SIN(2000)*SIN(3500)]
dBm
Frequency (kHz)
Running RBW=7.81 Hz Sample rate=16 kHz T=497.920
Spectrum Analyzer2
LPF[LPF[SIN(2000)*SIN(3500)]]
Running RBW=7.81 Hz Sample rate=16 kHz T=497.856
Spectrum Analyzer3
SIN(2000)*SIN(3500)
Runnin RBW=15.63 Hz Sample rate=16 kHz T=497.856

Figure (3.6) Matlab spectrum analyzer


Block Parameters: Lowpass Filter

Lowpass Filter

Design a FIR or IIR lowpass filter

Source code

Main Data Types

Parameters

Filter type: IIR

☑ Design minimum order filter

Passband edge frequency (Hz): 3000

Stopband edge frequency (Hz): 3900

Maximum passband ripple (dB): 1

Minimum stopband attenuation (dB): 120

☐ Inherit sample rate from input

Input sample rate (Hz): 16000

View Filter Response

Simulate using: Interpreted execution

OK Cancel Help Apply

Figure (3.7) low pass filter design using Matlab

3.4.2 Test-2

The audio signal from MIC is multiplied by sinusoid sine wave as in figure (3.8). Spectrum analyzer 3 show the spectrum after multiplication, which show the low band and the high band centered on sinusoid frequency(3500Hz). Spectrum analyzer 2 as in figure (3.9) show the spectrum after low pass filter with 3500Hz cut off, which represent the scramble channel (the high frequency become low frequencies). While spectrum analazer-1 as in figure (3.9) show the spectrum after second sinusoid multiplication and low pass filtering (which represent the descramble channel).

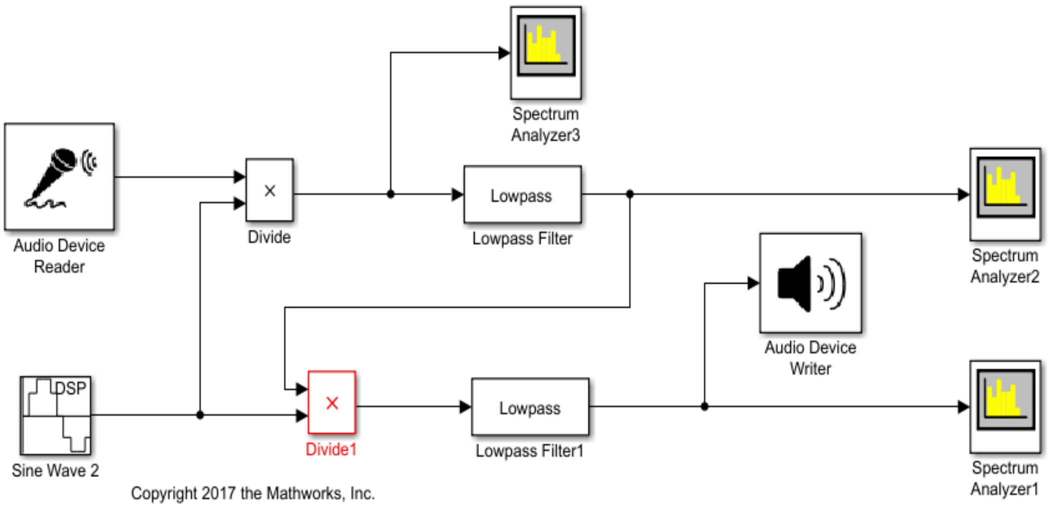


Figure (3.8) Matlab simlink (multiplication voice signal with sine wave)

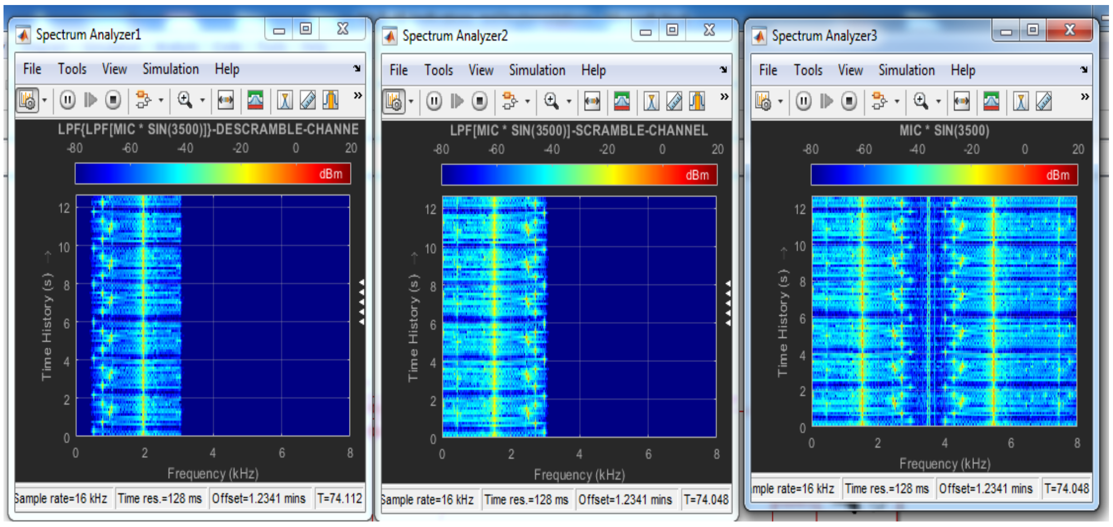


Figure (3.9) Spectrum analyzer 1 ,2 and 3

3.4.3 Test-3

In this test a time scramble is used the audio from MIC is divided in to 10 frames each of 0.1 sec. The position of the frames for scramble channel are changed in MATLAB FUNCTION as (10,1,9,2,8,3,7,4,6,5). While for descramble channel frames are arranged normally (1,2,3,4,5,6,7,8,9,10). Spectrum Analyser3 as in figure (3.11) for original voice, spectrum Analyser1 for scramble channel and spectrum Analyser2 for Descramble channel.

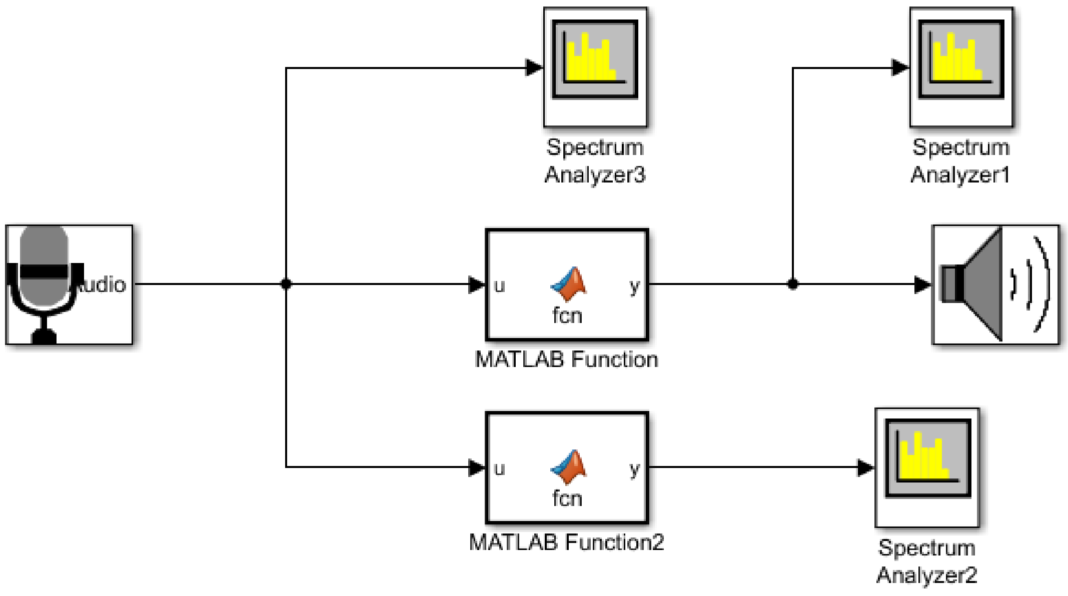


Figure (3.10) Matlab simlink of time scrambling

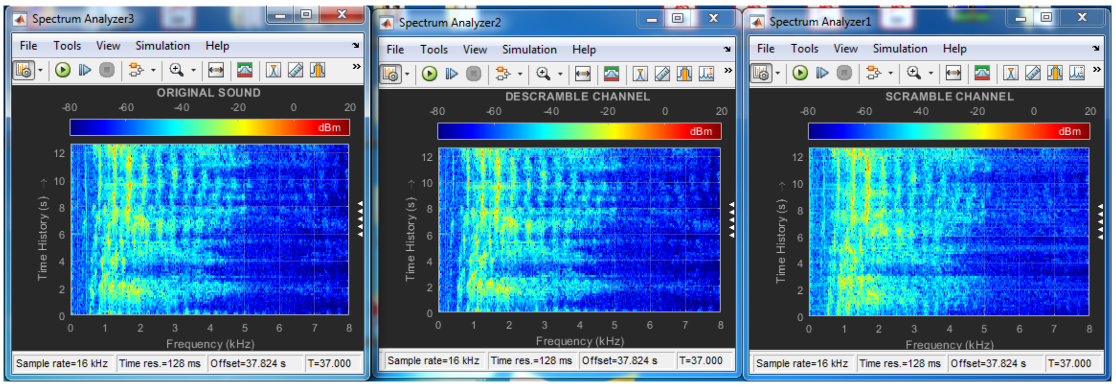
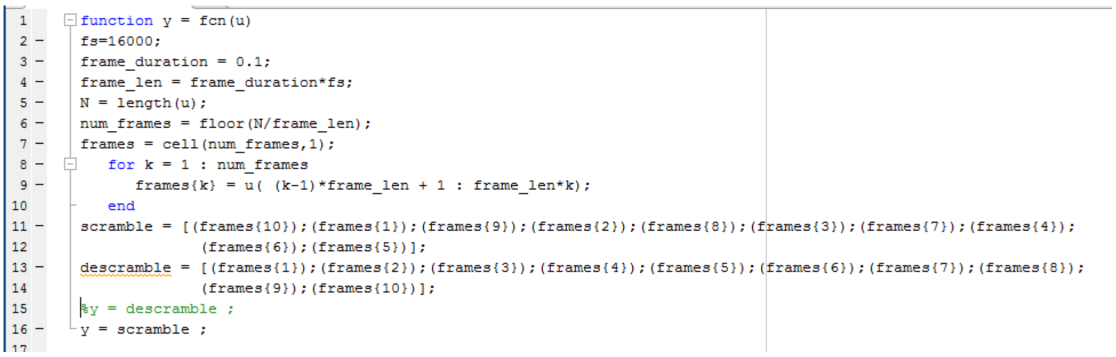


Figure (3.11) Spectrum analyzer for time domain



```
function y = fcn(u)
fs=16000;
frame_duration = 0.1;
frame_len = frame_duration*fs;
N = length(u);
num_frames = floor(N/frame_len);
frames = cell(num_frames,1);
   for k = 1 : num_frames
      frames{k} = u( (k-1)*frame_len + 1 : frame_len*k);
   end
scramble = [(frames{10});(frames{1});(frames{9});(frames{2});(frames{8});(frames{3});(frames{7});(frames{4});
                  (frames{6});(frames{5})];
descramble = [(frames{1});(frames{2});(frames{3});(frames{4});(frames{5});(frames{6});(frames{7});(frames{8});
                  (frames{9});(frames{10})];
%y = descramble ;
y = scramble ;
```

Figure (3.12) Matlab Function for time scrambling

3.4.4 Test - 4

In this type of scrambling the positions of FFT filters are changes in a certain manner (the Code) to produce scrambled channel. And return to normal position to produce Descramble Channel.

In this test a signal composed from three sinusoids having the frequencies (100,200,300Hz) is passed to FFT block of 1024 points and then to user defined function to implement scrambling algorithm. Then to user defined function to implement the descramble channel.

Noting that the changes of FFT filters are done in scramble user function after FFT and the return to the original FFT filters is done in user function before IFFT as shown in figure (3.13) and (3.14). Where spectrum analyzer is sum of thee signal together and spectrum analyzer 1 is for descramble of three signals and spectrum analyzer 2 descramble of three signals.

