SIMULTANEOUS VOICE AND IN-BAND DATA

A Thesis
Submitted to the College of Graduate Studies and Research
in Partial Fulfilment of the Requirements
for the Degree of Master of Science
in the Department of Electrical Engineering
University of Saskatchewan
Saskatoon

by

Dean Richard Zurburg

638? OCT 28 '99 [illegible]

August 1999

PERMISSION TO USE

ABSTRACT

This thesis presents a new technique for simultaneously transmitting in-band data and analog voice on a standard telephone voice-grade line. A portion of the speech spectrum is replaced with a medium speed digital data signal so that both speech and data signals are carried on a telephone channel to a remote receiver. In this work, a range of frequency components is removed from the mid-band region of the human voice spectrum. The modem then transmits data in this mid-band range. Processing at the receiving end separates the voice and data signals and demodulates the data for use by a computer.

The thesis describes a prototype system and presents experimental data gathered from several tests. The thesis also discusses advantages and disadvantages of the "data in voice" modulation technique and proposes new services which could not be supported with current voiceband modems.

ACKNOWLEDGMENTS

The author thanks Professor David Dodds and Professor Kunio Takaya for their valuable guidance throughout the course of this research. The author also thanks the staff of TRLabs for their encouragement, support and guidance. Telecommunications Research Laboratories (TRLabs) is gratefully acknowledged for providing financial support for this research. Special thanks is extended to the author's family and friends for their encouragement and support.

Table of Contents

List of Tables

List of Figures

ABBREVIATIONS

ADC	analog to digital converter
BER	bit error ratio
b/s	bits per second
DAC	digital to analog converter
dB	decibel
dBm	decibels relative to 1 milliwatt
dc	direct current
DSP	digital signal processor
GMSK	Gaussian minimum shift keying
ISI	Intersymbol interference
kb/s	kilobits per second
kHz	kilohertz
LPF	lowpass filter
mA	milliamps
MHz	megahertz
ns	nanoseconds
PC	personal computer
RAM	random access memory
SNR	signal to noise ratio

1. Introduction

1.1 Motivation for Research

With the ever increasing demand for communication services to the home and office, the telecommunication companies have been striving to offer new and useful services to the public.

Telecommunication companies have a large investment with twisted pair copper wires connecting each telephone subscriber to the switched public telephone network. With millions of kilometers of copper wiring in the ground throughout North America, telecommunication companies have a vested interest in trying to recover as much of that investment as possible before the deployment of fiber to the customer location. This return on investment can be increased by offering the public more services. These services may include data services like Internet access, or telephone functionality services like last call return and call forward. With a small charge for each service (\$10), the overall return per line is improved.

If it is possible to offer services that use the same pair of wires currently installed in the subscriber premises, then it avoids the costly installation of extra pairs of wires. This reduces the amount of time necessary to get the new service operating and helps the service become profitable quickly. Some services that currently require a second telephone line include ISDN (requires two pairs for operation), fax lines and data lines.

It is also possible that some of the new services would be from one subscriber premise to another. If it is also possible to use in-band data to carry data from one customer to another (end to end) without any modification of the telephone network, then external services would keep implementation costs down by not requiring a telephone network upgrade.

1.2 Historical Background

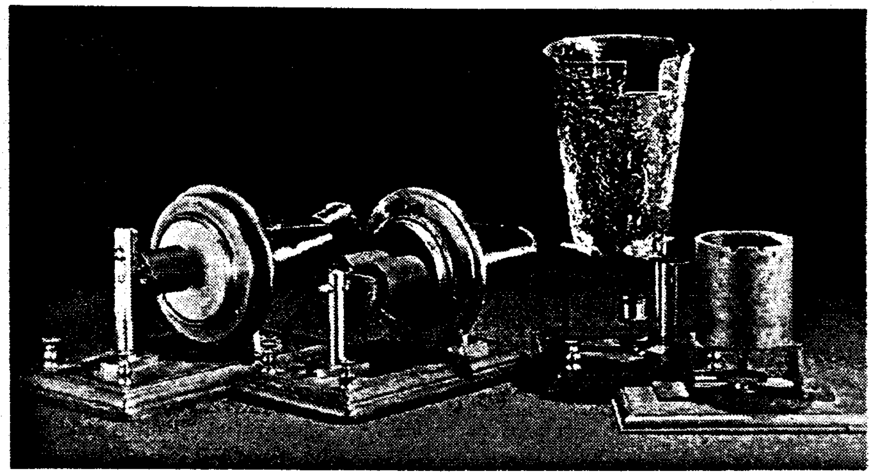


Figure 1.1 Bell's Early Equipment Revolution

On June 2, 1875, while tuning a resonant telegraph at a lab facility in New York, Alexander Graham Bell and Thomas Watson stumbled across one of the greatest inventions of the 20th century. Through their good fortune, they had discovered one of the key pieces necessary for the invention of the telephone. Their persistence and hard work over the next few years helped launch the world into the telecommunications era. Some of Bell's early equipment used to develop the telephone can be seen in Figure 1.1. While much of the technology to implement the telephone has changed, the basic concept of the telephone has remained constant for the last 100 years.

1.3 Data on the Phone Network

The introduction of the home personal computer in the late 70's brought a new challenge to the telephone system. No longer would the phone system carry only voice, but now, computer data would also be transported on the telephone network. With the use of a modulator-demodulator (modem), computers are able to send and receive digital data over the phone network.

From humble beginnings at 300 bits per second (b/s), two computers can now communicate over the public switched telephone network at speeds up to 33.6 kb/s.[1]

Over the years, a large number of computers have been interconnected through the use of modems and other similar devices. They have formed a network of computers that has become known as the Internet. With a modem connection, a user may connect to anywhere in the world through the use of the Internet. Home shopping, information services, and on-line chat are only a few services available through the data connection. The telephone companies not only provide a transmission medium, but also now offer information services of their own.

1.4 Low Speed Data on the Subscriber Loop

One of the more popular services that telephone companies now offer is calling number delivery (CND) services. Calling number delivery has proven to be a valuable option for people around the world. The service delivers the number and sometimes name of the calling individual to a display panel on the telephone. The digital information is delivered between the first and second ring of the telephone using a simple FSK modulation technique at 1200 b/s. A simplified diagram of the data sequence can be seen in Figure 1.2.

To deliver more services to the customer, it is desirable to combine the calling number delivery service with call waiting. Call waiting is a service that notifies you of a second caller. It interrupts the current phone conversation and delivers a short beep to notify the user of a second incoming call. By combining these services in a new fashion using in-band data, the name

[1]USRobotics, along with several other companies, has introduced 56kb/s "X2" technology. This allows the modem user to obtain download speeds up to 56kb/s while maintaining a 33.6kb/s upload speed.

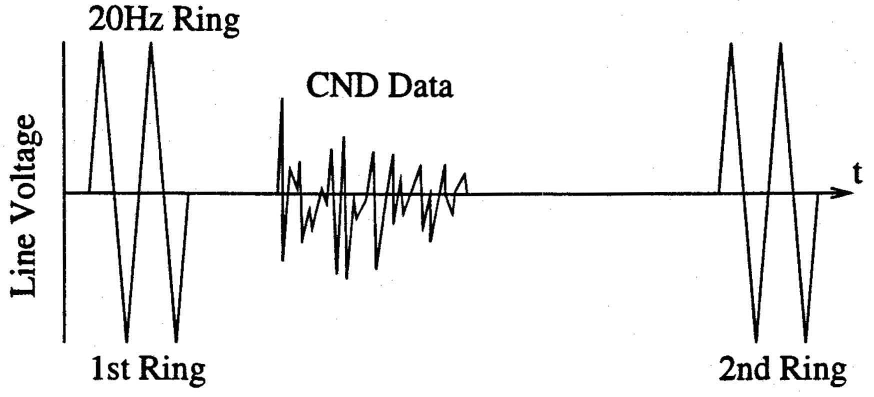


Figure 1.2 The Delivery of CND Information

and number of the second caller could be presented to the user without interruption of the original call.

Telephone Companies are also beginning to offer an on-line information service known as ADSI (Analog Display Services Interface). ADSI provides short information bursts to the telephone display device while the telephone is in use. The data is used to update a text display screen on the ADSI telephone. The text is used to provide interactive menus and choices to the user. Currently, the text service ADSI interrupts the voice channel to transmit the text data to the display screen. During this period, no voice transmission may take place. As this interruption may last for a couple of seconds, this service would be unattractive to individuals who need an uninterrupted voice path, such as in a conference call environment. The use of in-band data could provide a means to deliver the ADSI data to the telephone set without interrupting the voice conversation.

1.5 In-band Data Applications

High Speed modems (28.8 or 33.6 kb/s) which can carry both data and digitized voice are available on the market today. These modems are commonly referred to as DVM or Data and Voice modems. In a typical unit, voice is compressed to 9600 b/s using a voice compression algorithm such as

Code Excited Linear Predictive Coding (CELP) [1], and is combined with a digital data stream of 19.2 kb/s to form a combined transmission rate of 28.8 kb/s. With these modems, you can transfer digital data back and forth while maintaining a continuous voice conversation.

A problem experienced with this system is the inability to maintain a voice conversation with data transmission in a conference call environment. The DVMs are meant for point-to-point operation. This excludes simultaneous voice and data service among 3 or more sites. If the two services could exist together simultaneously in a 3-way (or N-way) conference situation, new services would be available to the telephone companies.

The use of interactive conferencing has begun to grow with the recent introduction of many computer based conferencing packages. While holding a conversation, both parties are presented with graphics and documents sent by the other. With in-band data transmission capability, the graphics can be sent without interruption between computers on the same phone line as the analog voice conversation is occurring.

1.6 Thesis Objectives

This thesis aims to demonstrate the ability to send digital data over the telephone line in a portion of the frequency band normally occupied by voice without causing perceptually noticeable effects in voice communication.

The thesis will demonstrate that the human ear does not need the entire voice frequency range for proper perception and understanding of the conversation. By choosing the data bandwidth, a trade off between the voice quality and the data rate will be reached. It will be shown that with limited voice degradation, a low to medium data rate can be obtained.

1.7 Thesis Organization

This thesis is organized into 6 chapters.

Chapter 1 provides background and motivation for the thesis. The objectives of this research are also stated.