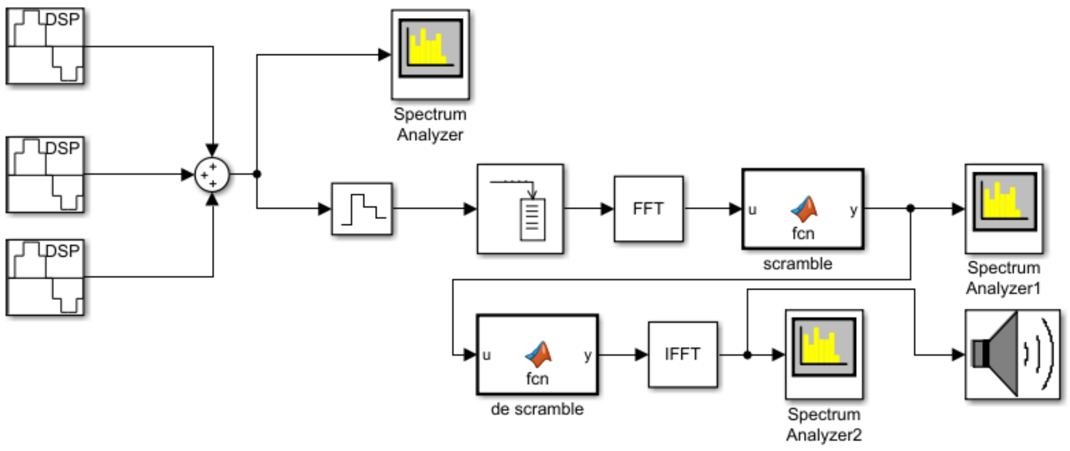


Figure (3.13) Matlab Simlink of FFT

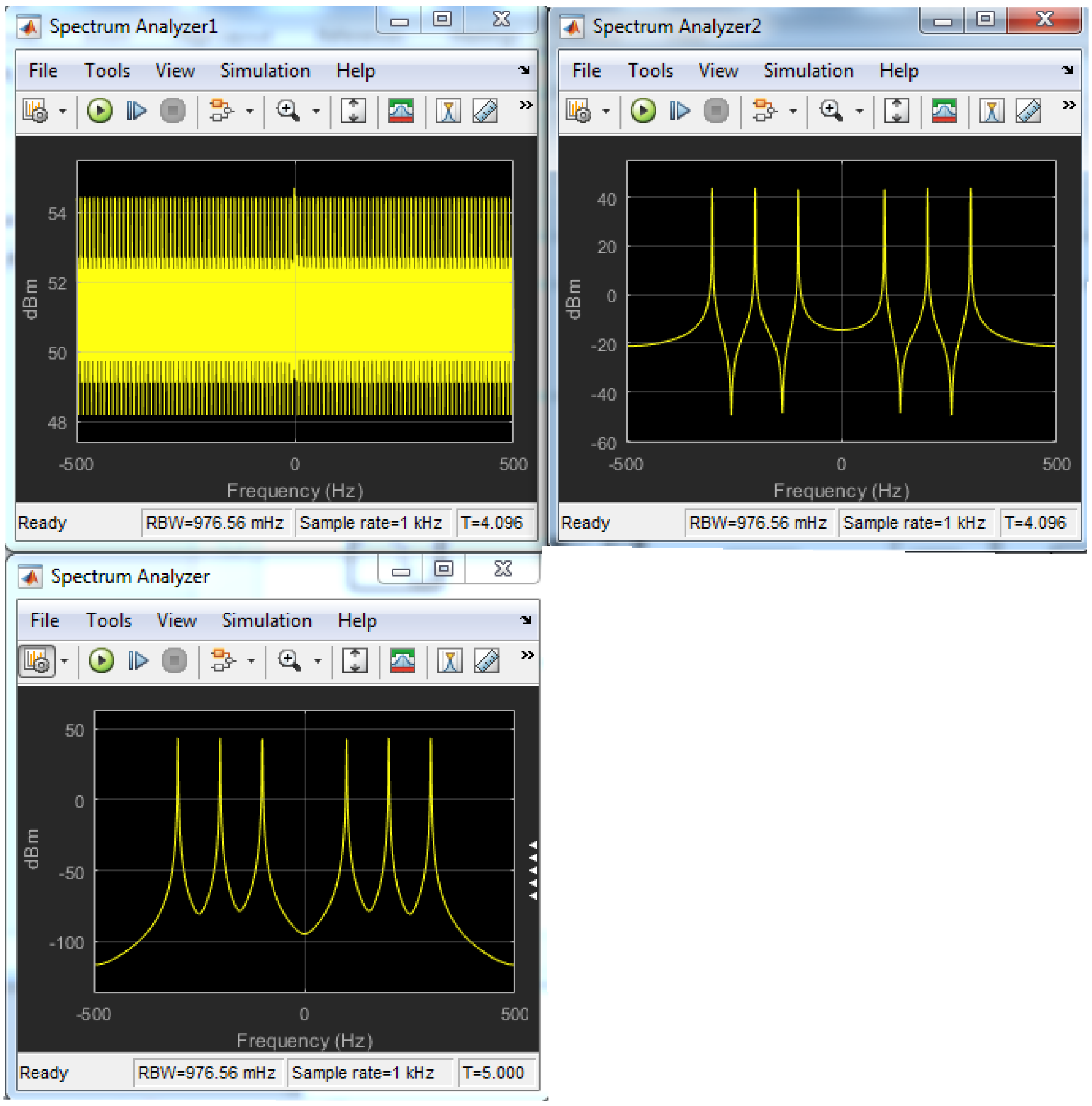


Figure (3.14) FFT spectrum analyzer for test 4

3.4.5 Test - 5

The same as in test4 but the input is from MIC to test real time application. It is clear the similarity between the spectrums of original input(Spectrum2) And the spectrum of the voice after descramble (Spectrum). While the Spectrum1 shows the scramble channel as shown in figures (3.15), (3.16).

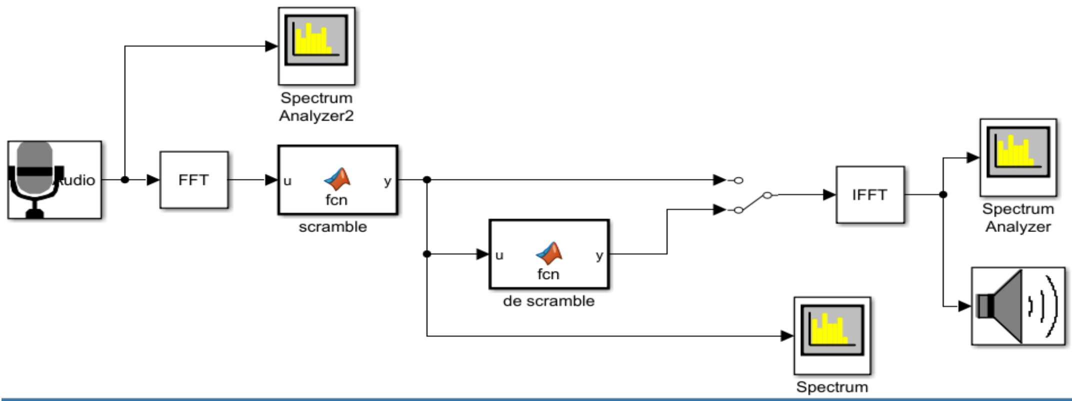


Figure (3.15) Matlab Simlink for FFT using real voice signal input

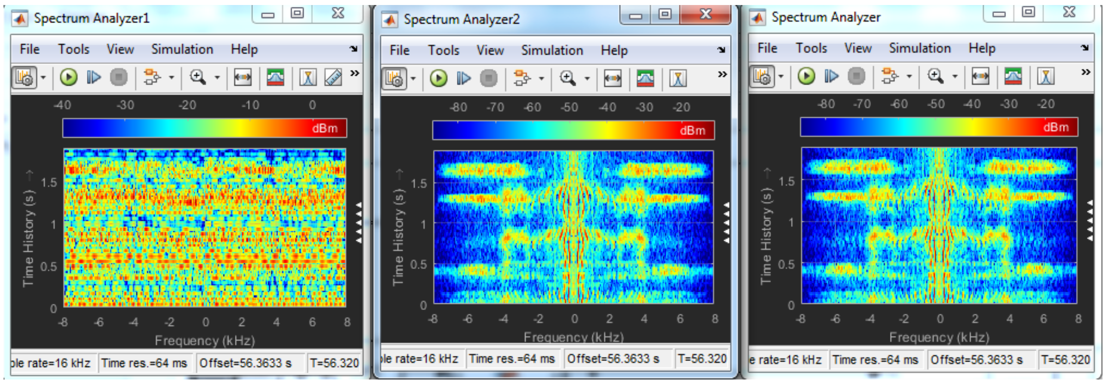


Figure (3.16) Spectrum analyzer for FFT using real voice signal

3.5 Step-2 Matlab .m Files Simulation

In step-2 verification and implementation a MATLAB program (.m files) are written to verify the algorithms to be implemented in microcontroller as final step for the system.

3.5.1 Test - 1

A MATLAB (.m file) is used to generate two sinusoids of frequencies 2000 and 3500Hz. The sin(2*pi*2000t) is multiplied by sin(2*pi*3500t) and pass through low pass filter to give the scramble channel (which contain two frequencies 1500 result from (3500-200) and 5500 result from (2000 + 3500)). The output from low pass filter will pass only sin (2*pi*1500). The scramble channel is multiplied by sin(2*pi*3500) to produce two frequencies (2000 and 5000).

Passing the results of multiplication through low pass filter gives us original frequency (2000), which is the output of descramble channel. As shown in the figures (3.17) and (3.18).

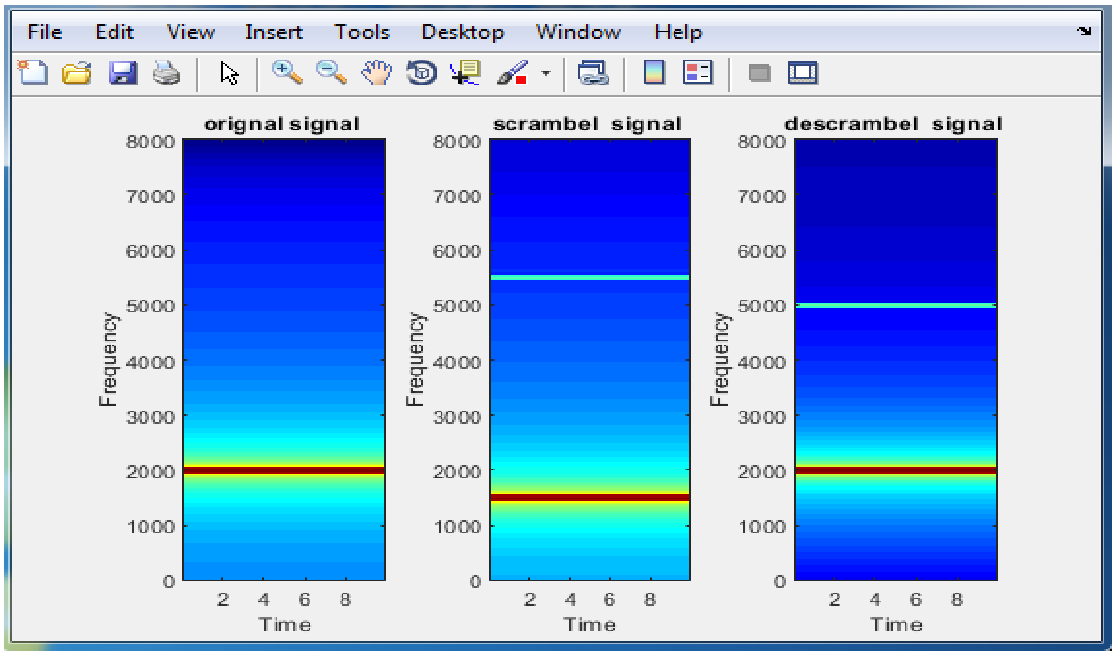


Figure (3.17) Spectrum analyzer for test 1

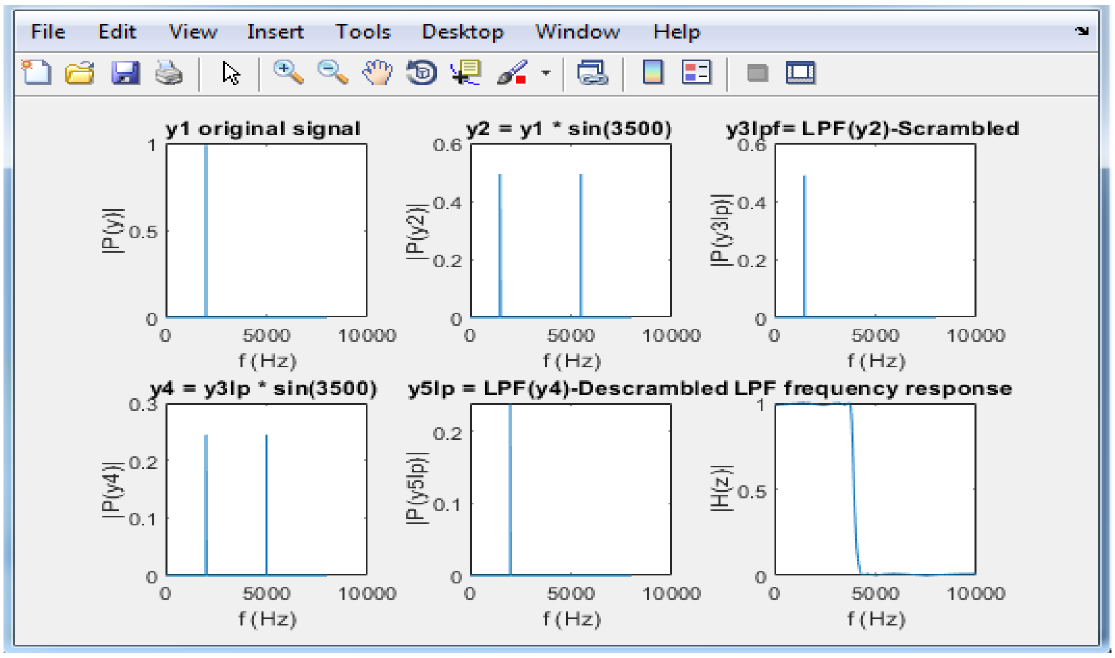


Figure (3.18) FFT for test 1

3.5.2 Test - 2

The same as above but the input is a recorded .wav format voice. Using IIR filter as shown in figure (3.19) and (3.20).

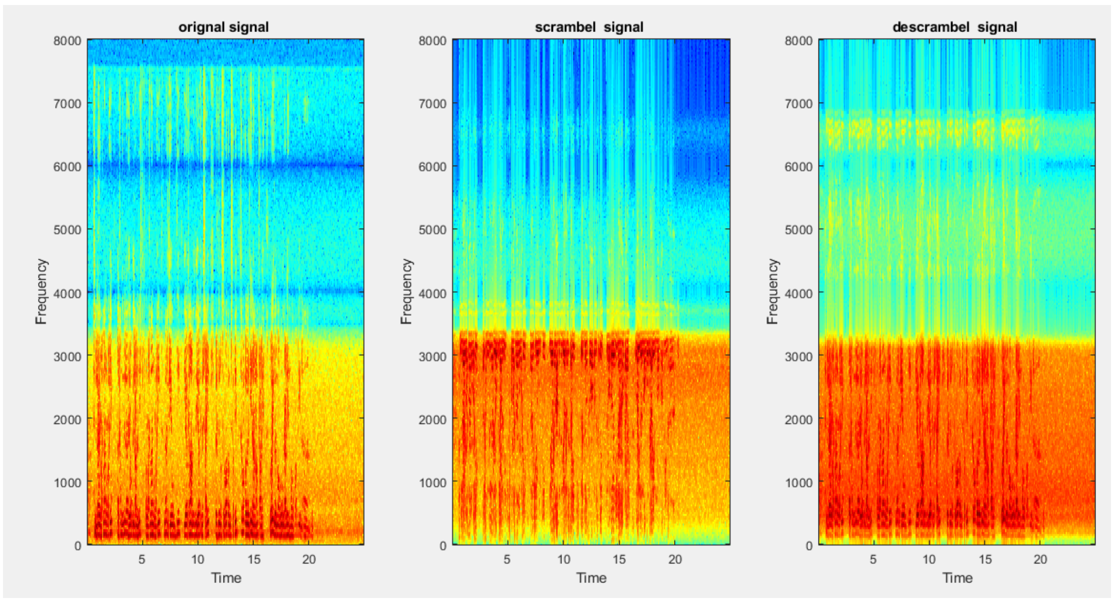


Figure (3.19) Spectrum analyzer for test 2 using IIR filter

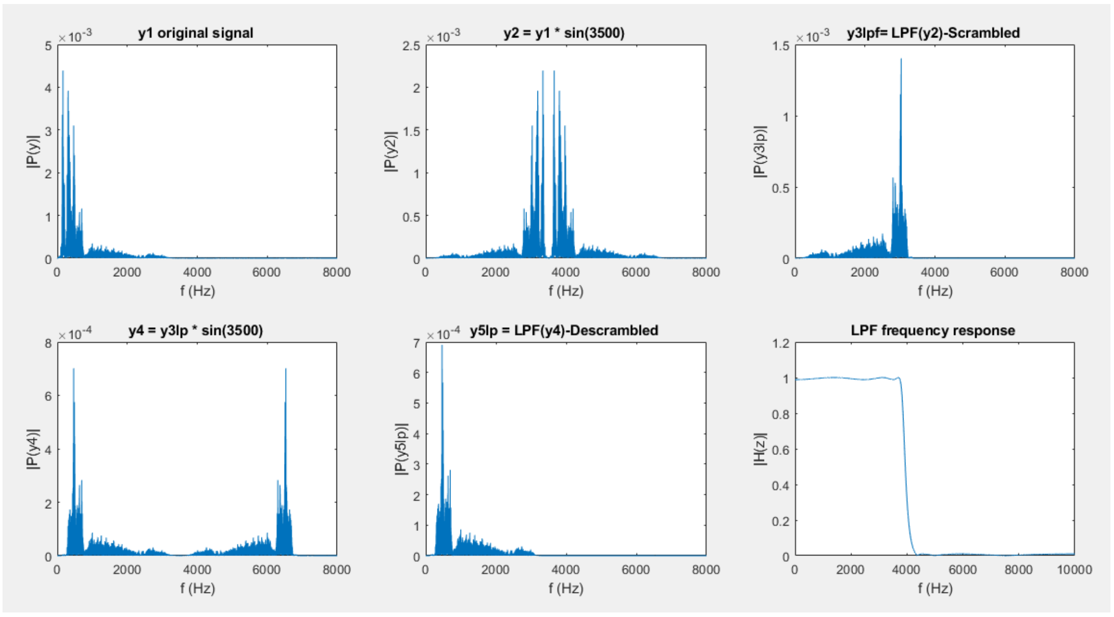


Figure (3.20) FFT for test 2 using IIR filter

3.5.3 Test - 3

The same as Test2 but using FIR filter as in figure (3.21) and (3.22).

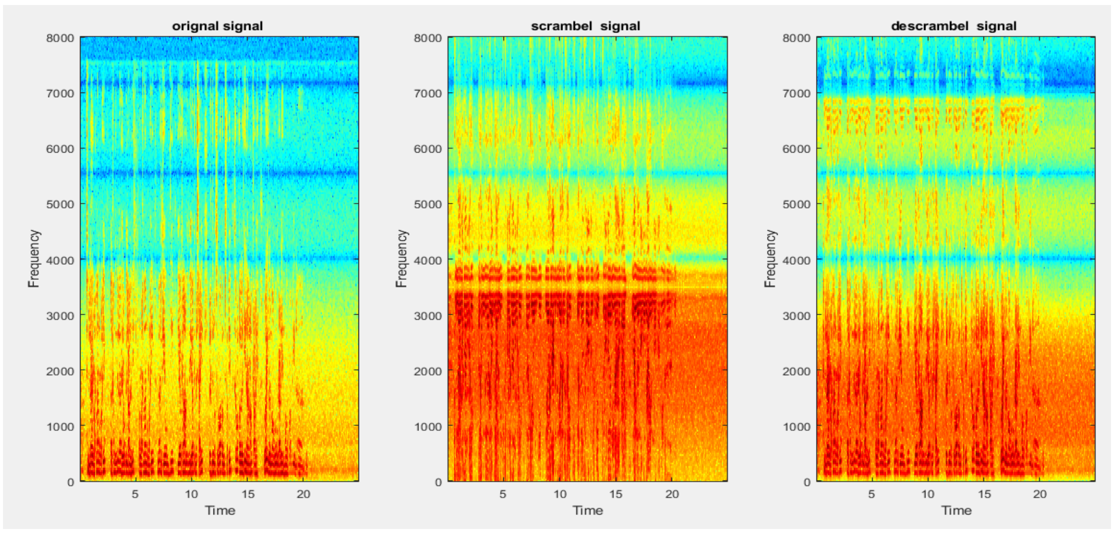


Figure (3.21) Spectrum analyzer for test 3 using FIR filter

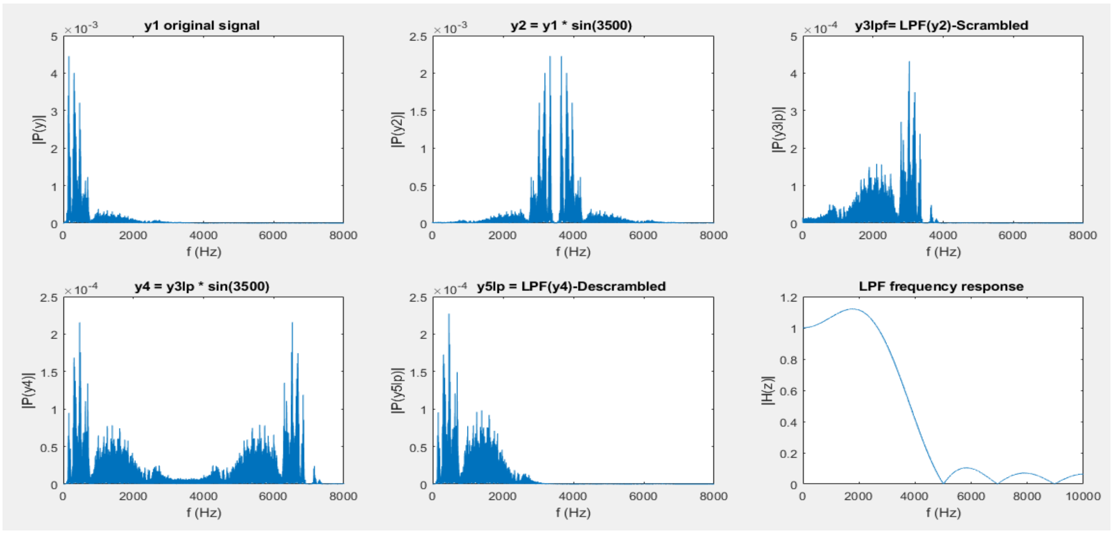


Figure (3.22) FFT for test 3 using FIR filter

3.6 Complete Hardware System

The complete hardware used in this design as show in figure (3.22).

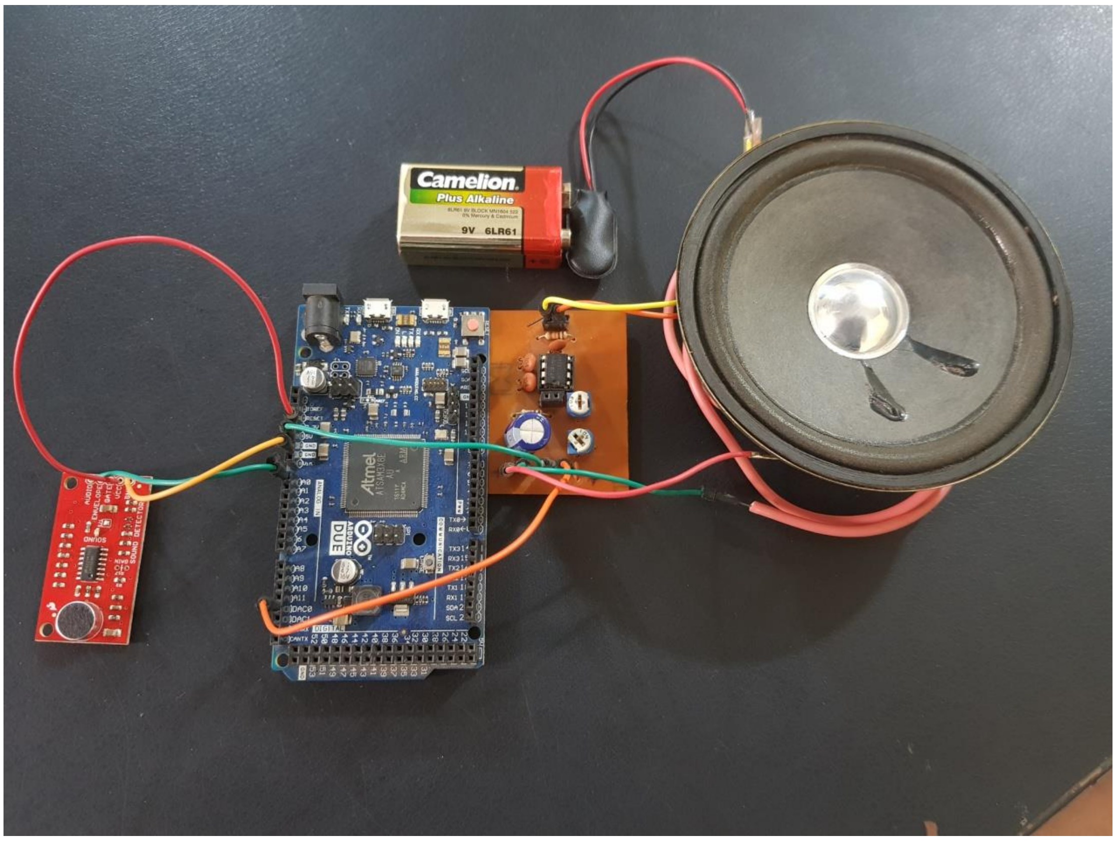


Figure (3.23) Complete hardware design

CHAPTER FOUR

SYSTEM IMPLEMENTATION

CHAPTER FOUR

SYSTEM IMPLEMENTATION

4.1 Introduction

This chapter introduces the system implementation in microcontroller and tests done to verify system functionality and eligibility

4.2 Design Implementation

4.2.1 Frequency Scrambling Implementation

4.2.1.1 Sampling Frequency Generation

16 KHz sampling frequency is used and generated by TC0 timer. TC0 is programmed in WAVE mode with WAVSEL = 10, in this mode the counter is incremented and reset when reached the value set in RC register. TIOA is set when the counter reaches the RC value and cleared when the value equal RA as shown in figure (4.1). So the value loaded in RC to generate a given frequency is calculated as follow.

$$RC\_VALUE = \frac{SYSTEM\_CLOCK}{2 * F_S}\text{, system clock =84mHz} \qquad (4.1)$$

For Sampling frequency = 16000 Hz

RC_VALUE = 2625

RA_VALUE = RC_VALUE/2 = 1312 to have square wave. The programming of timer 0 is done in setup () function.

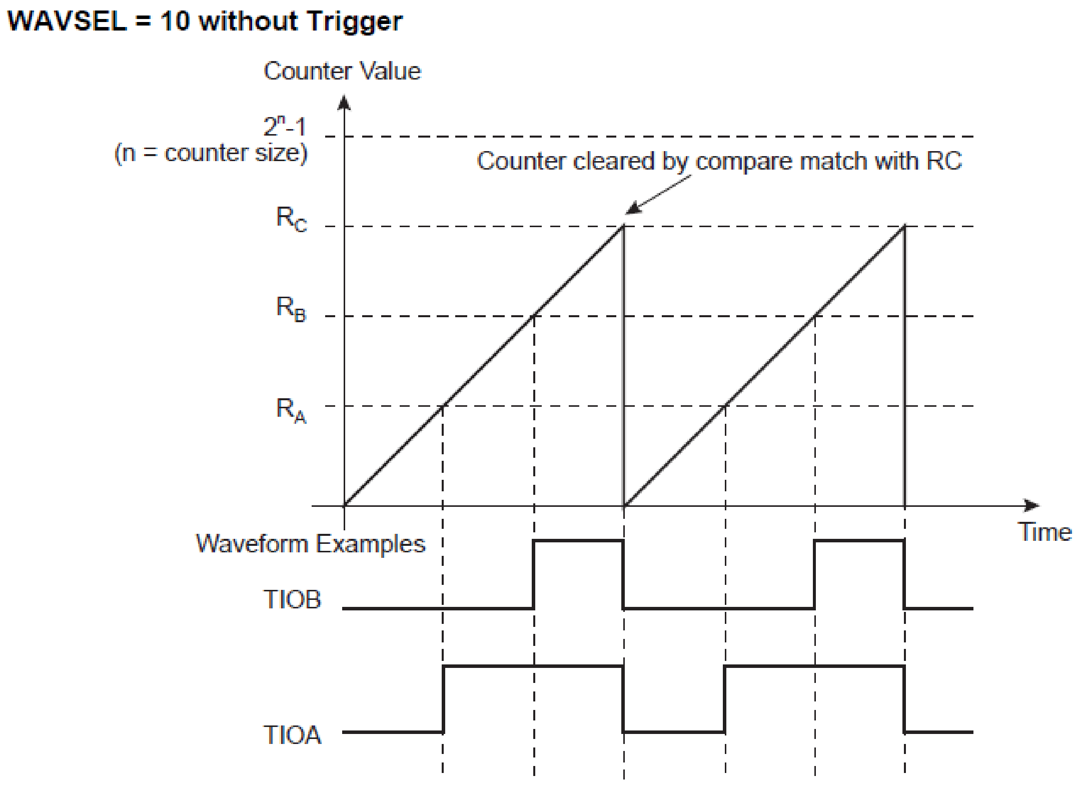


Figure (4.1) Sampling frequency generation

4.2.1.2 Analog to Digital Convertor (ADC)

The ADC channels (ch0 and ch1) are programmed as shown.

- Sampling frequency is 16KHz generated by TC0 as above.
- Interrupt mode when EOC (end of conversion) status is set at the end ADC conversion.
- Enable ch0 and ch1.
- Gain =1 and offset = 0.
- 12 bits resolution.
- Enable AD7 which corresponds to A0 on the board.

4.2.1.3 Digital to analog Convertor (DAC)

DAC channel 0 is programmed as follow

- Trigger by TC0 as ADC (16 KHz).
- Select and enable CH0.
- No interrupt mode (Continuous mode).
- 12 bits resolution.

4.2.1.4 Sine Wave Generation

The digital values of sinusoid of frequency 3500Hz sampled by 16000Hz

$$X = \sin(\frac{2*\pi*3500*n}{16000}) \tag{4.2}$$

Since the sine cycle is defined between 0 and 2π , to get an integer value of n for one cycle

$$\frac{2*\pi*3500*n}{16000} = k*2*\pi \tag{4.3}$$

Where k is an integer represent the number of cycles.

If we choose k=35, n=160.

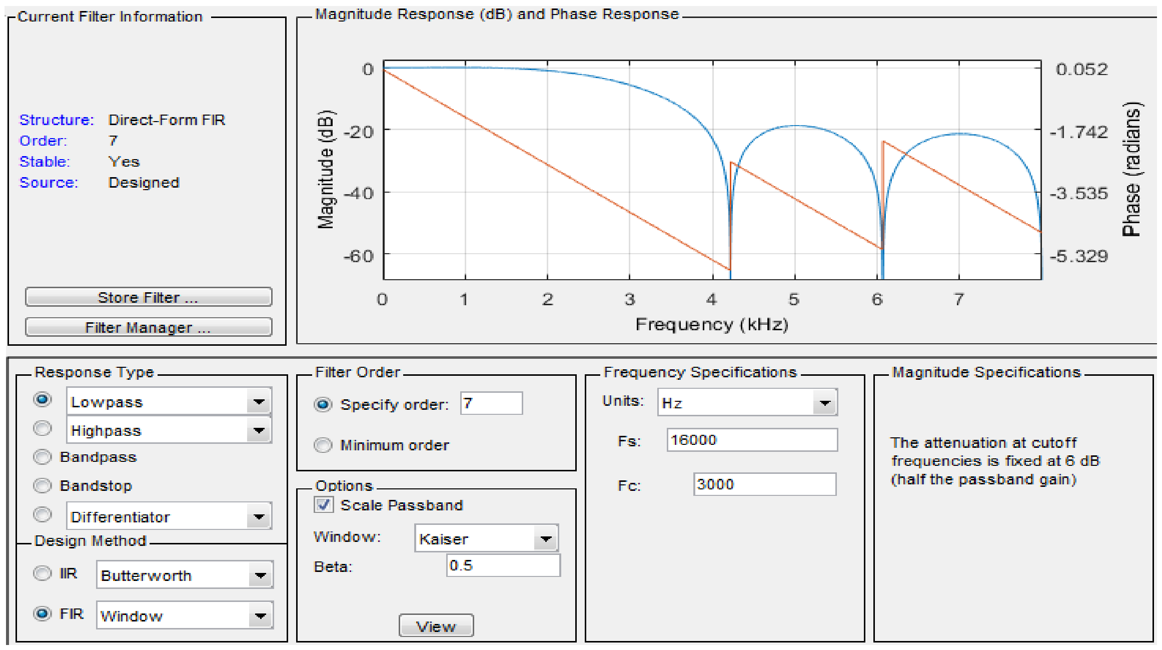
So we need a table to store 160 sinusoid values. The sine wave is generated off line and stored in lookup table, the size of the table =160.

```
for (int i = 0 ; i< 160 ; i++){
    vsine[i] =  sin($\frac{2*\pi*3500*i}{16000}$) ;
}
```

4.2.1.5 Filter Design and Implementation

A MATLAB tool filter Designer is used to calculate the filter coefficients.