Chapter 2 introduces the subscriber access loop, transmission bandwidth and quantizing noise.

Chapter 3 describes the techniques which were considered for implementation of the system using digital processing. Computer simulation was used as a tool for evaluation. It discusses the theory of modulation and demodulation of Gaussian Minimum Shift Keying (GMSK). Considerations necessary for implementation on digital signal processing (dsp) hardware from Atlanta Signal Processors Inc. are also presented. Chapter 3 also presents another digital signal processing algorithm known as Multirate digital signal processing. It summarizes the improvements which can be made to the original demodulation algorithm. These improvements allow for a better demodulation process which ideally should allow for a better bit error rate.

Chapter 4 details the system implementation using the combination of a personal computer and a digital signal processing board. Protocol handing and handshaking methods are also described.

Chapter 5 presents the results from testing on the system. Bit error rate tests in the presence of Gaussian noise in the transmission channel gives an indication of the performance of the overall system. Addition of Gaussian noise into the handset's microphone also demonstrates the systems tolerance to large noise additions in the "sub-band" voice channel.

Chapter 6 summarizes the overall system performance and the results. Future work which can improve various aspects of the system is also discussed.

2. Voice and Data Transmission in the Telecommunication Network

Today's telephone networks consist of several parts including the local loop, the voice switch, and the long distance network. A conceptual diagram of the telephone network is seen in Figure 2.1. The telephone network has traditionally been based on the voice transmission and thus the local loop is limited to a bandwidth between 300 Hz and 3400 Hz. The local loops are typically connected by copper wire to the switch. However, cellular telephones have replaced this wire path with a wireless transmission system. The voice switch provides the means by which the voice calls are connected between local loops. The long distance networks provide the final piece by interconnecting the world's switches together allowing most local loops to connect to any other local loop in the world.

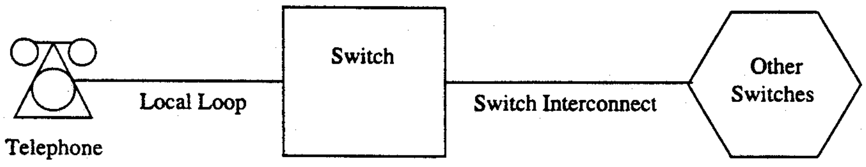


Figure 2.1 Conceptual Diagram of Telephone Network

2.1 The Local Loop

The description of the telephone system begins with the local subscriber loop. This basic loop consists of the telephone set (phone) located on the premises of the individual, the copper cable running from the phone to the central office, and the circuitry located inside the central office [3]. A conceptual diagram of this circuit can be seen in Figure 2.2.

When the receiver on the telephone is picked-up, current flows in the wires

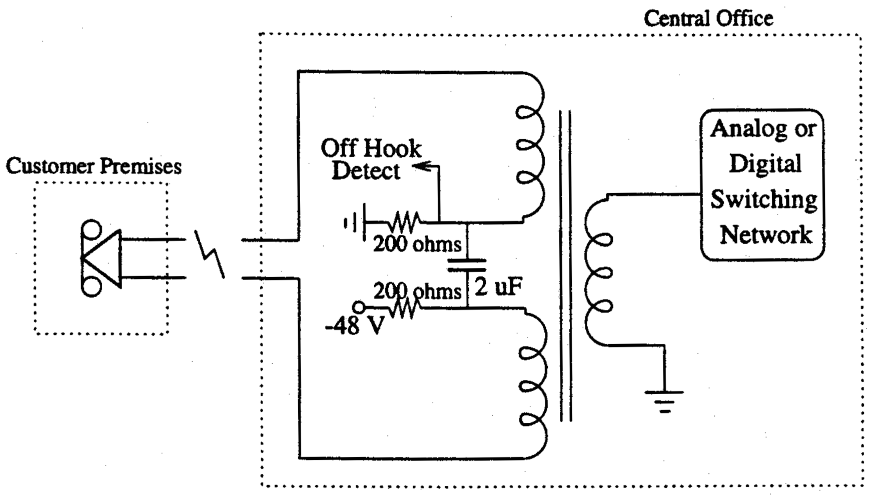


Figure 2.2 Conceptual Diagram of Local Public Subscriber Loop

between the telephone and the central office. A 48 V battery at the central office produces a current between 23 and 100 mA. The current is detected at the central office, where a dial tone signal is connected. This indicates that the switch is ready to receive address signalling. The telephone generates either a pulse or dual tone signal to represent each address digit. The switch located at the central office decodes the series of tones or pulses and routes the call through to its destination. If the destination is not on the same switch as the caller, the switch will use its long distance trunks to route the call towards the next voice switch nearer to the destination. A ringing or busy tone is generated to inform the caller as to the status of the call. If the called party answers the telephone, the switch (or switches) completes the connection and the voice conversation may begin.

To save on wiring installation cost, most telephones are wired to the central office using only a single pair of wires. Both directions of signals travel on the single pair of wires, and a hybrid coupler is used to separate the signals into a four-wire system (two for the ear piece, two for the microphone). The hybrid coupler has been traditionally implemented using transformers. By using the transformers with balanced impedances, incoming signals are directed to the receiver section while outgoing signals are directed to the transmission line. The hybrid coupler device relies on impedance matching with the transmission line to obtain good quality isolation in the two separate directions. However, as it is difficult to accurately match the impedance of the transmission line, echoes are often a problem. Transmission line theory states that a discontinuity in the transmission line (i.e., imperfect impedance matching) will result in signal reflection (i.e., echoes)[3]. While echoes on short transmission loops return too quickly to be noticed, echoes on long distance connections can cause the speaker to become confused. Echoes returning after about 30 ms will make the connection undesirable. On very long distance telephone calls, such as transatlantic calls, the telephone company will put echo cancellation devices on both ends of the transmission line to help reduce echoes on the voice call.

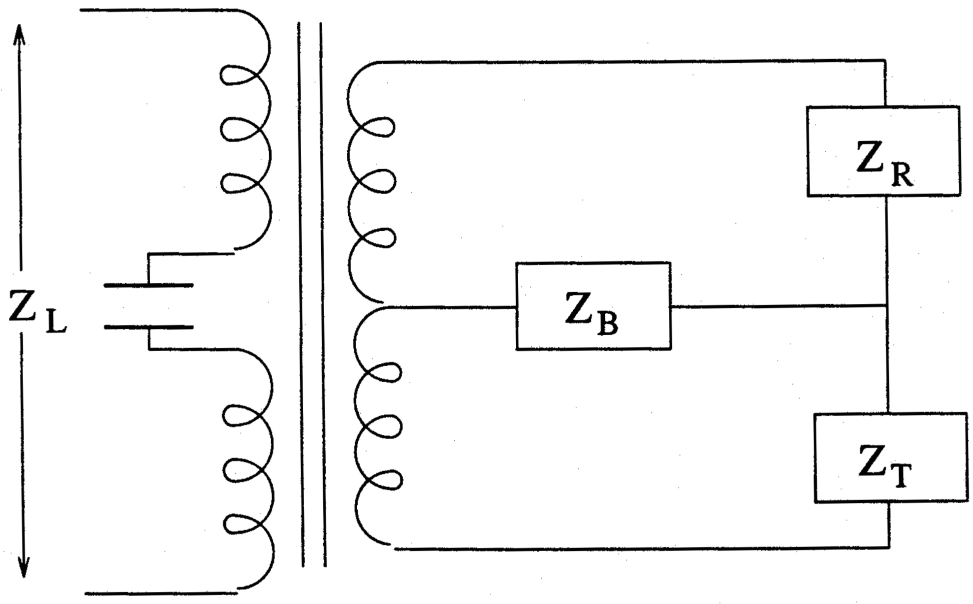


Figure 2.3 Single Transformer Hybrid Coupler

Figure 2.3 illustrates a simple single transformer implementation of a hybrid coupler. The two wires of the transmission line are connected across Z_L. The impedance Z_R represents the impedance of the receiver (earphone) section while the impedance Z_T represents the impedance of the transmitter (microphone) section. The impedance Z_B is an impedance which is adjusted to "balance" the hybrid coupler. In order to minimize echoes on the transmission line, the hybrid coupler must be made to closely match the impedance of the transmission line in the frequencies of interest (300 Hz - 4 kHz). Assuming an ideal transformer, the trans-hybrid received signal can be calculated from the following equation [3] :

$$\frac{V_R}{V_T} = \frac{Z_R(Z_B - Z_L)}{4Z_BZ_L + Z_RZ_B + Z_RZ_L} \tag{2.1}$$

where V_R is the voltage drop across the receiver section and V_T is the voltage drop across the transmitter section. The trans-hybrid received signal reduces to zero if the line impedance Z_L is equal to Z_B in magnitude and phase. Due to the variation of the characteristic impedance of the transmission line, it is difficult to match to the transmission line over a broad range of frequencies. Thus some reflections and echos will be generated on the telelphone line.

Digital transmission using modems (modulator-demodulator) is susceptible to echoes in the telephone network. Even small echoes can cause demodulator lock-up problems and erroneous data if not corrected. Modern modems use echo cancellation to reduce echoes in the return path and help improve the performance.

2.2 Digital Transmission of Analog Voiceband Signal

The telephone transmits an analog signal onto the transmission (or telephone) line and this signal is converted into a digital waveform at the central office before continuing in the telephone network. The encoder-decoder (codec), illustrated in Figure 2.4, converts the analog voice or data signal to companded pulse code modulation (CPCM).