-Type FilterDesigner on MATLAB
>> filterDesigner
-after input the filter specification as shown and pressing filter design Order =7, F_S = 16000 Hz , F_C = 3000 Hz , FIR-Window type as shown in figure (4.2) .



Current Filter Information

Structure: Direct-Form FIR
Order: 7
Stable: Yes
Source: Designed

Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator

Design Method: IIR Butterworth [✓ FIR Window]

Filter Order: [✓ Specify order: 7] Minimum order

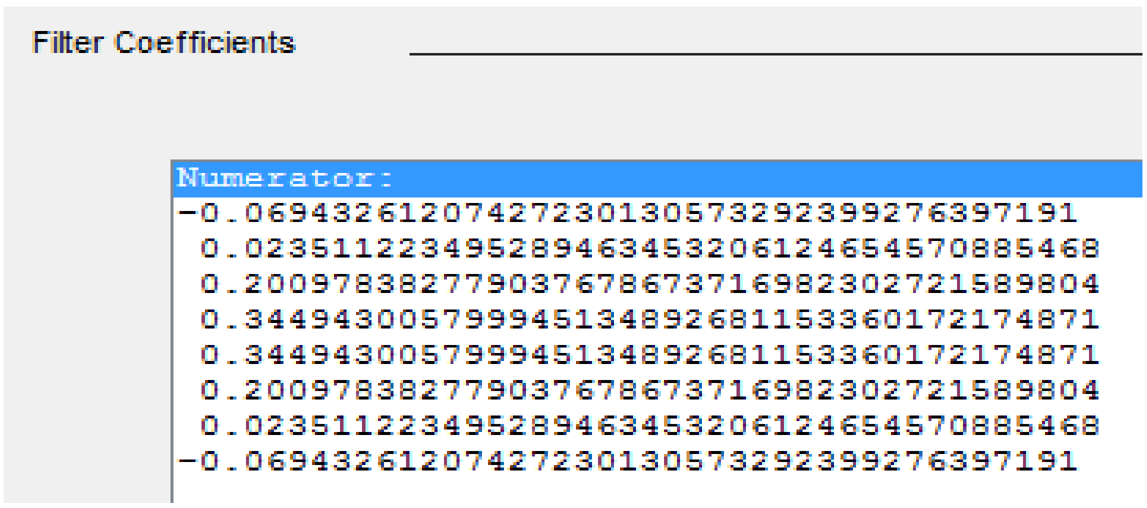
Options: [✓ Scale Passband]
Window: Kaiser
Beta: 0.5

Frequency Specifications
Units: Hz
Fs: 16000
Fc: 3000

Magnitude Specifications
The attenuation at cutoff frequencies is fixed at 6 dB (half the passband gain)

Figure (4.2) Filter design tools using MATLAB

-After choosing the Analysis--- Filter coefficients could be obtained as shown in figure (4.3) below.



Filter Coefficients

```
Numerator:
-0.069432612074272301305732923992763971 91
```

```
Numerator:
-0.06943261207427230130573292399276397191
 0.02351122349528946345320612465457088546 8
```

Figure (4.3) Filter coefficients

So

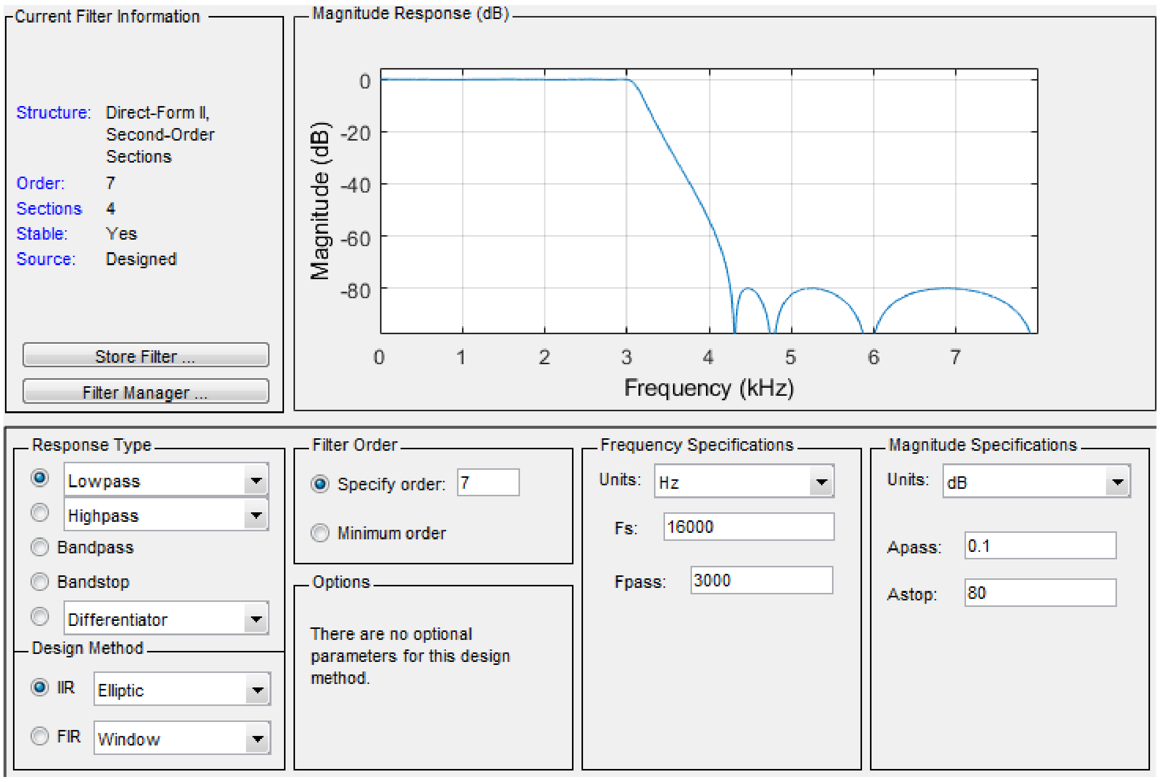
$$H(z)=\frac{Y(z)}{X(z)}=-0.0694(x(z)+x(z)z^{-7})+0.0235(x(z)z^{-1}+x(z)z^{-6})+ \quad (4.4)$$

$$0.2(x(z)z^{-2}+x(z)z^{-5})+0.344(x(z)z^{-3}+x(z)z^{-4})$$

The difference equation for FIR filter design to be implemented in microcontroller is as follow.

$$Y(n) = -0.0694(X(n) + X(n-7)) + 0.02351(X(n-1) + X(n-6)) + 0.2(X(n-2) + X(n-5)) + 0.344(X(n-3) + X(n-4)) \tag{4.5}$$

Noting that X (n) presents current sample and X (n-1) presents previous sample and so on. To implement inside the microcontroller, we need only 4 multiplications. Better specification could be obtained if we use IIR filter with the same specifications. As figure (4.4) below, this will reduce the interference between the lower and upper bands after multiplications with sine



Current Filter Information

Structure: Direct-Form II, Second-Order Sections
Order: 7
Sections: 4
Stable: Yes
Source: Designed

Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator

Design Method: [✓ IIR] Elliptic; FIR Window

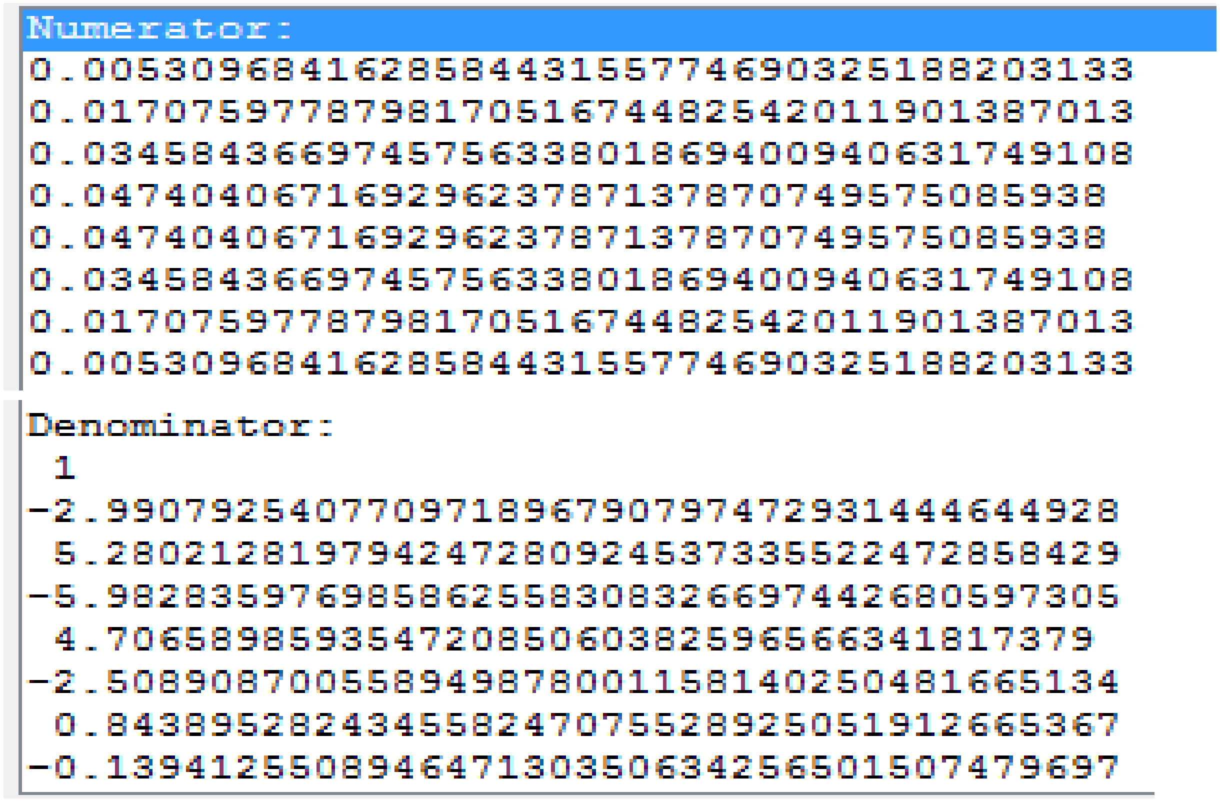
Filter Order: [✓ Specify order: 7] Minimum order

Options: There are no optional parameters for this design method.

Frequency Specifications
Units: Hz
Fs: 16000
Fpass: 3000

Magnitude Specifications
Units: dB
Apass: 0.1
Astop: 80

Figure (4.4) IIR filter design tool using Matlab



```
Numerator:
0.005309684162858443155774690325188203133
0.017075977879817051674482542011901387013
0.034584366974575633801869400940631749108
0.047404067169296237871378707495750859 38
0.047404067169296237871378707495750859 38
0.034584366974575633801869400940631749108
0.017075977879817051674482542011901387013
0.005309684162858443155774690325188203133
```

```
Denominator:
 1
-2.990792540770971896790797472931444644928
 5.280212819794247280924537335522472858429
-5.982835976985862558308326697442680597305
 4.70658985935472085060382596566341817379
-2.508908700558949878001158140250481665134
 0.843895282434558247075528925051912665367
-0.139412550894647130350634256501507479697
```

Figure (4.5) IIR filter coefficients

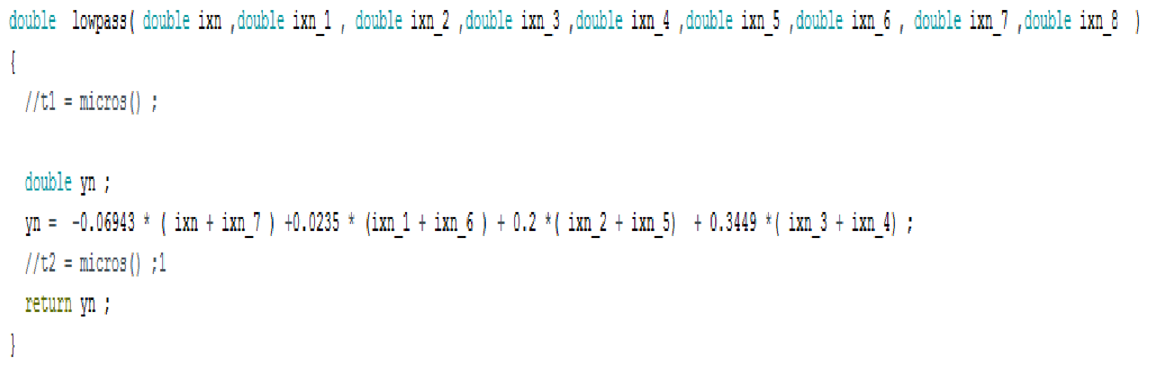
$$H(z) = \frac{y(z)}{x(z)} = \qquad (4.6)$$

$$\frac{0.0053(1+z^{-7}) + 0.017(z^{-1}+z^{-6}) + 0.034(z^{-2}+z^{-5}) + 0.0474(z^{-3}+z^{-4})}{1 - 2.99z^{-1} + 5.28z^{-2} - 5.98z^{-3} + 4.76z^{-4} - 2.508z^{-5} + 0.8438z^{-6} - 0.1394z^{-7}}$$

The difference equation.

$$\begin{aligned} y(n) &= 0.0053(x(n) + x(n-7)) + 0.017(x(n-1) + x(n-6)) \\ &+ 0.0345(x(n-2) + x(n-5)) + 0.0474(x(n-3) + x(n-4)) \\ &+ 2.99y(n-1) - 5.28y(n-2) + 5.982y(n-3) - 4.706y(n-4) \\ &+ 2.508y(n-5) + 0.1394y(n-6) \end{aligned} \qquad (4.6)$$

Which could not be implemented inside the ARDUINO due since need 11 multiplications and could not executed in time less than 62.5μs for 16000Hz sampling. So FIR is used for filter implementation and executed by calling low pass function as shown in figure (4.6).

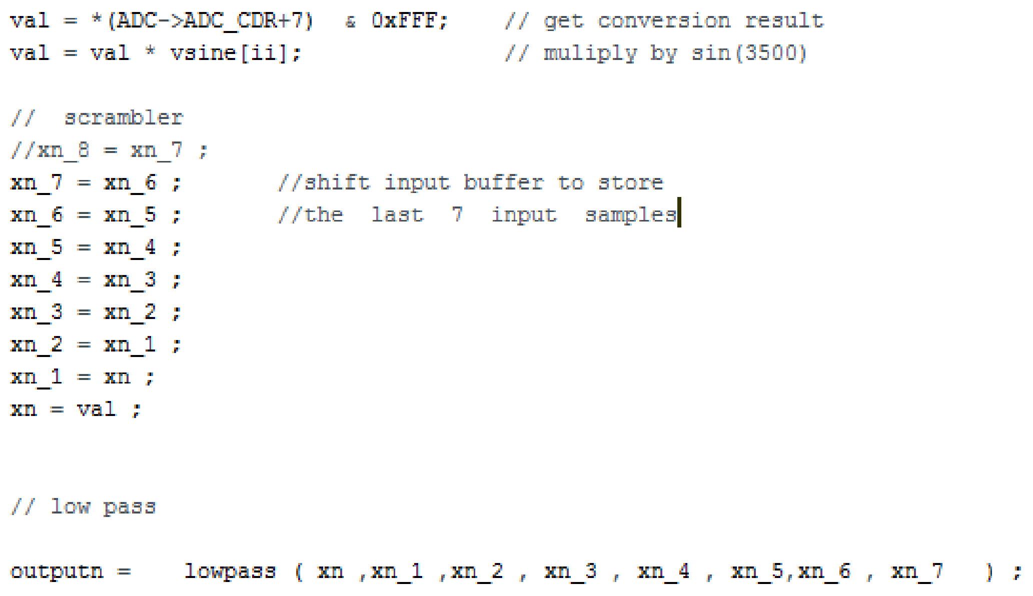


```
double  lowpass( double ixn ,double ixn_1 , double ixn_2 ,double ixn_3 ,double ixn_4 ,double ixn_5 ,double ixn_6 , double ixn_7 ,double ixn_8  )
{
  //t1 = micros() ;

  double yn ;
  yn =  -0.06943 * ( ixn + ixn_7 ) +0.0235 * (ixn_1 + ixn_6 ) + 0.2 *( ixn_2 + ixn_5)  + 0.3449 *( ixn_3 + ixn_4) ;
  //t2 = micros() ;1
  return yn ;
}
```

Figure (4.6) Low pass filter function

4.2.1.6 Frequency Inversion Implementation

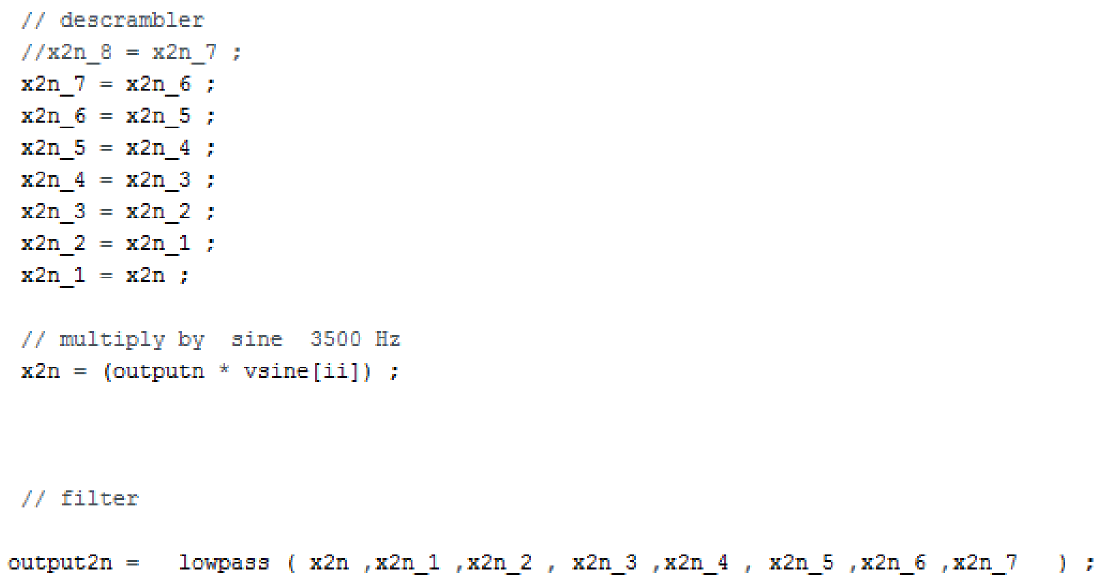
The frequency inversion is implemented by sine multiplication followed by low pass filter to give scrambling channel. The descramble channel is implemented by multiplication of scramble channel by sine and then filtering with low pass filter. Scrambling implementation is done inside ADC interrupt service routine (void ADC_Handler (void)) as shown in figure (4.7).



```
val = *(ADC->ADC_CDR+7)  & 0xFFF;    // get conversion result
val = val * vsine[ii];               // muliply by sin(3500)

//  scrambler
//xn_8 = xn_7 ;
xn_7 = xn_6 ;       //shift input buffer to store
xn_6 = xn_5 ;       //the  last  7  input  samples
xn_5 = xn_4 ;
xn_4 = xn_3 ;
xn_3 = xn_2 ;
xn_2 = xn_1 ;
xn_1 = xn ;
xn = val ;


// low pass

outputn =    lowpass ( xn ,xn_1 ,xn_2 , xn_3 , xn_4 , xn_5,xn_6 , xn_7   ) ;
```

Figure (4.7) Frequency inversion scrambling implementation

And for descramble channel as shown in figure (4.8) below.



```
// descrambler
//x2n_8 = x2n_7 ;
x2n_7 = x2n_6 ;
x2n_6 = x2n_5 ;
x2n_5 = x2n_4 ;
x2n_4 = x2n_3 ;
x2n_3 = x2n_2 ;
x2n_2 = x2n_1 ;
x2n_1 = x2n ;

// multiply by  sine  3500 Hz
x2n = (outputn * vsine[ii]) ;



// filter

output2n =   lowpass ( x2n ,x2n_1 ,x2n_2 , x2n_3 ,x2n_4 , x2n_5 ,x2n_6 ,x2n_7   ) ;
```

Figure (4.8) Frequency inversion descrambling implementation

4.2.2 Time Permutation Implementation

The algorithm for time permutation is also implemented in ADC_Handler ISR. The algorithm implemented as follow in figure (4.9).

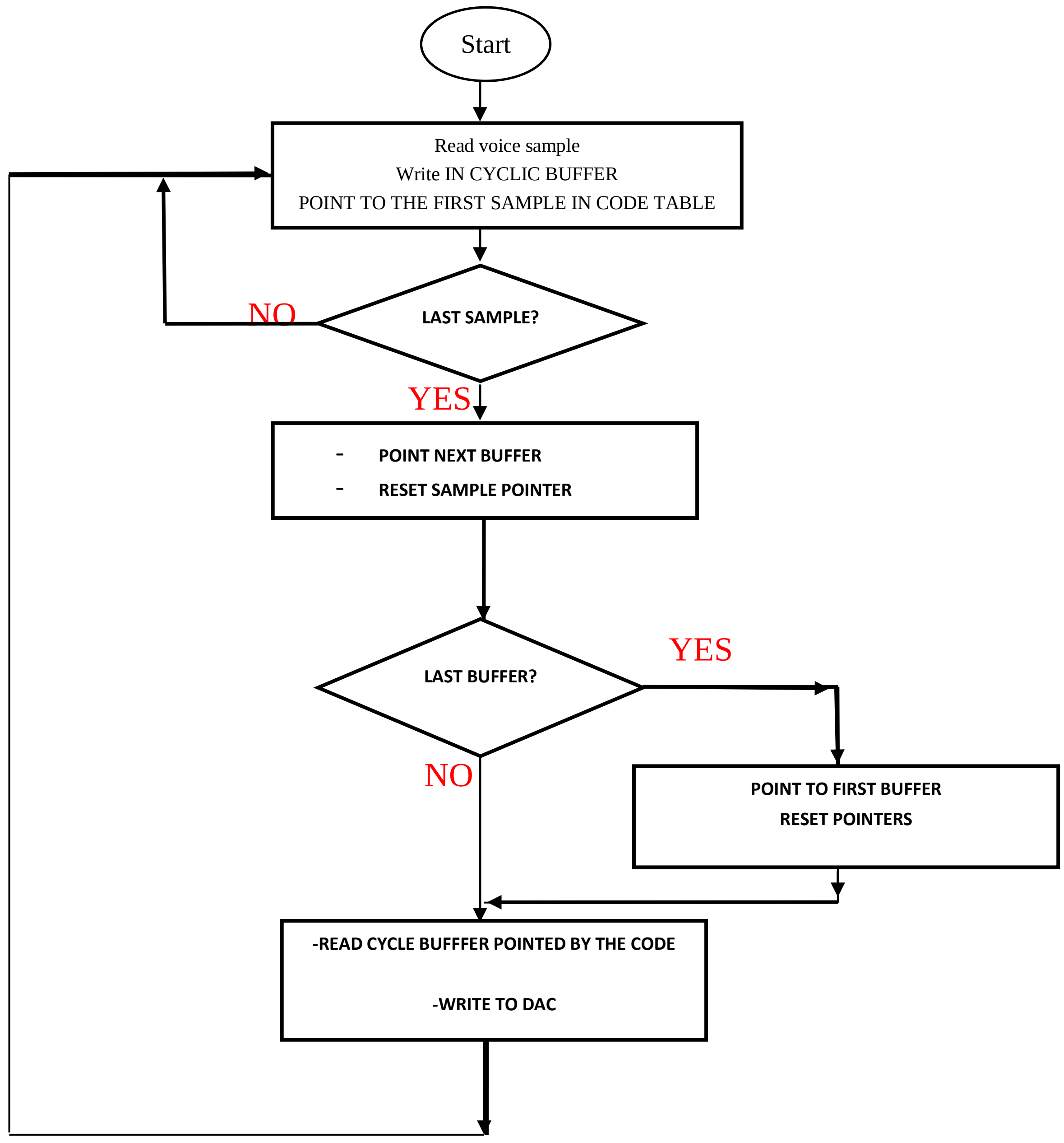


Figure (4.9) Flow chart for time permutation

4.2.3 ADC Interrupts Service Routine (ISR)

All the processing of frequency inversion and time scrambling have to be done by this ISR. Noting that the execution time have to less than 125 MS. When using 16KHz sampling and 62.5 For frequency inversion the following are executed.

- Read ADC value.
- Multiply by Sine.

- Low Pass filter to produce scramble channel.
- Multiply scramble channel by sine.
- Low pass filter to produce descramble channel.

For time scrambling the following are executed.

- A 16 buffers each with 1024 samples are reserve for DAC samples [16] [1024].
- The writing sequence {samples [0] [1024], samples [1] [1024], samples [2] [1024], samples [15] [1024], samples [0] [1024],}.
- The time permutation is done by controlling the reading sequence from the buffers to DAC for the output channel

For test the sequence is {Samples [15] [1024], samples [14] [1024], samples [13] [1024], samples [0] [1024], samples [15] [1024] ...}.

CHAPTER FIVE

RESULTS AND DISCUSSIONS

CHAPTER FIVE

RESULTS AND DISCUSSIONS

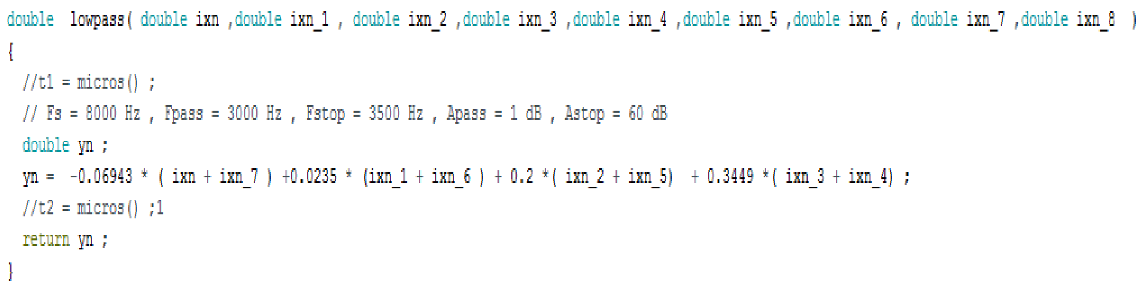
5.1 Introduction

This chapter introduces the system validation tests and results using arduino microcontroller program.

5.2 Static Tests

An ARDUINO software is written to read the voice from the MIC with 16KHz using 7-order FIR filter with cut off frequency of 3000Hz, the sine wave used is 3500Hz.

Noting that we used FIR linear phase filter to have symmetry in the coefficients and need only 4 multiplications for filter implementation as shown below figure (5.1).



```
double  lowpass( double ixn ,double ixn_1 , double ixn_2 ,double ixn_3 ,double ixn_4 ,double ixn_5 ,double ixn_6 , double ixn_7 ,double ixn_8  )
{
  //t1 = micros() ;
  // Fs = 8000 Hz , Fpass = 3000 Hz , Fstop = 3500 Hz , Apass = 1 dB , Astop = 60 dB
  double yn ;
  yn =  -0.06943 * ( ixn + ixn_7 ) +0.0235 * (ixn_1 + ixn_6 ) + 0.2 *( ixn_2 + ixn_5)  + 0.3449 *( ixn_3 + ixn_4) ;
  //t2 = micros() ;1
  return yn ;
}
```

Figure (5.1) Lowpass filter implemntation in Arduino

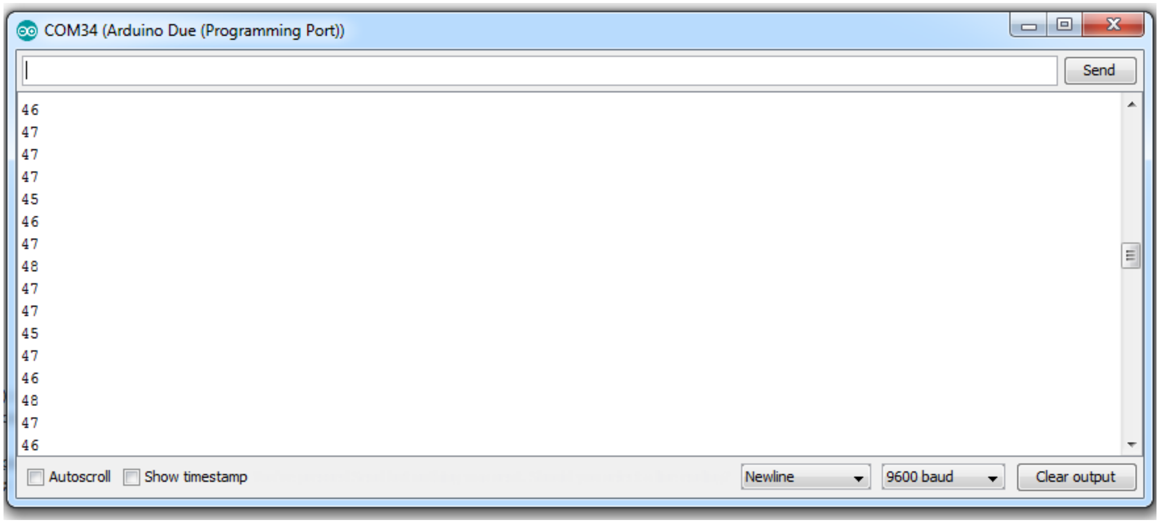
5.2.1 Test-1 Execution Time Calculation

To calculate the execution time for scrambling (multiplication with sine + filter) and descrambling (multiplication by sine and filter) we used the instruction micros (). Which available in ARDUINO software to store the time at the start of ADC ISR and the time at the end of ISR with Microsecond resolution (µs).

t1 = micros ();

t2 = micros ();

The execution time = Serial.println(t2-t1) =47 µs; which is less than 62.5 µs. Which indicate the program could be executed in real time as shown on figure (5.2) after loading and executing the program in ARDUINO DUE.



```
46
47
47
47
45
46
47
48
47
47
45
47
46
48
47
46
```

Figure (5.2) Scrambling and descrambling execution time

5.2.1.1 Test-2 ADC IT RATE

Since 16KHz is used as sampling frequency which means that the time between two executive interrupts must equal to 62.5 µs which calculated as follow.