To introduce digital transmission of voice signals, linear PCM is first considered. Linear PCM is a technique where the analog signal is compared against quantized levels to determine a level that most closely matches the

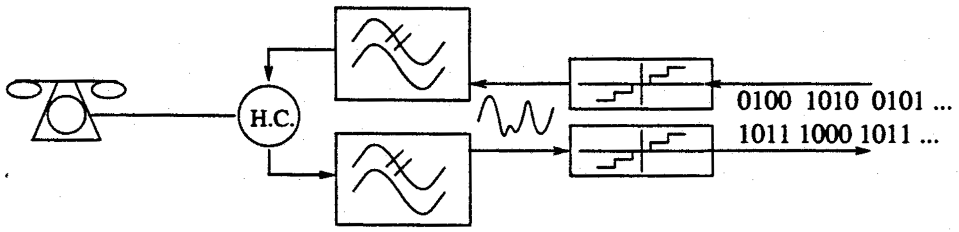


Figure 2.4 Hybrid Coupler, Filters and Voice Codec

input signal. The input signal is compared at regular intervals, resulting in a series of quantized samples indicating the approximate input signal level at the given sampling times. The number of different quantizing levels is set by selecting the number of bits used to represent the quantized sample. For example, if four bits are used to represent the quantized sample, then there will be 16 different quantizing levels to compare against the input signal level. In linear PCM, these levels are equally spaced. These levels are also dependent on a constant multiplier that is used to set the maximum absolute value the quantizer will represent. If a four bit quantizer needs to swing between $\pm$ 2 volts, the multiplier would be $\frac{1}{4}$ Volt as $16 * \frac{1}{4}$ results in the 4 Volt swing. There are 16 levels in

The quantization process introduces a certain amount of noise and distortion into the output signal. The difference between the actual voltage level and the quantized level is known as "quantization noise".

Figure 2.5 illustrates the resulting quantizing error $e(t)$ that is produced in the quantizing process. The error is the difference between the input signal $x(t)$ and the output signal $\tilde{x}(t)$. The quantization noise has a white spectral density. The worst case error for quantizing will be one half of a quantizing interval. The quantizer always gives a result that is in the middle of the quantizing interval. For example, if the quantizer has its intervals set to integer numbers (1 to 2, 2 to 3, 3 to 4 etc.), then the quantizer will give results that are in the middle (1.5, 2.5, 3.5, etc.). Thus, when an analog signal is near the edge of the quantizer interval (i.e. 1.99), an error of one

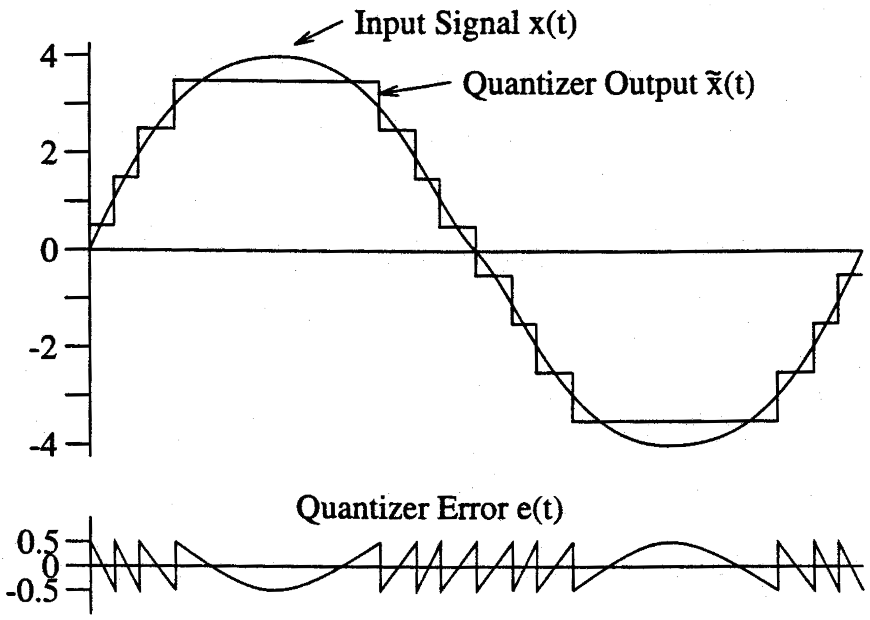


Figure 2.5 Output Signal and Quantizing Error for PCM

half of the interval will exist for that sample (0.49 in this case). This error on each quantizer output will be statistically random in nature and will give the white spectral density mentioned previously. Establishing a large number of small quantization intervals can minimize this noise. However, as the number of levels increases, the number of bits needed to represent the different levels must also increase.

As the quantizing error is related to the quantizer bin sizes, it will not decrease as the level of the input signal decreases. Consequently, the signal-to-noise ratio will decrease as the level of the input signal decreases.

Figure 2.6 illustrates that as the signal input level of the PCM decreases, the SNR of the system will also decrease. The figure uses the signal amplitude (A) divided by the quantizer range (A_{max}) as a reference. A reading of $0dB$ would represent a signal that occupies the entire range of the quantizer.

To reduce the quantizing error for small signals, a small quantizing interval is used. If the quantizing interval is not linear, but allowed to increase in proportion to the sample value, a more constant signal to noise ratio (SNR)

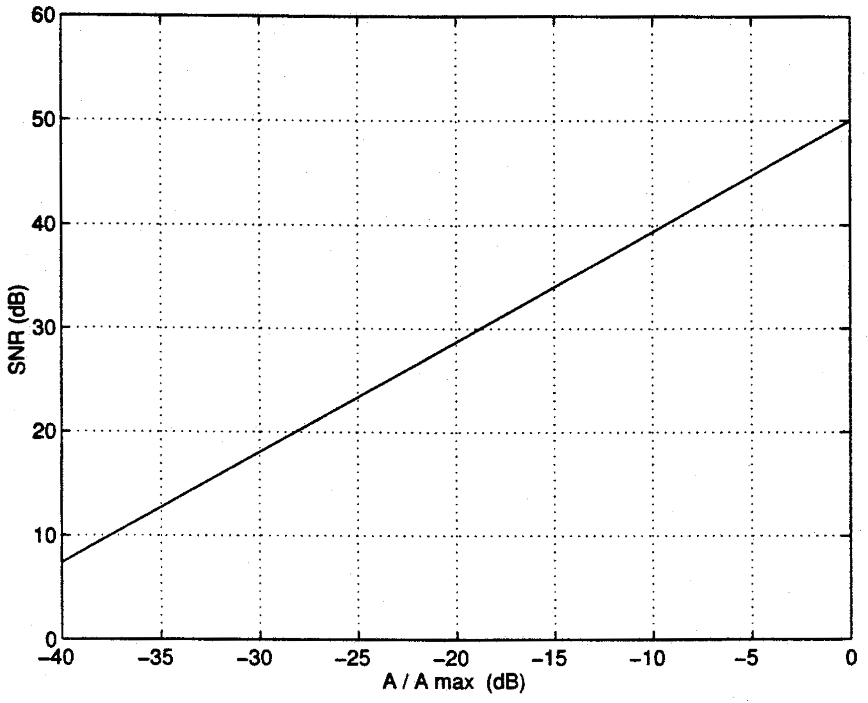


Figure 2.6 Signal to quantizing noise ratio of 8 bit uniform PCM coding

is generated for the entire effective range of the encoder [1]. With this technique, a fixed number of bits per sample provides a specified SNR for small signals, and an adequate dynamic range for large signals.

North American telephone systems use a technique known as μ-law (mulaw) companding. This technique is used to generate the compressed PCM. Figure 2.7 is an example of a four bit (sign + three bits) μ-law PCM encoder. Only the positive half of the companding range is shown for clarity. The negative half of the companding range is a mirror image of the positive half. Notice that the quantizer levels become larger as the signal amplitude increases. Eight bit, $\mu = 255$ PCM is the standard companding formula used today in North America. The eight bits are broken into a sign bit, three exponent bits, and four mantissa bits. Sixteen segments (eight positive segments and eight negative segments), each with 16 steps, are generated. Beginning from the middle two segments, each progressive segment contains steps that are twice the size of the previous segment. The decoded value for μ-law is given by:

$$2^E(M + 16.5) - 16.5 \tag{2.2}$$

where E represents the exponent and M represents the mantissa. The resulting number is multiplied by the sign bit and by the base voltage. For μ-law, this base voltage is $386\mu V$[3].

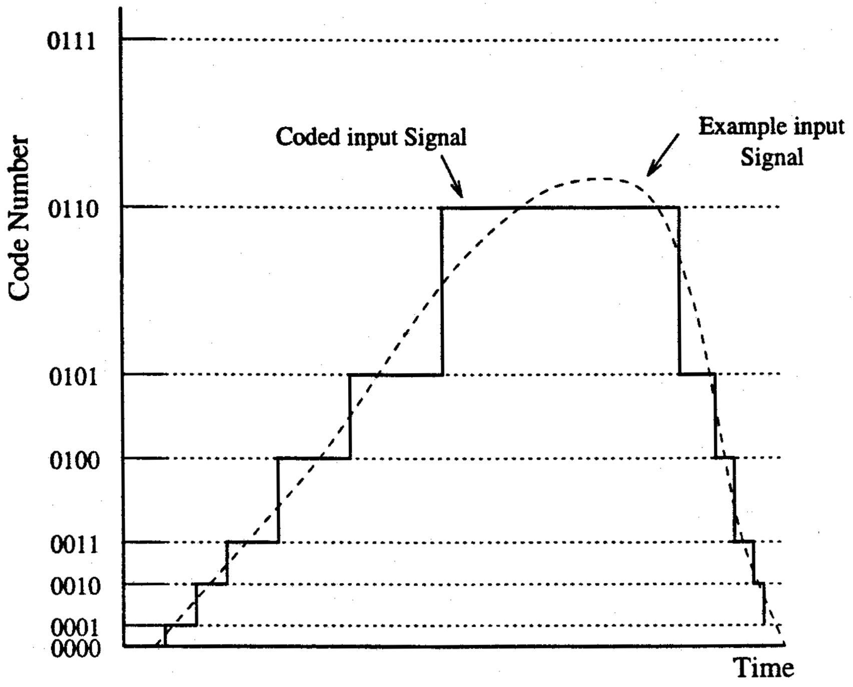


Figure 2.7 Companded Coding using 4 bit μ-law PCM applied to a time signal

With the eight bit code word, an effective dynamic range of 48.4 dB [1] is achieved. This dynamic range extends from a signal occupying the entire range of the first quantizing level up to the limits of the code word. In an eight bit μ-law system, the maximum input signal amplitude would be ± 1.55 V. Figure 2.8 compares the signal-to-quantizing noise of μ-law coding with the sine wave signal amplitude. The SNR does not begin to deteriorate until the signal level decreases below -50 dBm0.[1]

[1]The unit dBm0 is an adjusted measure of signal strength. The abbreviation dBm0 is commonly used to indicate what the signal magnitude would be (in dBm) at the central