- Store the previous time at the start of ISR

t3 = t1; // store previous time

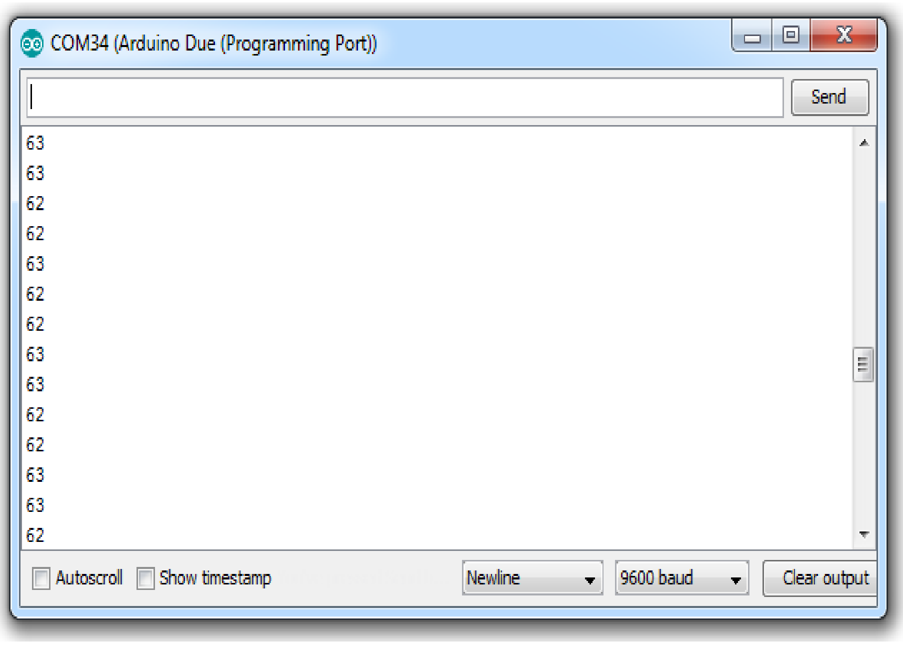
-store the current time

t1 = micros (); // store current time

-The time between two consecutive interrupts = t4 = t1 - t3

Serial.println(t4);

As shown in figure (5.3) when executing the program on ARDUINO the time was 62ms.


COM34 (Arduino Due (Programming Port))

```
63
63
62
62
63
62
62
63
63
62
62
63
63
62
```

Figure (5.3) Sample rate check

5.2.1.2 Test 3 Sine Wave Generation

The digital value for sin (2*π*3500*t) is digitized as follow using 16KHz sampling frequency.

$$\sin(n) = \frac{2*\pi*3500*n}{16000} \tag{5.1}$$

As mentioned before the length of look up table to store sine value is 160 and calculated and stored in a table as.

```
 for ( int i = 0 ; i< 160 ; i++){
      vsine[i] =  sin(0.4375 * pi*i)  * 4096;
//      Serial.print(vsine[i] );
//      Serial.print(" ; " );
//     Serial.println(i );
 }
}
```

The sine of sample n= 180 is equal to sample (180-160 = 20).

Sine (0.4375*π*20) = 0.707

Sine (0.4375* π*180) = 0.707

5.2.1.3 Time Scrambling Implementation Test

The input is stored in circular buffer (0---15) defined in the program as two-dimension variable. (volatile int samples [16] [BUFSIZE] ;) where BUFSIZE = 1024). The read from the circular buffer is done as define in the code defined in the program as.

volatile int code [16] = {15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0};

The scrambled time output written to DAC is defined as

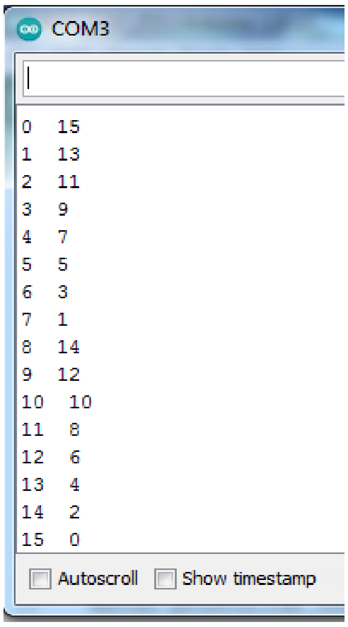
dac_write (samples [obufn] [sptr]);

Where obufn = code[k] , k = 0------15

Test-1

Time scrambling buffer sequence writes and read In the loop function of ARDUINO.Write buffer is printed whenever a new buffer started as shown in figure (5.4). In column one and the same thing for read buffer in column two.

So the sequence of write buffer is sequential (normal voice) while the sequence of read buffer is random as in table code.



```
0   15
1   13
2   11
3   9
4   7
5   5
6   3
7   1
8   14
9   12
10   10
11   8
12   6
13   4
14   2
15   0
```

Figure (5.4) Time scramble check

5.3 Arduino Real Time Test

A MIC is connected to pin A0 of ARDUINO board which is connected channel 0 of ADC, with each sample of input audio the following are executed

1- The output of ADC is multiplied by sinusoid of 3500Hz and pass to low pass filter as explained in chapter 3 to give scramble channel (Out- 2 signal as shown in the diagram).

2- out-2 is multiplied by the same sinusoid and pass to low pass filter with the same specification as above to give descramble channel (Out-3 signal).

3- The output from ADC is written in memory buffers sequentially and read in a sequence as in code table as explained before to give Out-4 signal.

4- The input to DAC is chosen manually in the program for each run and download as follow.

   - connect to out-1 to listen and evaluate ADC and DAC and the effect on voice output.

- connect to out-2 to listen and evaluate scrambling.
- connect out-3 to listen and evaluate descrambling.
- connect to out-4 to listen and evaluate time scrambling .

As in the figure (5.5) and (5.6) below.

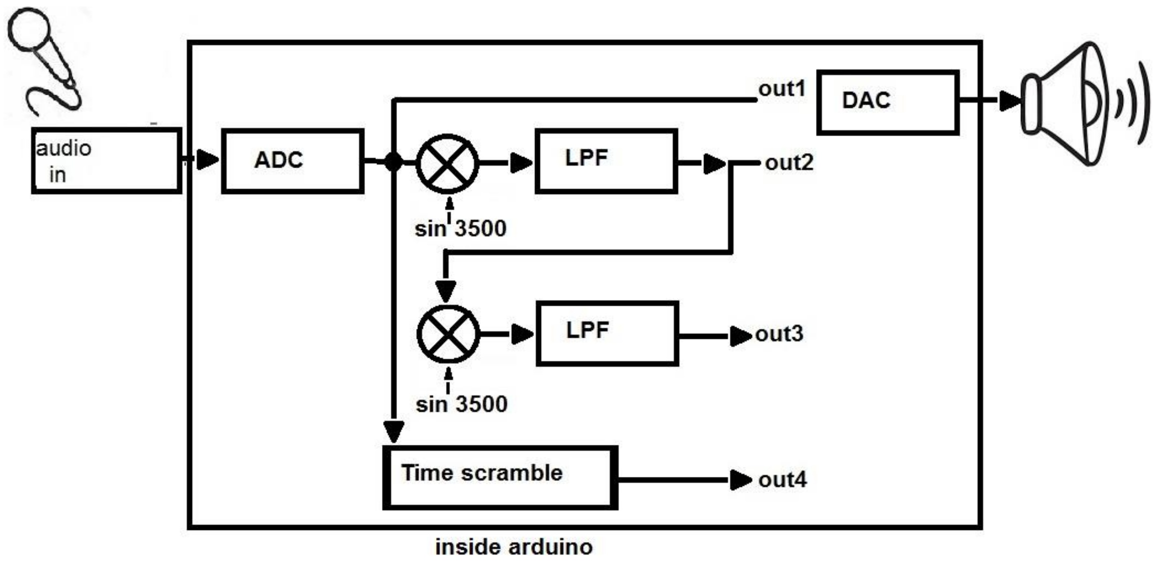


Figure (5.5) Real time test block diagram.

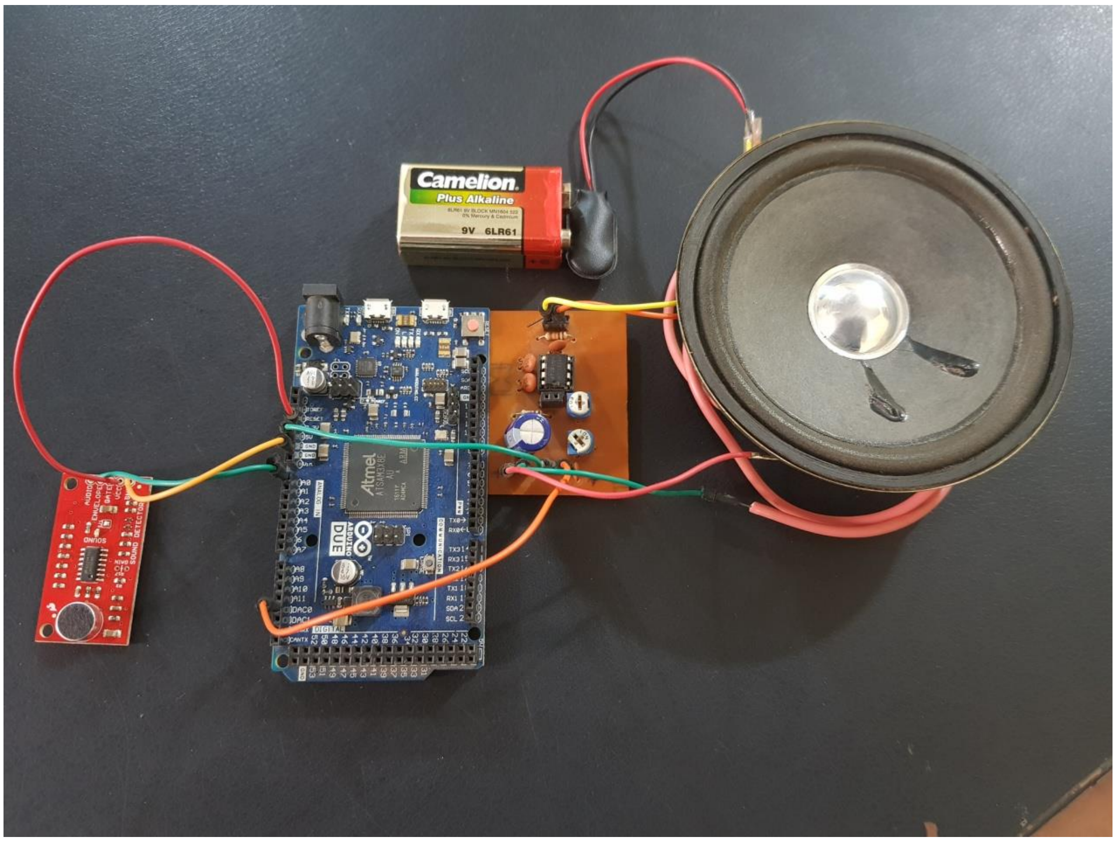


Figure (5.6) Complete hardware design

CHAPTER SIX

CONCLUSION AND FUTURE WORK

CHAPTER SIX

CONCLUSION AND FUTURE WORK

6.1 Conclusion

The real time implementation of frequency inversion could be achieved using advanced microcontroller with specifications oriented to real time application like ADC, DAC working with DMA and interrupt mode with low latency time.

The different advanced techniques based on frequency inversion like Rolling Code Scrambling, Frequency Hopping and frequency sweeping could be achieved by changing the frequency center through software which gives high Flexibility in changing the code.

The time permutation scrambling was achieved through address manipulation of memory buffers for read and write, the data from ADC are write in circular buffers while the read from buffers is done randomly as stored in code table without additional burden on microcontroller.

System implementation based on MATLAB (Simulink and .m files) show some delay between the voice input and the output using the PC. Although using more advanced PC with multi-processors techniques could reduce the delay but it still not suitable for portable real time operation. Using IIR filter in frequency inversion will give better result but need more multiplications than FIR filters and more execution time, so FIR is used for microcontroller implementation.

The effectiveness of scrambling is determined by the amount of residual intelligibility which indicates the amount of redundant information which helps in easier recovery of the original signal.

The parameters used to measure the residual intelligibility need more processing which could not be implemented in real time using the proposed hardware. The intelligibility is evaluated by using trained and untrained human listeners to listen to the scramble audio.

6.2 Future Work

A more accurate and higher sampling frequency with higher filter order could be achieved using microcontroller or DSP with floating point operation.

More emphasis has to be done on synchronization between scrambler and descrambler specially for time domain scrambling. More research and analysis for the design and implementation of voice scrambling without the need for synchronization specially the implementation of multi-band pass filters based on the processing of voice

The residual intelligibility has to be measured for different algorithms used in this work. Which could be done by logging the scrambled audio on SD card in real time and later transfer to PC for calculation and evaluation, or transferring the scrambled audio to PC through high speed link between the ARDUINO Due board and the PC.

The system could be implemented using System on Chip technology (SOC). Which contain processor and logic on the same chip. For example, Xilinx Zynq 7000 family which contain two processors working in parallel with 1 GHz and 1.5 million gates for more professional system.

REFRENCES

REFRENCES

[1] T. Al-Sabbagh, "Front End to Back End Speech Scrambler," International Journal of Computing and Network Technology, vol. 7, pp. 65-70, 01/08 2020.

[2] N. Hayati, Y. Suryanto, K. Ramli, and M. Suryanegara, End-to-End Voice Encryption Based on Multiple Circular Chaotic Permutation, 2019.

[3] D. G and J. J, "An efficient voice scrambling technique for next generation communication systems," International Journal of Engineering and Technology, vol. 8, 02/01 2016.

[4] G. Bagwe, D. Apsingekar, S. Gandhare, and S. Pawar, Voice encryption and decryption in telecommunication, 2016.