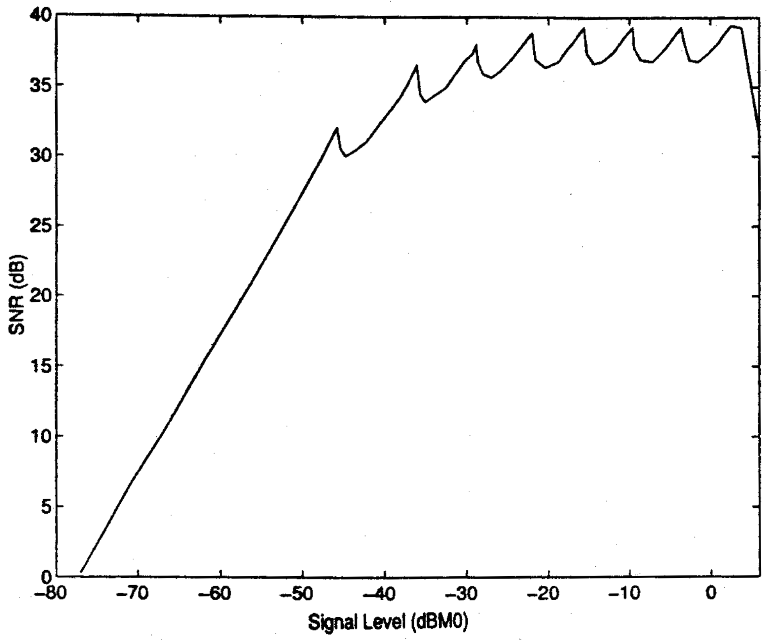


Figure 2.8 Signal-to-quantizing Noise of μ-law Coding with Sine Wave Inputs

Quantizer overload will also cause the SNR to decrease. Signal levels above three dBm0 show a significant reduction in SNR as clipping occurs to the signal.

The work preformed in this thesis occured using the North American telephone system standard coding. The standard telephone system uses a sampling rate of 8 kHz, an 8-bit code word, and μ-law companding coding. The total data throughput of the system is a continuous 64 kb/s.

Because sampling occurs at 8 kHz, precautions must be taken to avoid aliasing. The Nyquist Sampling Theorem [14] states that frequencies above half the sampling rate will appear as an alias frequency in the lower baseband frequency range. To avoid this problem, an anti-aliasing filter is applied to the analog signal prior to sampling. This filter limits the upper frequency band of the telephone to approximately 3500 Hz. The frequency range of a

office. The measurement combines absolute level with the gain or loss between the central office and the measurement point[3].

typical telephone circuit illustrated in Figure 2.9 is also limited in the lower range to approximately 200 Hz. This lower limit is a result of many causes, including the transformer in the hybrid couplers, and the frequency response of the microphone in the handset. The microphone unit in the telephone has a lower frequency limit governed by the ability of the diaphragm to vibrate at low frequencies. A useful bandwidth of approximately 3300 Hz is left for audio processing.

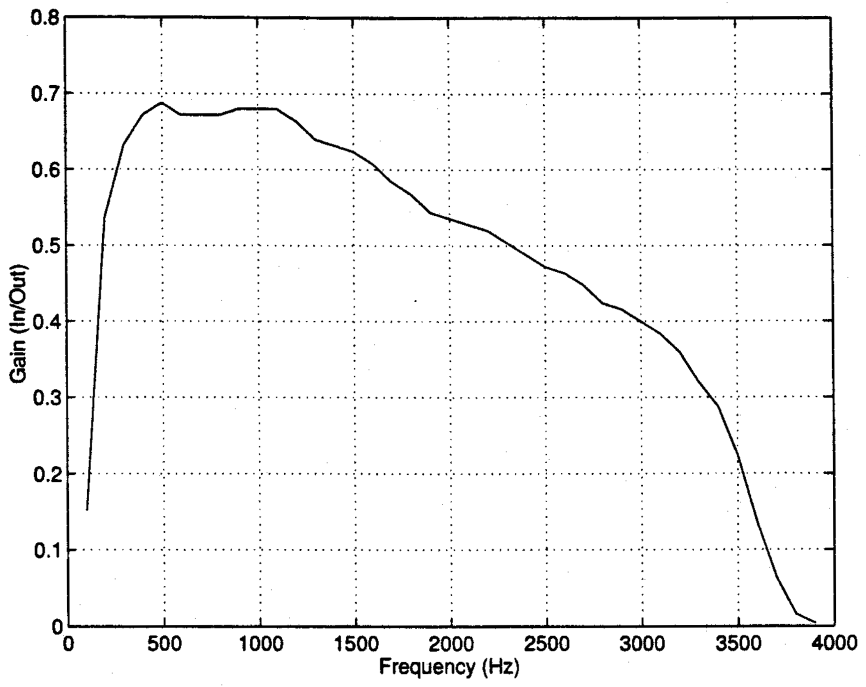


Figure 2.9 Magnitude Frequency Response of a typical Telephone Circuit

2.3 Digital Switching

The purpose of the digital switch is to route the digitized voice samples from one local loop port to another. Unlike its analog predecessors that used mechanical levers to connect correct local loops together (crossbar switch), the digital switch is based on computer memory. Each digitized sample is temporarily stored in a memory location until the computer transfers the sample to the appropriate digitizing local loop card. The digital switch synchronizes all the local loop cards (line cards) to provide and receive digitized samples at the same time and at the same rate. This synchronization allows for orderly flow of digitized samples in and out of the switch's memory.

2.4 Digital Transmission

When a telephone call must navigate over more then one digital switch, the digital transmission system is used. Digital switches are interconnected to others using high-speed digital data lines (or trunks). Typically, many calls will be leaving and entering a switch at any given instance. The digital samples are multiplexed together and transmitted on a high-speed digital data connection. Common channel signalling (CCS) is a protocol that is sent between switches to navigated each multiplexed digital signal to the correct destination port. As the multiplexed digital data enters the switch, one of two things will happen. It will be removed from the multiplexed data stream and sent to a local line card, or it will be multiplexed onto another digital trunk and sent towards another switch nearer the destination.

On telephone calls covering vast distances, the telephone company will typically use echo cancellers. Echoes are created when analog signals travelling on the telephone line meet the discontinuity of the telephone set. A portion of the original signal is reflected back towards the central office switching center. Normally, echoes that return quickly are not noticed by the listener. However, on the long distance telephone calls, echoes may return long after the person has said something. The echo canceller will subtract a portion of a delayed version of the signal to "cancel" the reflected echo. It is interesting to note that if an echo-cancelling device is placed in the local switch, it will only benefit the remote caller. The device would cancel echoes from the local caller so that the remote caller would not hear them. The local caller must rely on the remote Telephone Company having echo-cancelling devices in place.

2.5 Voice and Data Transmission Using Time Division Multiplexing (TDM)

Time division multiplexing is defined as the time interleaving (alternating) of samples from several sources so that the information from these sources can be transmitted serially over a single communication channel [2]. Some examples of TDM include Ethernet, AppleTalk, DS-3, SONET, and Synchronous TDM-T1. TDM typically uses multiplexing to provide several "virtual" data paths within the serial data stream. A simple visual example of multiplexing in TDM can be seen in Figure 2.10.

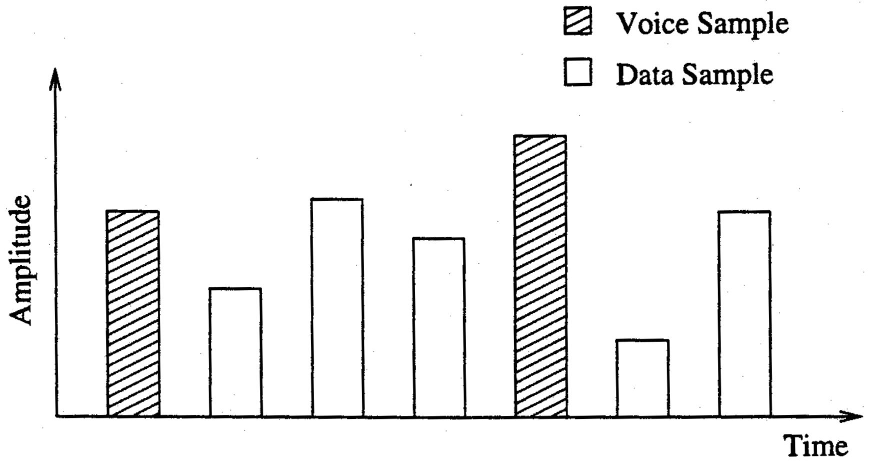


Figure 2.10 Example of Simple TDM Multiplexing

To use TDM to send simultaneous voice and data, the voice must first be converted to digital. Depending on available bandwidth, the voice may be compressed to conserve bandwidth. The use of high-speed digital modems allows the compressed voice to be multiplexed with the digital data and sent over the telephone line to another location. The bandwidth requirement of the compressed voice can change depending on whether the person is talking or silent. A smart implementation of the digital modems will increase data throughput in times when the voice is silent. Figure 2.11 illustrates a block diagram of this system.

This method has been commercially developed and proven effective on

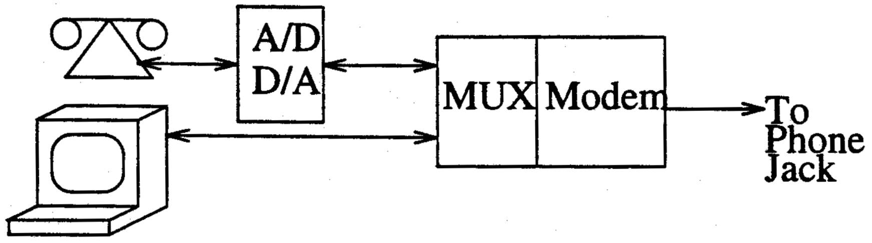


Figure 2.11 Block Diagram of TDM Point to Point Voice and Data Method

two-way calls with adequate line quality. Commercial units use a 33.6 kb/s modem to send the digital stream. The voice is compressed to a 7.2 kb/s digital stream leaving up to 26 kb/s available for the data stream and supporting handshaking. The handshaking allows the modem to continually adjust for voice bandwidth requirements. In times of silence, nearly the entire modem bandwidth is available for the data.

Current manufacturers of the voice and data modems include Zoom technology (Boston, MA) and Microcom (a subsidiary of Compaq Computers). Microcom manufactures a 33.6 kb/s Data/Fax modem called the OfficePorte. It supports the AudioSpan[2] technology. The AudioSpan or Analog Simultaneous Voice and Data (ASVD) is a protocol to support simultaneous voice and data across a standard analog switched network telephone connection. The protocol was developed to implement the ITT standard *V*.61 and the ITT draft proposal *V*.34*Q*. Modems implementing the AudioSpan protocol were first available on the market in July of 1996. Zoom Technology also makes a similar product called the ComStar XT SVD. It also uses the AudioSpan protocol to implement simultaneous voice and data.

AudioSpan equipped modems were not seen as a possible solution to the problem presented in this thesis. They do not allow a conference call environment to function without the use of a centralized dial-in system. The modems are unable to correctly operate in a star (bridge) connected conferencing environment as the modems become confused if more than one other

[2]AudioSpan is a trademark of Rockwell Semiconductoı Systems

modem is connected. It would be necessary to operate a separate dial-in conference facility (like that seen in Figure 2.12), where specialized conference equipment using computer processing would combine data from each of the modems and redistribute to each site. The cost involved in constructing and operating the "dial-in" facility makes the use of AudioSpan modems unattractive.

The modems could also be operated in a polling method. Polling is when each participant is repetitively queried to see if any data is available. While this would eliminate the need for the specialized centralized conference equipment, it would introduce the problem of not having simultaneous voice conversations. The polled method of modem operation would be unable to transmit more than one voice conversation in one direction at a time. This also makes the use of the AudioSpan modems unattractive.

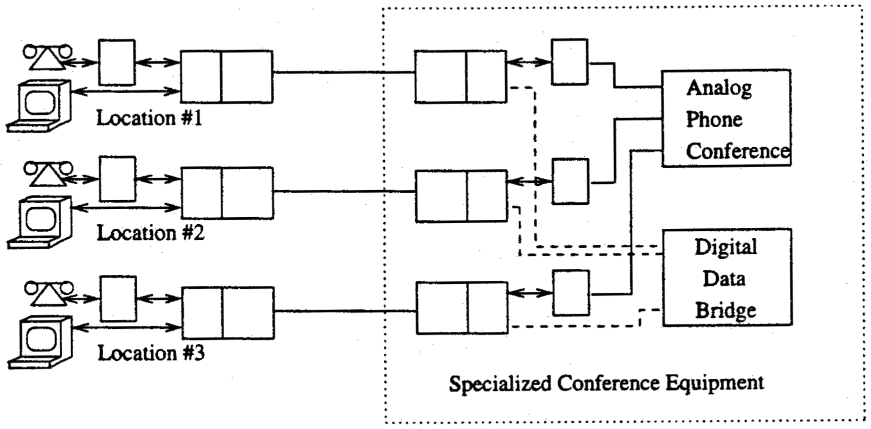


Figure 2.12 Block Diagram of TDM Conference Call Method for Voice and Data

2.6 Voice and Data Transmission using Frequency Division Multiplexing (FDM)

Frequency Division Multiplexing is defined as a technique for transmitting multiple messages simultaneously over a wideband channel by first modulating the message signals into several subcarriers and forming a composite signal that consists of the sum of these modulated subcarriers [2]. This technique is used in many systems including cellular telephones, AM and FM Radio, cable TV, and short wave radio.

To implement simultaneous voice and data using FDM, the 300 Hz - 3400 Hz frequency band is divided into sections. One portion of the frequency bands is used for data, while the remaining portion is used for voice. Using FDM, the voice can be left in its analog form allowing the use of simple bridged conference calls. With the use of multi-point (or polled) modems, each site can take turns sending digital information. This way, multiple sites may be included in the conferencing environment. No complicated processing or separate dial-in facilities are necessary to send and receive the voice channel as the voice remains in its analog form. All participants may talk simultaneously and be heard by every other conferencing participant. No time sharing of the voice channel is necessary.

A FDM system would be able to take advantage of the Telephone Company's existing analog conferencing equipment that is available in most areas of North America. For three-way conference calls, participants may use the three-way-calling feature available as part of the telephone customer services. For larger conference calls, most Telephone Companies offer multi-person, operator-controlled conference calls. The voice conferencing equipment is generally owned and operated by the Telephone Company and is located on the central office premises.

Figure 2.13 shows a simplified version of the frequency division multiplexing configuration for transferring data and voice between multiple sites. The analog voice occupies one subset of frequencies while the digital modem signal occupies another.

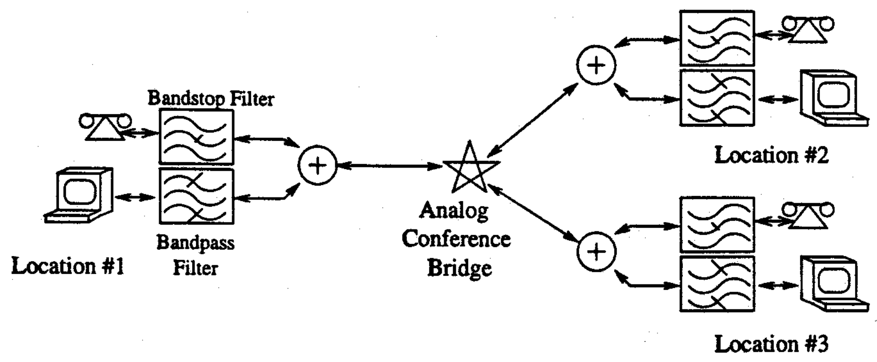


Figure 2.13 Block Diagram of FDM Conference Call Method for Voice and Data

2.7 Selecting a Method for In-band Data

As the previous sections introduced, there are two obvious methods for providing in-band data on the telephone transmission medium. The first is using Time Division Multiplexing (TDM) and the second is using Frequency Division Multiplexing (FDM). While both would provide a solution to the problem, FDM was selected as the method to use for continuing the research. The decision was made based on several factors including expected hardware complexity, expected software complexity, and experience. As a result of this decision, the remainder of this thesis will focus on in-band data using FDM.

3. Requirements for FDM Voice and Data Transmission

3.1 Required Frequencies for Voice Transmission

Initial testing focused on using some of the voice band for data transmission without significantly affecting the quality of the voice. This research was guided by the assumption that *the human voice is comprised of mostly odd harmonics* [1]. Accordingly, one would say that filtering out the frequency areas where the even harmonics dominate and odd harmonics are scarce will not significantly reduce the voice quality. Considering this assumption, testing began to determine approximate frequencies where the stopband filters should be placed. The bandwidth of the stopband filter and its effect on the perceived voice quality was also tested.

The first step was to digitize some sound and voice samples so that Matlab [2] could be used to run various simulations. A sampling rate of 8 kHz and a quantization word length of 8 bits were used to model the transmission environment of the telephone system. For simplicity, the experimental system used linear pulse code modulation (LPCM) rather than the companded (CPCM) coding used in commercial telephony. An analog anti-aliasing filter was used on the audio input to eliminate the effects of aliasing. Many "test sentences" were recorded to test the different filter configurations using a diverse set of audio samples. The test sentences included the author's voice

[1]In the course of past research on voice coding, Prof. Dodds noticed that the fundamental component of voice frequencies is on the order of 800 Hz and that frequencies in the band around 1600 Hz could be removed without a large perceived change in voice quality.

[2]Matlab is a trademark of Mathworks Inc.

while reading a passage from a text book, a male speaker from a French CBC radio talking about baseball scores, a female voice from English CBC radio talking about the weather forecast, and various audio samples collected from the Internet.

Matlab was used to generate filters with varying widths and center frequencies. Each test sentence was processed using the different filters before being recorded onto audio cassette tape.

To generate the various filters in Matlab, a built-in filter generation function CHEBY1 was used. The function CHEBY1 generates coefficients for a Type 1 Chebyshev FIR (finite impulse response) filter. When the CHEBY1 function is given the center frequency, number of coefficients, and passband ripple, the function returns the computed coefficients of the filter. The coefficients can then be used to filter the test sentence using another built-in Matlab function (FILTER).

To help visually illustrate the effect of the passband filtering, a display technique known as a "periodogram" was used. This technique uses the Fast Fourier Transform (FFT) on progressive segments of the test sentence. By using the FFT, you get a "snap-shot" of the frequency components in the tested section of the audio sample. Each segment represents a portion of time. By plotting the frequency components verses time on a three-dimensional graph, a 3-D representation of the test sentence can be seen.

An example from this process, illustrated in Figure 3.1, shows the result of filtering the test sentence "Hello, how are you?". The figure represents the spectral composition of the audio sample over a period of time, and shows the frequency bands that have been opened up for data transmission.

The stopband filter was implemented as a cascade of a low pass and a high pass filter. These filters are Chebyshev Type I filters [13], whose general frequency response is illustrated in Figure 3.2. Chebyshev Type I

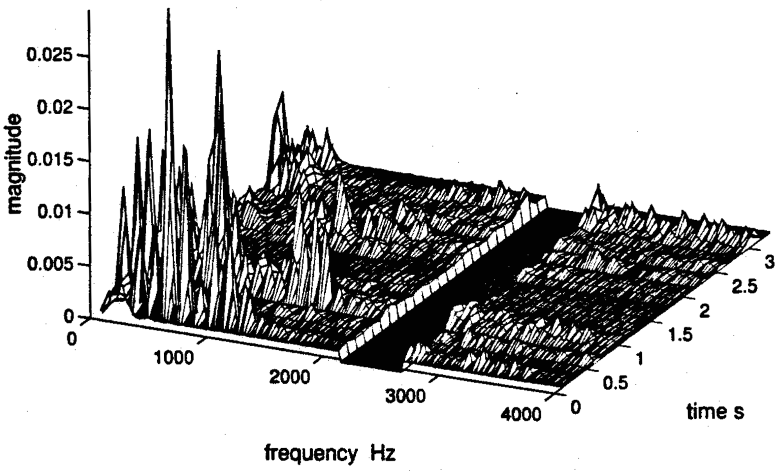


Figure 3.1 Power Spectrum of Filtered Voice Sample
(Filter - 2.4kHz Center Freq, 500Hz Bandwidth)

filters are pole-zero filters that exhibit equiripple behaviour in the pass-band and monotonic behaviour in the stopband.