[5] F. Enache, D. Deparateanu, T. Oroian, F. Popescu, and I. Vizitiu, "Theoretical and practical implementation of scrambling algorithms for speech signals," in 2015 7th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), 2015, pp. S-49-S-52.

[6] L. A. A.-R. Samir J. Mohammad, "Modified Scrambling Based on permutation function Matrix and 2D-DMWT with QPSK," 2014.

[7] M. K. Anjana D S, "Frequency Speech Scrambler based on Hartley Transform and OFDM Algorithm," 2013.

[8] L. H. M. Al-Saadi, "Analog Speech Encryption Based On Biorthogonal Transforms," 2013.

[9] P. V. R. I. Hemlata Kohad, Dr.M.A.Gaikwad, "An Overview of Speech Encryption Techniques," 2012.

[10] R. G. T. Francisco Assis de O. Nascimento, "Frequency Speech Scrambler Based on the Hartley Transform and the Insertion of Random Frequency Components," 2012.

[11] J. A. a. N. Ikram, "Frequency-domain speech scrambling/descrambling techniques implementation and evaluation on DSP," 7th International Multi Topic Conference, 2003. INMIC 2003., Islamabad, Pakistan, 2003, pp. 44-48, doi: 10.1109/INMIC.2003.1416613., 2003 2003.

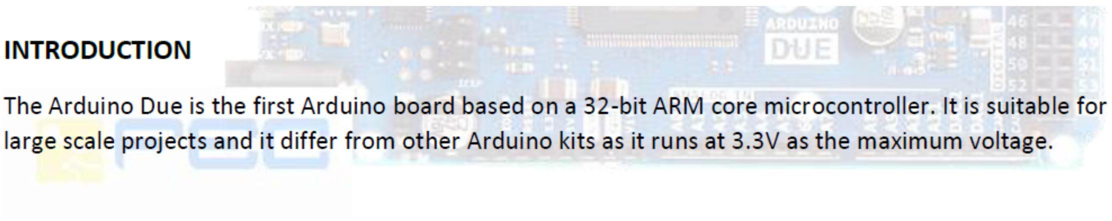
[12] Frederick Huang, "United States Patent, Patent Number: 4,773,092," 1998.

[13] F. Huang and E. V. Stansfield, "Time sample speech scrambler which does not require synchronization," IEEE Transactions on Communications, vol. 41, pp. 1715-1722, 1993.
[14] H. S. Malvar, "A TMS320C25-based telephone scrambler using fast-computable filter banks," 1992.
[15] A. Matsunaga, M. Ohkawa, K. Koga, and K. Sakurai, "A full-duplex analog speech scrambler using the FFT technique and its performance," Electronics and Communications in Japan (Part III: Fundamental Electronic Science), vol. 73, pp. 85-97, 1990.
[16] S. Udalov, "MICROPROCESSOR-BASED ANALOG VOICE SCRAMBLING TECHNIQUES," 1979.

APPENDICES

Appendix A

Arduino Due



INTRODUCTION

The Arduino Due is the first Arduino board based on a 32-bit ARM core microcontroller. It is suitable for large scale projects and it differ from other Arduino kits as it runs at 3.3V as the maximum voltage.

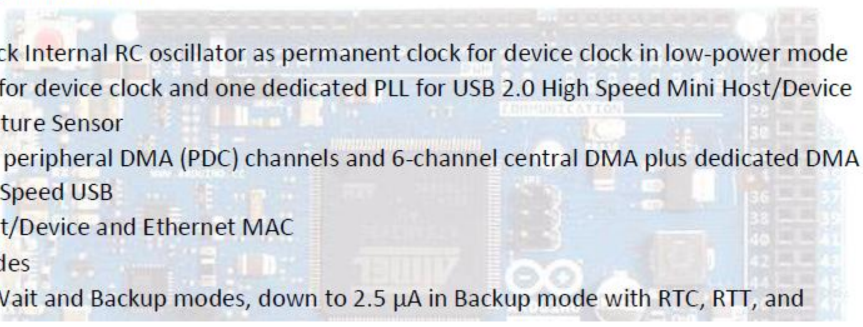
ARDUINO DUE PHYSICAL COMPONENTS

It based on **Atmel SAM3X8E ARM Cortex-M3** microcontroller is a member of a family of Flash microcontrollers based on the high performance 32-bit ARM Cortex-M3 RISC processor. It operates at a maximum speed of 84 MHz and up to 512 Kbytes of Flash memory and up to 100 Kbytes of SRAM. And its features as follow

- Core
  - ARM Cortex-M3 revision 2.0 running at up to 84 MHz
  - Memory Protection Unit (MPU)
  - 24-bit SysTick Counter
  - Nested Vector Interrupt Controller

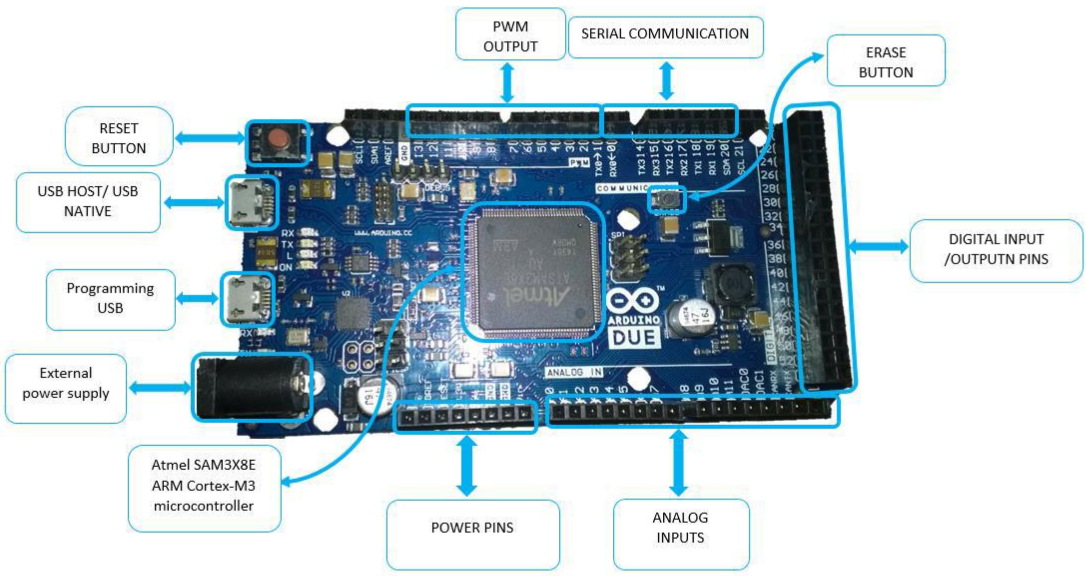
- Memories
  - 256 to 512 Kbytes embedded Flash, 128-bit wide access, memory accelerator, dual bank
  - 32 to 100 Kbytes embedded SRAM with dual banks
  - 16 Kbytes ROM with embedded bootloader routines (UART, USB) and IAP routines
  - Static Memory Controller (SMC): SRAM, NOR, NAND support. NFC with 4 Kbyte RAM buffer and ECC

- System
  - Embedded voltage regulator for single supply operation
  - Power-on-Reset (POR), Brown-out Detector (BOD) and Watchdog for safe reset
  - Quartz or ceramic resonator oscillators: 3 to 20 MHz main and optional low power 32.768 kHz for RTC or device clock
  - High precision 8/12 MHz factory trimmed internal RC oscillator with 4 MHz default frequency for fast device
- startup
  - Slow Clock Internal RC oscillator as permanent clock for device clock in low-power mode
  - One PLL for device clock and one dedicated PLL for USB 2.0 High Speed Mini Host/Device
  - Temperature Sensor
  - Up to 17 peripheral DMA (PDC) channels and 6-channel central DMA plus dedicated DMA for High-Speed USB
  - Mini Host/Device and Ethernet MAC
- Low-power Modes
  - Sleep, Wait and Backup modes, down to 2.5 µA in Backup mode with RTC, RTT, and



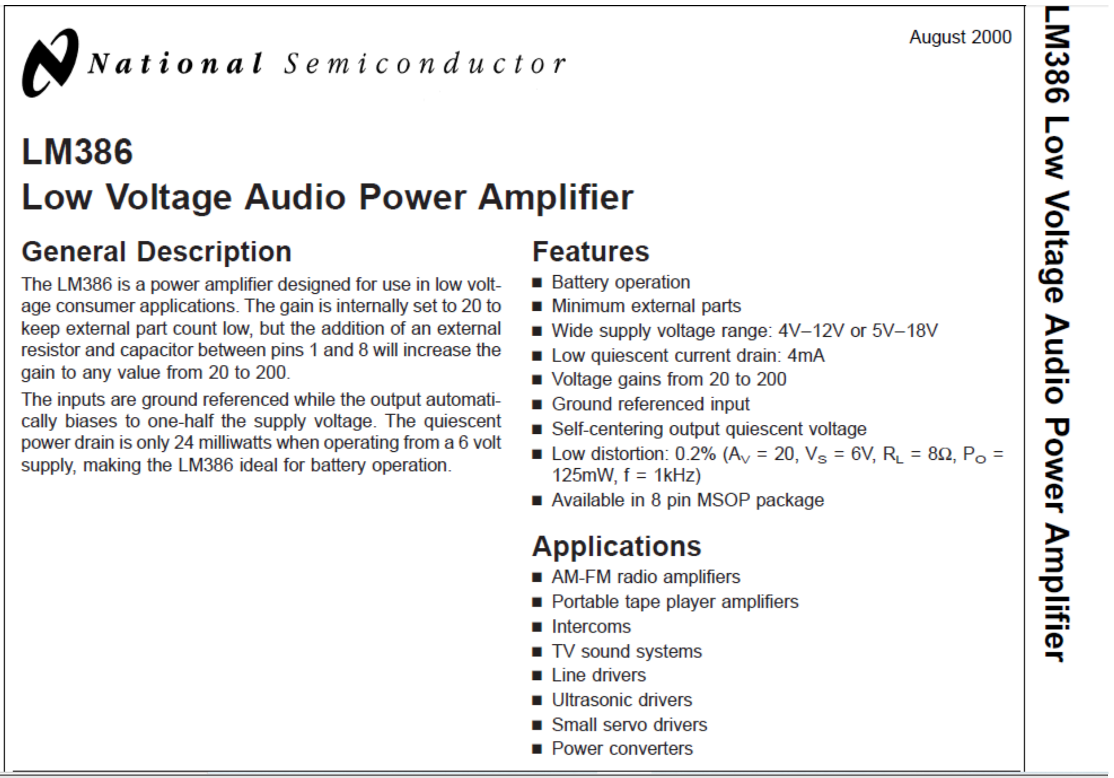
ARDUINO DUE FEATURES

Microcontroller	AT91SAM3X8E
Operating Voltage	3.3V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-16V
Digital I/O Pins	54 (of which 12 provide PWM output)
Analog Input Pins	12
Analog Output Pins	2 (DAC)
Total DC Output Current on all I/O lines	130 mA
DC Current for 3.3V Pin	800 mA
DC Current for 5V Pin	800 mA
Flash Memory	512 KB all available for the user applications
SRAM	96 KB (two banks: 64KB and 32KB)
Clock Speed	84 MHz
Length	101.52 mm
Width	53.3 mm
Weight	36 g


PWM OUTPUT
SERIAL COMMUNICATION
ERASE BUTTON
RESET BUTTON
USB HOST/ USB NATIVE
DIGITAL INPUT /OUTPUTN PINS
Programming USB
External power supply
Atmel SAM3X8E ARM Cortex-M3 microcontroller
POWER PINS
ANALOG INPUTS

Appendix B

Lm386 Low Voltage Audio Power Amplifier



National Semiconductor

August 2000

LM386 Low Voltage Audio Power Amplifier

LM386
Low Voltage Audio Power Amplifier

General Description

The LM386 is a power amplifier designed for use in low voltage consumer applications. The gain is internally set to 20 to keep external part count low, but the addition of an external resistor and capacitor between pins 1 and 8 will increase the gain to any value from 20 to 200.

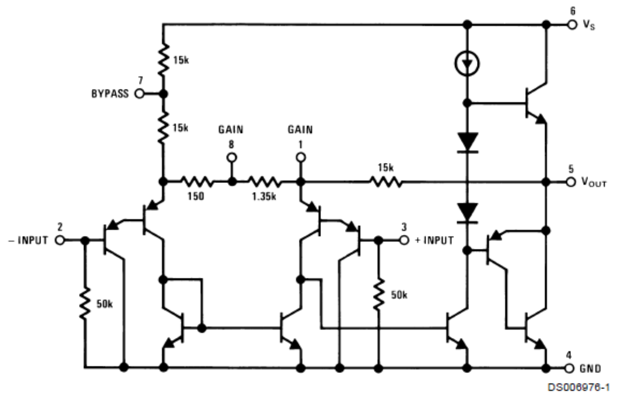
The inputs are ground referenced while the output automatically biases to one-half the supply voltage. The quiescent power drain is only 24 milliwatts when operating from a 6 volt supply, making the LM386 ideal for battery operation.

Features

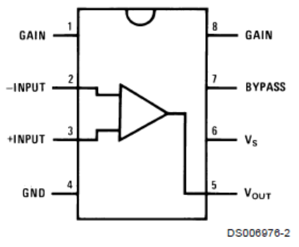
- Battery operation
- Minimum external parts
- Wide supply voltage range: 4V–12V or 5V–18V
- Low quiescent current drain: 4mA
- Voltage gains from 20 to 200
- Ground referenced input
- Self-centering output quiescent voltage
- Low distortion: 0.2% (A_V = 20, V_S = 6V, R_L = 8Ω, P_O = 125mW, f = 1kHz)
- Available in 8 pin MSOP package

Applications

- AM-FM radio amplifiers
- Portable tape player amplifiers
- Intercoms
- TV sound systems
- Line drivers
- Ultrasonic drivers
- Small servo drivers
- Power converters



Small Outline, Molded Mini Small Outline, and Dual-In-Line Packages



Top View
Order Number LM386M-1, LM386MM-1, LM386N-1, LM386N-3 or LM386N-4
See NS Package Number M08A, MUA08A or N08E

Absolute Maximum Ratings (Note 2)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Supply Voltage	
(LM386N-1, -3, LM386M-1)	15V
Supply Voltage (LM386N-4)	22V
Package Dissipation (Note 3)	
(LM386N)	1.25W
(LM386M)	0.73W
(LM386MM-1)	0.595W
Input Voltage	±0.4V
Storage Temperature	−65°C to +150°C
Operating Temperature	0°C to +70°C
Junction Temperature	+150°C
Soldering Information	
Dual-In-Line Package	
Soldering (10 sec)	+260°C
Small Outline Package	
(SOIC and MSOP)	
Vapor Phase (60 sec)	+215°C
Infrared (15 sec)	+220°C

See AN-450 "Surface Mounting Methods and Their Effect on Product Reliability" for other methods of soldering surface mount devices.