To remove incoming data from the audio channel before passing it to the listener, the audio channel needed to be filtered with a bandstop filter. This bandstop filter was similar to the filter used at the transmitter to remove part of the audio. By a series of audio listening tests, it was determined that a stopband attenuation of -50 dB was adequate to make the sound of data signals inaudible. The stopband filter at the voice receiver is required to reject data signal power at the frequencies allocated to data transmission. At a maximum input of ± 2.5V (the limit of the A-D/D-A converters used in the experiment), the data signal would be reduced to less than ± 0.008V. Reduced over 300 times, the sound becomes inaudible.

To choose the center frequency and filter bandwidth for the experimental system, a subjective quality of speech test was used. The International

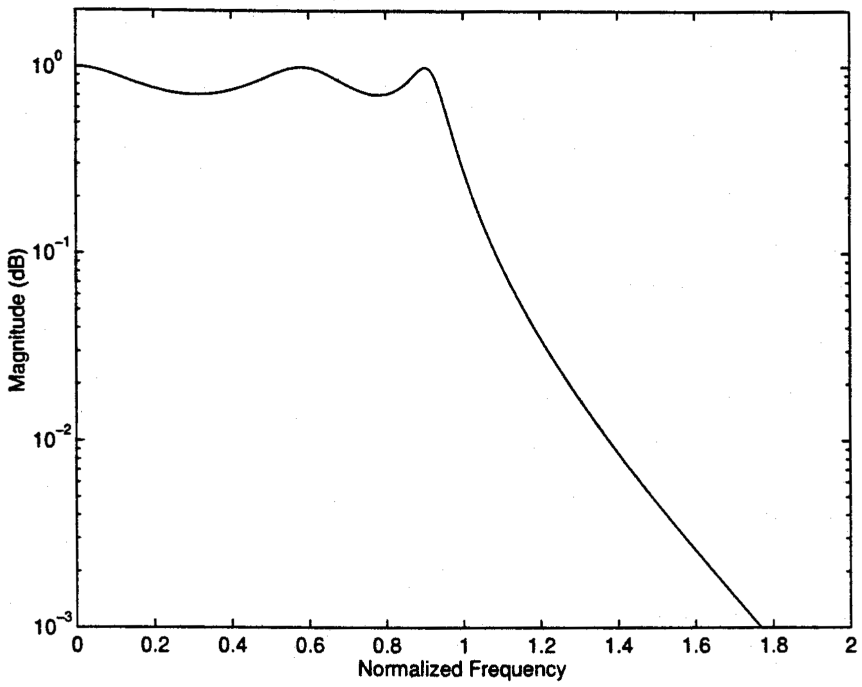


Figure 3.2 Frequency response of Chebyshev Type I filter

Telecommunications Union - Telecommunications Standards Sector (ITU-T) provides a recommendation on subjective evaluation of voice over a transmission system. ITU-T Recommendation P.800 "Methods for Subjective Determination of Transmission Quality" provides guidelines for the test. When the "Absolute Category Rating" is used, each test sentence is compared against five subjective levels. These levels (and their corresponding scores) can be seen below.

Table 3.1 Absolute Category Rating Scores

Quality of Speech	Score
Excellent	5
Good	4
Fair	3
Poor	2
Bad	1

After each of the test sentences were filtered using the different filters, the team of unbiased observers were asked to rate each test sentence. The numerical representations of the ratings were averaged yielding the mean listening-quality opinion score, or simply MOS. Since the results would be used to choose the "optimum" filter bandwidth and center frequency, the filter who's test sentences had the highest average MOS score would be chosen.

Table 3.2 shows the overall results of the test.

Table 3.2 Quality of Filtered Voice Signal

	Center Frequency			
stopband	1000	1500 Hz	2000 Hz	2500 Hz
500 Hz	2	4	4.5	4
1000 Hz	1	3	4	3

As expected, as the bandwidth of the filter is increased, the voice quality is decreased. It becomes a trade-off between voice quality and data bandwidth. It can also be noted that a center frequency near 2 kHz produced the best results, while a filter bandwidth of 1000 Hz still produced a good voice quality.

These results did not confirm the original assumption. A filter with a center frequency around 1600Hz did not provide the best audio quality. Instead, a filter with a center frequency around 2kHz produced better results.

A final date bitrate of 600b/s was chosen. This would require approximately 900Hz of bandwidth. The carrier frequency was centered at 2400Hz. As the bitrate was a multiple of the carrier frequency, it was easier to implement the modulator.

3.2 Filter Requirements for Data Transmission

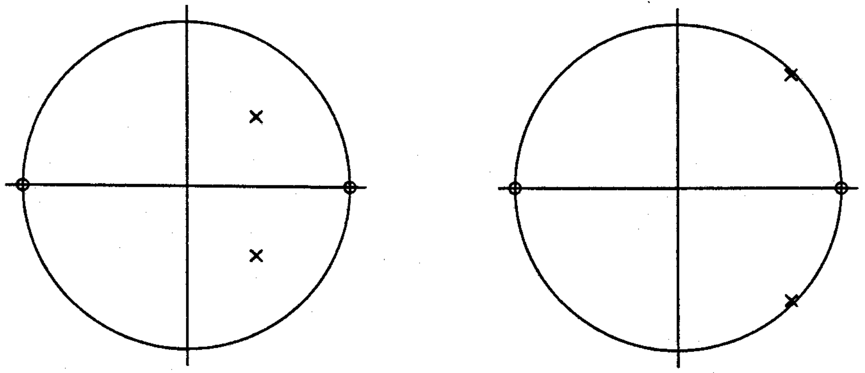
Selecting the transmission amplitude of the digital signal required a trade-off between the complexity of the filters and the noise susceptibility. While a large digital signal would require a higher order filter at the receiver to remove a large portion of the digital signal from the voice path, a small digital signal would be more susceptible to the effects of telephone line noise. The analog-to-digital converter (ADC) of the DSP used eight-bit PCM to sample the incoming signal. As discussed previously, the SNR of the PCM signal degrades linearly with the decrease in the incoming signal amplitude. Figure 2.6 shows this relationship. Consequently, as the amplitude of the digital signal is reduced, so is the SNR. The system implemented in this thesis used 8-bit linear PCM for simplicity. Because of the differences in signal to noise ratios between LPCM and CPCM, a production system would require 12 or 13 bit LPCM to give similar performance as 8-bit CPCM.

To reduce the computational power necessary to implement steep digital filters, it is possible to design low order digital filters with very steep roll-off at the expense of passband ripple. Steep roll-off filters are necessary to minimize the amount of "wasted" bandwidth in the transition zones between the data and the voice. These filters rely on both zeros and poles to give the steep frequency response. An IIR filter does not generally have linear phase response, but can have a much steeper roll-off when compared to a FIR filter of similar complexity. Linear phase is equivalent to constant delay. It is necessary to have near linear phase to minimize the complexity of the receiver. Without near linear phase, receiver compensation of the received signal must be done before using a non-coherent demodulator. This compensation would then require more processing time and possibly a larger processor. The lack of linear phase was deemed a necessary trade off in order to obtain a filter with the desirable frequency response that would operate on the digital signal processor in real time. A non-coherent demodulator is used as it is generally less complex then coherent demodulation. It does not require the demodulator to recover the incoming signal phase. It also introduces a slight (about 3 or 4dB) degradation on the overall system performance.

By placing the poles of the IIR filter very close to the unit circle, a low order filter with a steep roll-off is realized. However, as the poles are placed closer to the unit circle, the impulse response begins to have large oscillations that last for many samples. While this has little effect on a sinusoidal input, it does have a large effect on noise impulses that enter the system. The multiplication that occurs in the digital filter can end up overflowing the DSP's math registers as the values oscillate to large extremes. The DSP then clips the multiplication values to the DSP's minimum and maximum values. Consequently, clipping noise is introduced into the filter stream. Of course, this noise is unwanted and will only degrade the response of the filter.

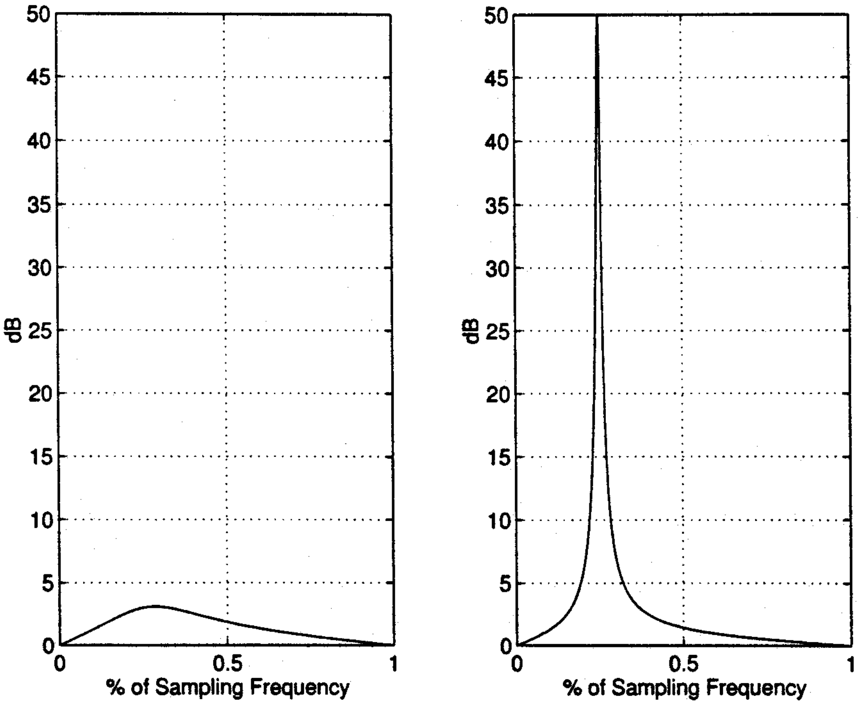
Figures 3.3, 3.4, and 3.5 show the differences between an IIR filter with the poles near the unit circle and one with the poles not near the unit circle. The left side graphs of Figures 3.3, 3.4, and 3.5 had the poles placed at $0.6 \angle \pm 45^o$ with zeros at ± 1. The right side graphs had the poles placed at $0.98 \angle \pm 45^o$ with the zeros also placed at ± 1. The figures on the left show the dampered response of an IIR filter with the poles not near the unity circle. While this frequency response is not very useful, it does show how the impulse response dies out very quickly. On the right side is an IIR filter with poles very close to the unity circle. The frequency response shows a graph with a large amount of rejection outside the passband. Figure 3.5 shows the trade off as the impulse response oscillates much longer and is larger in amplitude.

Also of importance is the finite precision of the digital signal processor. As the digital filter's coefficient values approach unity, the precision of calculations needed to maintain the filter response within 5% of the planned filter response is increased. As the DSP has fixed precision in its multiplication registers, the filter response will differ significantly from the planned filter response as the coefficients approach unity. When filters are designed with very steep roll-off characteristics, the internal memory elements of the filter



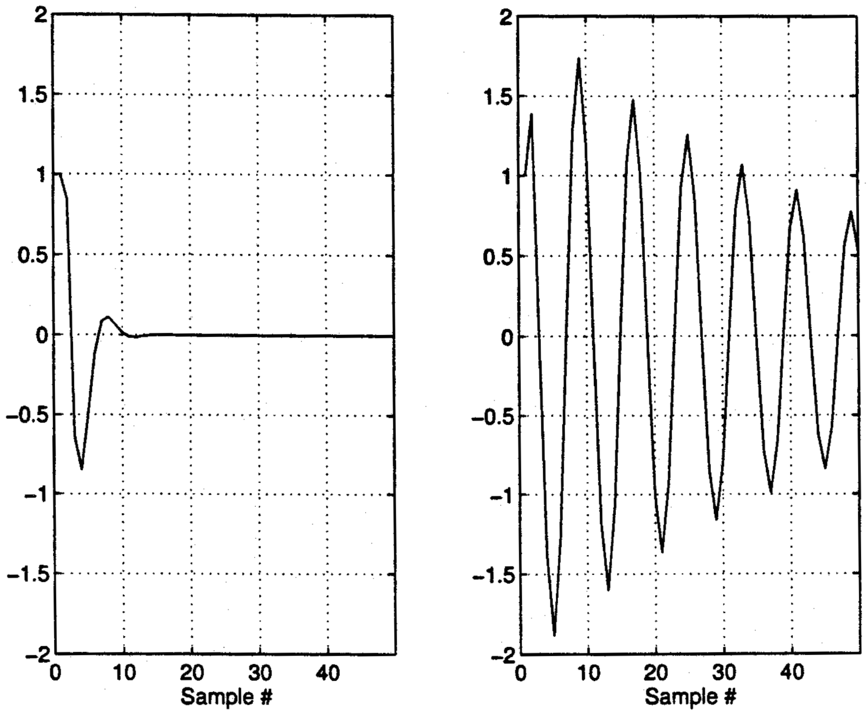
a) “Low Q” Poles b) “High Q” Poles

Figure 3.3 Pole-Zero Plot of IIR Filter Coefficients



a) “Low Q” Poles b) “High Q” Poles

Figure 3.4 Magnitude Frequency Response of IIR Filters



a) "Low Q" Poles b) "High Q" Poles

Figure 3.5 Impulse Response of IIR Filters

that contain the intermediate filter calculations can fluctuate very dramatically. It is the precision of these internal memory elements that typically limit the values that can be used as the digital filter coefficients.

The Matlab simulation routines were written to closely mimic the operation of the digital signal processor. The Matlab mathematical operations were performed in the same order as the dsp program. Memory elements were given the same precision so that the simulation could test for any expected element overflow. The routines were written to help estimate the processor's abilities to perform all the necessary calculations in real time. It was soon determined that if FIR filters were used, the processor would not be able to complete the filter operations in real time. When FIR and IIR filters are compared, the use of the IIR filter results in a 90% saving in DSP instructions. However, the trade off also resulted in unwanted non-linear phase response and long impulse response times. This introduced more unwanted noise into the system which affected the systems performance. While the simulations determined that the simultaneous voice and inband data concept was practical using IIR filters, its performance in real world conditions was yet unknown.

3.3 Data Modulation

The previous section described the filtering of a stopband section in the telephone's available bandwidth. A modulation scheme is next used to insert data into the frequency band filtered by the DSP. For good performance from the system, the modulation scheme must produce a spectrum which rolls off sharply and requires a minimum of band-limiting before transmission. This band-limiting (filtering) usually takes place after the modulation module and before the final output stage. Care must also be taken to include the capabilities and limitations of the digital signal processor before deciding on the final modulation scheme. To minimize processor load, the modulation technique chosen should permit non-coherent demodulation techniques. Coherent demodulation requires a digital phase locked loop which would require too many instruction cycles from the digital signal processor. The modulation methods which can be non-coherently demodulated include on-off keying (OOK), frequency shift keying (FSK), minimum shift keying (MSK), Gaussian minimum shift keying (GMSK) and differential modulation schemes such as differential quadrature phase shift keying (DQPSK).

FSK has, at best, a null to null bandwidth of 1.5 Hz/bit/s [2]. This applies to the special case of FSK when the modulation index is set to 0.5 and the phase is kept continuous. The special case is referred to as MSK. A 600 b/s data stream modulated using MSK would require 900 Hz of bandwidth. Typically, GMSK has a null to null bandwidth of 1.25 Hz/bit/s [2]. Thus a 600 b/s data stream modulated using GMSK would only need 750 Hz of bandwidth. While the numbers quoted refer to ideal modulation schemes, practical systems will add a buffer zone on either side of the modulated signal to accommodate non-ideal bandstop filters.

GMSK has a narrower 3dB corner frequency than MSK and a sharper roll-off in the frequency domain. This compact spectrum allows the data transmit filter to be eliminated without significant interference in the voice signal spectrum.

GMSK uses a baseband Gaussian filter for the data. The use of the Gaussian filter on the baseband data is computationally easier than using a bandpass channel filter on the data after it has been modulated. The number of coefficients in the baseband filter will be significantly less than the number needed in the bandpass channel filter. As a result, the baseband filter will require fewer dsp instructions to operate.

The Gaussian filter has the following impulse response.

$$h_t(t) = \frac{1}{\sqrt{2\pi}\alpha T} exp(\frac{-t^2}{2\alpha^2 T^2}) \tag{3.1}$$

The parameter α is related to the 3-dB bandwidth (B_b) and the symbol period (T) of the filter. It is defined as

$$\alpha = \frac{\sqrt{ln(2)}}{2\pi B_b T} \tag{3.2}$$

The Gaussian filters have a smooth transfer function and also a smooth impulse response. This in contrast to Nyquist filters [14] which have impulse response zero crossings at time intervals equal to the symbol rate and a transfer function which is truncated in the frequency domain. Figure 3.6 compares the impulse responses of the Gaussian and Nyquist filters. The Nyquist filter's impulse response can be seen crossing the zero threshold at adjacent symbol times while the Gaussian filter still has substantial amplitude at the adjacent symbol times peaks.

In digital FIR filter design, Nyquist filters are generated from formulas

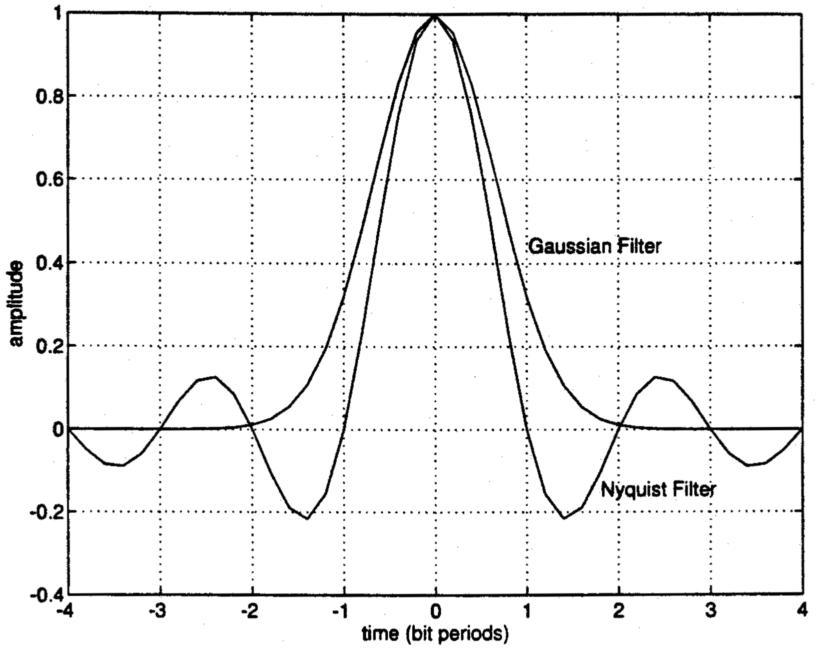


Figure 3.6 Impulse responses of Nyquist and Gaussian Filters

based on modified $\sin(\omega_c t)/(\omega_c t)$ functions where ω_c is the ideal low-pass cutoff frequency. Theoretically, the Nyquist formula would return an infinite number of coefficients, truncating the number of coefficients as necessary in order to have a manageable number. With the Nyquist design formula $\sin(\omega_c t)/(\omega_c t)$ response, truncation is usually performed as the coefficient values approach the quantization size of the digital processor's registers.

It should be noted that the Gaussian pulse shaping filter does not satisfy the Nyquist criterion for zero inter-symbol interference (ISI). Inter-symbol interference is caused when energy from an adjacent symbol effects the current symbol. If a filter does not have an impulse response that goes to zero at the time of adjacent symbol peaks, it will create inter-symbol interference and degradation in the system performance. The degradation is a measure of the necessary increase in E_b/N_o to maintain the same bit error rate (BER) in the system for a given noise level. E_b/N_o is a ratio of signal energy per bit to in-band noise spectral density. Figure 3.7 [5], shows the expected degradation in the Gaussian system when compared with an ideal coherently demodulated antipodal (binary) system, such as MSK or binary phase shift keying (BPSK). The degradation is plotted against the normalized bandwidth of the Gaussian filter (B_bT_s).