Thermal Resistance	
θ_{JC} (DIP)	37°C/W
θ_{JA} (DIP)	107°C/W
θ_{JC} (SO Package)	35°C/W
θ_{JA} (SO Package)	172°C/W
θ_{JA} (MSOP)	210°C/W
θ_{JC} (MSOP)	56°C/W

Electrical Characteristics (Notes 1, 2)

T_A = 25°C

Parameter	Conditions	Min	Typ	Max	Units
Operating Supply Voltage (V_S)					
LM386N-1, -3, LM386M-1, LM386MM-1		4		12	V
LM386N-4		5		18	V
Quiescent Current (I_Q)	V_S = 6V, V_{IN} = 0		4	8	mA
Output Power (P_{OUT})					
LM386N-1, LM386M-1, LM386MM-1	V_S = 6V, R_L = 8Ω, THD = 10%	250	325		mW
LM386N-3	V_S = 9V, R_L = 8Ω, THD = 10%	500	700		mW
LM386N-4	V_S = 16V, R_L = 32Ω, THD = 10%	700	1000		mW
Voltage Gain (A_V)	V_S = 6V, f = 1 kHz		26		dB
	10 µF from Pin 1 to 8		46		dB
Bandwidth (BW)	V_S = 6V, Pins 1 and 8 Open		300		kHz
Total Harmonic Distortion (THD)	V_S = 6V, R_L = 8Ω, P_{OUT} = 125 mW		0.2		%
	f = 1 kHz, Pins 1 and 8 Open				
Power Supply Rejection Ratio (PSRR)	V_S = 6V, f = 1 kHz, C_{BYPASS} = 10 µF		50		dB
	Pins 1 and 8 Open, Referred to Output				
Input Resistance (R_{IN})			50		kΩ
Input Bias Current (I_{BIAS})	V_S = 6V, Pins 2 and 3 Open		250		nA

Application Hints

GAIN CONTROL

To make the LM386 a more versatile amplifier, two pins (1 and 8) are provided for gain control. With pins 1 and 8 open the 1.35 kΩ resistor sets the gain at 20 (26 dB). If a capacitor is put from pin 1 to 8, bypassing the 1.35 kΩ resistor, the gain will go up to 200 (46 dB). If a resistor is placed in series with the capacitor, the gain can be set to any value from 20 to 200. Gain control can also be done by capacitively coupling a resistor (or FET) from pin 1 to ground.

Additional external components can be placed in parallel with the internal feedback resistors to tailor the gain and frequency response for individual applications. For example, we can compensate poor speaker bass response by frequency shaping the feedback path. This is done with a series RC from pin 1 to 5 (paralleling the internal 15 kΩ resistor). For 6 dB effective bass boost: R ≃ 15 kΩ, the lowest value for good stable operation is R = 10 kΩ if pin 8 is open. If pins 1 and 8 are bypassed then R as low as 2 kΩ can be used. This restriction is because the amplifier is only compensated for closed-loop gains greater than 9.

INPUT BIASING

The schematic shows that both inputs are biased to ground with a 50 kΩ resistor. The base current of the input transistors is about 250 nA, so the inputs are at about 12.5 mV when left open. If the dc source resistance driving the LM386 is higher than 250 kΩ it will contribute very little additional offset (about 2.5 mV at the input, 50 mV at the output). If the dc source resistance is less than 10 kΩ, then shorting the unused input to ground will keep the offset low (about 2.5 mV at the input, 50 mV at the output). For dc source resistances between these values we can eliminate excess offset by putting a resistor from the unused input to ground, equal in value to the dc source resistance. Of course all offset problems are eliminated if the input is capacitively coupled.

When using the LM386 with higher gains (bypassing the 1.35 kΩ resistor between pins 1 and 8) it is necessary to bypass the unused input, preventing degradation of gain and possible instabilities. This is done with a 0.1 µF capacitor or a short to ground depending on the dc source resistance on the driven input.

Appendix C

Arduino system code

```
// Serial.println(sptr);
#define BUFSIZE 0x400
#define BUFMASK 0x3FF
volatile int samples[16] [BUFSIZE] ;
volatile int code[16]  = { 15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0 } ; // change manuplation
//double vReal1[BUFSIZE] ;
volatile int sptr = 0 ,bufn = 0 , obufn = 2 ,  obufn2 = 1;
volatile int isr_count = 0 ;   // this was for debugging
unsigned long t1 , t2 , t3 ,  t4 , t5 , t6 ;
int fs = 2625  , id_buf_full=0 , id_buf_full2 = 0 , ii = 0;      // sampling freq  = 16000Hz
int k = 0   ; // code index
double   xn ,xn_1 , xn_2,xn_3 , xn_4, xn_5, xn_6 , xn_7, xn_8 ;
double   x2n ,x2n_1 , x2n_2 ,x2n_3 , x2n_4, x2n_5 ,x2n_6 , x2n_7 ,x2n_8;
double   outputn ;
double   output2n ;
int   x= 1  , amp = 4095 ;
double   vsine[160], pi = 3.141592654 ;
int val,val2 ;
void setup()
{
  Serial.begin (9600) ;
  adc_setup () ;
  pmc_enable_periph_clk (TC_INTERFACE_ID + 0*3+0) ;
  TcChannel * t = &(TC0->TC_CHANNEL)[0] ;
  t->TC_CCR = TC_CCR_CLKDIS ;
  t->TC_IDR = 0xFFFFFFFF ;
  t->TC_SR ;
  t->TC_CMR = TC_CMR_TCCLKS_TIMER_CLOCK1 |
            TC_CMR_WAVE |
            TC_CMR_WAVSEL_UP_RC |
            TC_CMR_EEVT_XC0 |
            TC_CMR_ACPA_CLEAR | TC_CMR_ACPC_CLEAR |
            TC_CMR_BCPB_CLEAR | TC_CMR_BCPC_CLEAR ;
  t->TC_RC =  fs ;
  t->TC_RA =  fs/2 ;
  t->TC_CMR = (t->TC_CMR & 0xFFF0FFFF) | TC_CMR_ACPA_CLEAR | TC_CMR_ACPC_SET ;  // set clear
and set from RA and RC compares
  t->TC_CCR = TC_CCR_CLKEN | TC_CCR_SWTRG ;
  setup_pio_TIOA0 () ;
  dac_setup () ;
 for ( int i = 0 ; i< 160 ; i++){
    vsine[i] =  sin(0.4375 * pi*i) ;
 }
}
```

```
void setup_pio_TIOA0 ()
{
  PIOB->PIO_PDR = PIO_PB25B_TIOA0 ;
  PIOB->PIO_IDR = PIO_PB25B_TIOA0 ;
  PIOB->PIO_ABSR |= PIO_PB25B_TIOA0 ;
}
void dac_setup ()
{
  pmc_enable_periph_clk (DACC_INTERFACE_ID) ;
  DACC->DACC_CR = DACC_CR_SWRST ;
  DACC->DACC_MR =
    DACC_MR_TRGEN_EN | DACC_MR_TRGSEL (1) |
    (1 << DACC_MR_USER_SEL_Pos) |
    DACC_MR_REFRESH (0x0F) |
    (24 << DACC_MR_STARTUP_Pos) ;
  DACC->DACC_IDR = 0xFFFFFFFF ;
  DACC->DACC_CHER = 0x2 ;
}
void dac_write (int Dinput)
{
  DACC->DACC_CDR = Dinput & 0xFFF ;
}
void adc_setup ()
{
  NVIC_EnableIRQ (ADC_IRQn) ;
  ADC->ADC_IDR = 0xFFFFFFFF ;
  ADC->ADC_IER = 0x80 ;
  ADC->ADC_CHDR = 0xFFFF ;
  ADC->ADC_CHER = 0x80 ;
  ADC->ADC_CGR = 0x15555555 ;
  ADC->ADC_COR = 0x00000000 ;
  ADC->ADC_MR = (ADC->ADC_MR & 0xFFFFFFF0) | (1 << 1) | ADC_MR_TRGEN ;  // 1 = trig source TIO
from TC0
}
// Circular buffer, power of two.
#ifdef __cplusplus
extern "C"
{
#endif

void ADC_Handler (void)
{
  if (ADC->ADC_ISR & ADC_ISR_EOC7)
  {
    ii = ii + 1 ;
    if (ii == 160){ ii = 0 ;}
    t3 = t1 ;
    t1 = micros() ;
    t4 = t1- t3 ;
    id_buf_full2 = 1 ;
    val = *(ADC->ADC_CDR+7)  & 0xFFF;
    val2=val;
    xn_8 = xn_7 ;
    xn_7 = xn_6 ;
    xn_6 = xn_5 ;
    xn_5 = xn_4 ;
    xn_4 = xn_3 ;
    xn_3 = xn_2 ;
    xn_2 = xn_1 ;
    xn_1 = xn ;
    xn = (double)(( val ) * vsine[ii])   ;
    outputn =    lowpass ( xn ,xn_1 ,xn_2 , xn_3 , xn_4 , xn_5,xn_6 , xn_7 , xn_8  ) ;
    x2n_8 = x2n_7 ;
    x2n_7 = x2n_6 ;
    x2n_6 = x2n_5 ;
    x2n_5 = x2n_4 ;
    x2n_4 = x2n_3 ;
    x2n_3 = x2n_2 ;
    x2n_2 = x2n_1 ;
    x2n_1 = x2n ;
    x2n = (outputn * vsine[ii]) ;
   output2n =   lowpass ( x2n ,x2n_1 ,x2n_2 , x2n_3 ,x2n_4 , x2n_5 ,x2n_6 ,x2n_7 , x2n_8  ) ;
    t2 = micros() ;
    samples[bufn] [sptr] = val ;
    sptr = (sptr+1)  ;
    if(sptr==1024 ){
              sptr = 0 ;
              bufn = bufn + 1 ;
              k = k +1 ;
                if ( k == 16 ){
                   k = 0 ;
                }
              obufn = code[k] ;
                  if(bufn == 16){
                      bufn =0 ;
                  }
              }
    if(sptr == 0 ){
      id_buf_full = 1 ;
      if ( obufn2 == 16 ){
         obufn2 = 0 ;
      }
    }
  }
  isr_count ++ ;
}
#ifdef __cplusplus
}
```

```
#endif
//   filter
double  lowpass( double ixn ,double ixn_1 , double ixn_2 ,double ixn_3 ,double ixn_4 ,double ixn_5
,double ixn_6 , double ixn_7 ,double ixn_8  )
{
  //t1 = micros() ;
  // Fs = 8000 Hz , Fpass = 3000 Hz , Fstop = 3500 Hz , Apass = 1 dB , Astop = 60 dB
  double yn ;
  yn =  -0.06943 * ( ixn + ixn_7 ) +0.0235 * (ixn_1 + ixn_6 ) + 0.2 *( ixn_2 + ixn_5)  + 0.3449 *( ixn_3 +
ixn_4) ;
  //t2 = micros() ;1
  return yn ;
}
void    loop()
 {
  if(id_buf_full == 1 ){
    id_buf_full = 0 ;
  }
  if(id_buf_full2 == 1 ){
    id_buf_full2 = 0 ;
}
```

Appendix D

Matlab .m code

```
clear all;
b = [ -0.078394 -0.041101 0.116124 0.306933 0.392876 0.306933
0.116124 -0.041101 -0.078394 ];
a = 1;

[h,w]=freqz(b,a,10000);
%samples = [ 1,15*16000 ] ;  % read 15 sec
[x2,fs]=audioread('handel2.wav');

y = filter(b,a,x2);
L=length(y) ;
fs
for i=1: length(y)
    x(i)=sin(2*pi*3500*i/fs);
    y2(i)=y(i)*x(i);
end

%s = spectrogram(y);
%s = spectrogram(y);
%spectrogram(y,'yaxis')
figure;
subplot(2,3,1);
Y_fft = fft(y);
%y_fft_max = max(Y_fft);
P2_y = abs(Y_fft/L);
P1_y = P2_y(1:L/2+1);
P1_y(2:end-1) = 2*P1_y(2:end-1);
f = fs*(0:(L/2))/L;
plot(f,P1_y)
%plot(f,P2_y)
title('y1 original signal ')
xlabel('f (Hz)')
ylabel('|P(y)|')

%figure;
subplot(2,3,2);
Y2_fft = fft(y2);
%y_fft_max = max(Y_fft);
P2_y2 = abs(Y2_fft/L);
P1_y2 = P2_y2(1:L/2+1);
P1_y2(2:end-1) = 2*P1_y2(2:end-1);
f = fs*(0:(L/2))/L;
plot(f,P1_y2);

%plot(f,P2_y)
title('y2 = y1 * sin(3500) ')
xlabel('f (Hz)')
ylabel('|P(y2)|')

%figure;
subplot(2,3,3);
%b = [0.331434 1.225337 1.792292 1.225337 0.331434 ];
%a = [1 1.770923 1.737695 0.757021 0.238799 ];
```

```
y2lp = filter(b,a,y2);
y3lp = filter(b,a,y2lp);

Y3lp_fft = fft(y3lp);
%y_fft_max = max(Y_fft);
P2_y3lp = abs(Y3lp_fft/L);
P1_y3lp = P2_y3lp(1:L/2+1);
P1_y3lp(2:end-1) = 2*P1_y3lp(2:end-1);
f = fs*(0:(L/2))/L;
plot(f,P1_y3lp)
%plot(f,P2_y)
title('y3lpf= LPF(y2)-Scrambled ')
xlabel('f (Hz)')
ylabel('|P(y3lp)|')

for i=1: length(y3lp)
    x(i)=sin(2*pi*3500*i/fs);
    y4(i)=y3lp(i)*x(i);
end
%figure;

subplot(2,3,4);
Y4_fft = fft(y4);
%y_fft_max = max(Y_fft);
P2_y4 = abs(Y4_fft/L);
P1_y4 = P2_y4(1:L/2+1);
P1_y4(2:end-1) = 2*P1_y4(2:end-1);
f = fs*(0:(L/2))/L;
plot(f,P1_y4)
%plot(f,P2_y)
title('y4 = y3lp * sin(3500) ')
xlabel('f (Hz)')
ylabel('|P(y4)|')
%figure;
subplot(2,3,5);
y4lp = filter(b,a,y4);
y5lp = filter(b,a,y4lp);
Y5lp_fft = fft(y5lp);
%y_fft_max = max(Y_fft);
P2_y5lp = abs(Y5lp_fft/L);
P1_y5lp = P2_y5lp(1:L/2+1);
P1_y5lp(2:end-1) = 2*P1_y5lp(2:end-1);
f = fs*(0:(L/2))/L;
plot(f,P1_y5lp)
%plot(f,P2_y)
title('y5lp = LPF(y4)-Descrambled ')
xlabel('f (Hz)')
ylabel('|P(y5lp)|')
subplot(2,3,6);
plot(abs(h));
title('LPF frequency response ')
xlabel('f (Hz)')
ylabel('|H(z)|')
figure;
subplot(1,3,1);
specgram(y,512,fs);
title('orignal signal ')
subplot(1,3,2);
specgram(y3lp,512,fs);
title('scrambel  signal ')
```

```
subplot(1,3,3);
specgram(y5lp,512,fs);
title('descrambel  signal ')
%Yd_fft = P2_y - P2_y3lp ;
%plot(f,abs(P2_y - P2_y3lp));
%title('Yd_fft = abs(y_fft -y3lp_fft)')
%xlabel('f (Hz)')
%ylabel('|P(f)|')
```