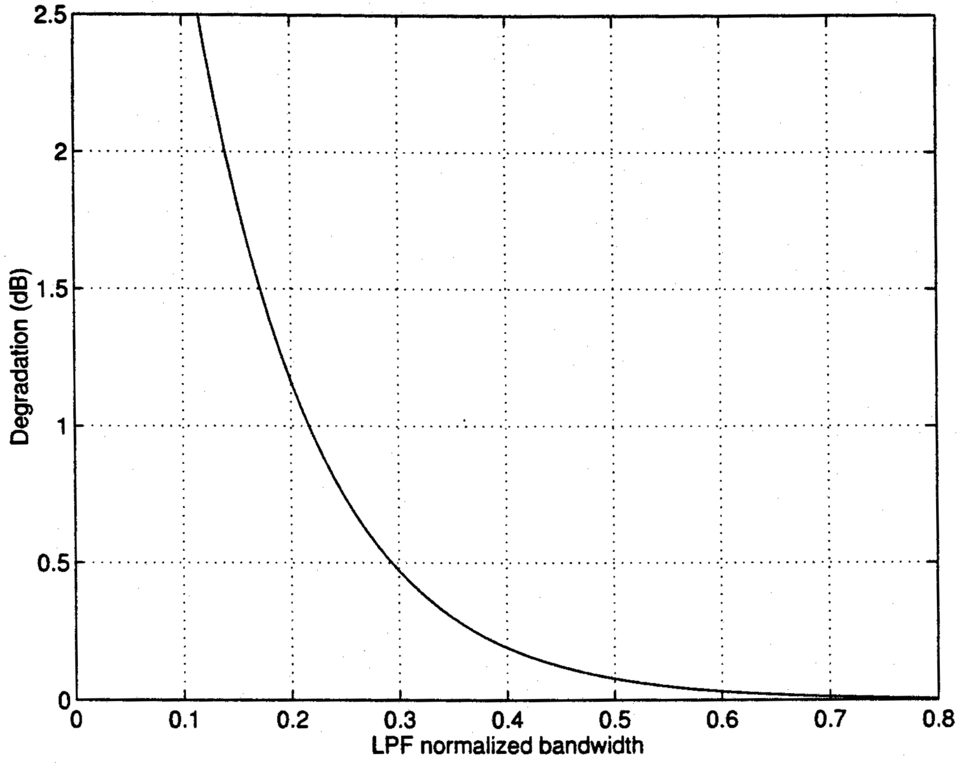


Figure 3.7 BER performance degradation of GMSK referenced to ideal antipodal system

From Figure 3.7 we can conclude that the degradation in performance as a result of the Gaussian filter is smaller than 1 dB for values of B_bT_s greater that 0.22 . A value of 0.5 was deemed sufficient as it provided a large amount of out of band roll off while resulting in less than a 0.25 dB implementation loss.

A GMSK system may be created by first filtering the base band signal using a Gaussian low pass filter, and then modulating with a FM modulator with a modulation index of 0.5 [5]. Thus the filtered baseband signal is modulated using a MSK modulation method.

The length of the digital FIR filter needed to accurately model a Gaussian low pass filter can be several symbols long. When the difference between the sampling frequency and the data rate is large, there are a large number of samples in each symbol period. Each sample is represented by a tap in the filter delay line. For each tap in the FIR filter, the processor must perform a multiply and add during each sampling period. As the number of taps grow, the processor reaches a point where it can not complete the necessary instructions in one cycle time (sample period). This makes it difficult to generate GMSK using a baseband FIR filter due to the limitation of processor power. It was necessary to use an approximation to GMSK that did not require the same number of DSP instructions. The GMSK approximation must maintain a continuous phase and a constant amplitude envelope in order for the demodulation process to use a non-coherent algorithm. The continuous phase helps shape the transmit spectrum while the non-coherent demodulation will help simplify the demodulation process at the receiver.

The GMSK approximation method chosen was based on the sinusoidal table look up method. The approximation does not have the ISI normally produced by GMSK. Each baseband symbol starts at the limits (either 1 or -1) and no localized DC drift or symbol offset is present. Normal GMSK does tend to have small fluctuations in the localized DC average of the modulated signal. Because the approximate algorithm does rely on a series of mathematical additions, there should not be any DC offset present as it could slowly sum toward infinity.

Approximate Gaussian low pass filtering of the baseband data is the first step in generating GMSK. Depending on the values of the current and previous data bits, four waveforms are generated, as seen in Figure 3.8. Each waveform is labelled by an ordered pair of numbers which represent the current and previous input data bit $[d_{k-1}, d_k]$. These four waveforms are calculated at the program start up time and stored in a look-up table. They are stepped though one sample at a time, at a sampling rate of 16 kHz and with a resolution of 8 bits per sample. The result of the first table lookup is a baseband signal which now has smooth transitions between the bits, instead of the abrupt changes of the previous square-wave baseband signal.

The amplitude output of the first look-up table is in the range of −1 to 1. The continuous time equivalent filtered baseband waveforms can be given by:

$$m(t) = \cos(-\pi) \quad if(d_{k-1}, d_k) = (0,0) \tag{3.3}$$
$$m(t) = \cos(\tfrac{\pi t}{T_s}) \quad if(d_{k-1}, d_k) = (0,1) \tag{3.4}$$
$$m(t) = -\cos(-\tfrac{\pi t}{T_s}) \quad if(d_{k-1}, d_k) = (1,0) \tag{3.5}$$
$$m(t) = \cos(-\pi) \quad if(d_{k-1}, d_k) = (1,1) \tag{3.6}$$

where d$_{k-1}$ and d$_k$ are the current and previous data bits, and T_s is the sample period.

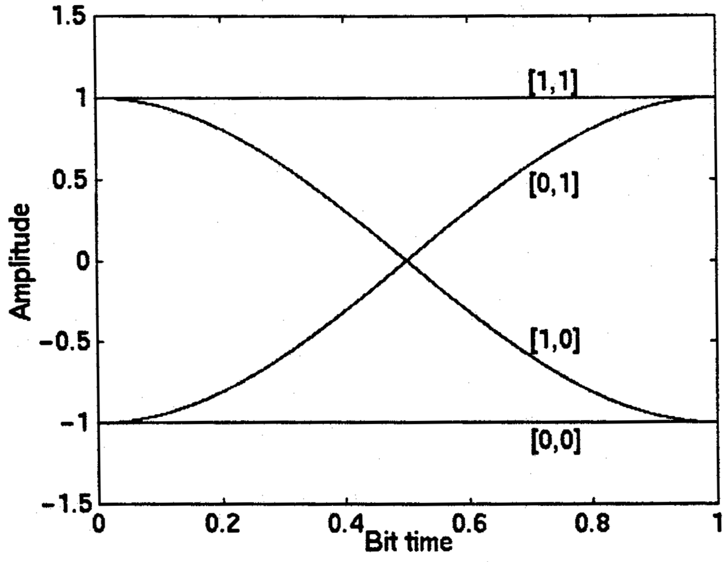


Figure 3.8 Four Possible Amplitude Waveforms for GMSK Approximation

The next step is to perform MSK modulation on the signal. The modulation index h is set to 0.5 and the approximate baseband signal is used to modulate the carrier to generate the GMSK approximation.

The MSK can be generated using the equation [15]:

$$s(t) = A_c \cos(\omega_c t + D_f \int_{-\infty}^{t} m(\lambda) d\lambda) \tag{3.7}$$

where m(t) is the amplitude signal generated by the Gaussian filter and $D_f \int_{-\infty}^{t} m(\lambda) d\lambda$ is the phase offset $\theta(t)$. The equation can be rewritten as:

$$s(t) = A_c \cos(\omega_c t + \theta(t)) \tag{3.8}$$

In discrete time systems, integration is calculated by a running sum. The expression $D_f \int_{-\infty}^{t} m(\lambda) d\lambda$ is equivalent to $D_f \sum_{-\infty}^{t} m(t)$ with the exception of a constant multiplier. This means that a single unit delay element, shown as z^{-1}, along with an addition unit can be used to generate the running sum.

The difference between the value of $\omega_c t$ at any two sampling times will be constant as both Δt and ω_c are constants. Also, because the difference between $D_f \sum_{-\infty}^{t} m(t)$ at any two adjacent sampling times can be calculated from the incoming baseband signal, it is easier to calculate $\Delta(\omega_c t) + \Delta\theta(t)$. The running sum of this expression will then generate $\omega_c t + \theta(t)$. This running sum is shown in Figure 3.9 as a z^{-1} block and an addition unit. To generate the term $\Delta(\omega_c t) + \Delta\theta(t)$, a second look-up table is used. Knowing that the maximum and minimum frequency output of the VCO is generated when m(t) is equal to ± 1, and knowing that a simple linear frequency relationship exists between the value of m(t) and the frequency of the VCO output, a second look-up table can be generated to translate the current value of m(t) to the appropriate value of $\Delta(\omega_c t) + \Delta\theta(t)$.

The running sum of the output of the second look-up table is now used to generate the final output. By calculating the cosine of the running sum, the approximation to GMSK is generated. To decrease the processing time necessary for calculating the cosine function, a third look-up table is used to calculate an approximation for the cosine value. The cosine look-up table is calculated in one degree increments at program start-up.

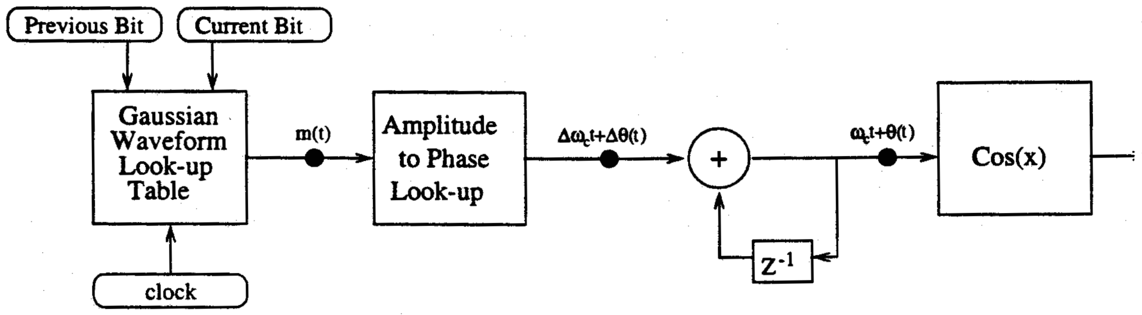


Figure 3.9 Generation of GMSK Approximation

Power spectral densities were generated using Matlab to compare the spectrums of GMSK (BT=0.3), MSK (BT=∞), and the approximation to GMSK. In Figure 3.10, we can see that both GMSK and its approximation roll off faster that MSK. Figure 3.10 shows a GMSK response where the bandwidth of the Gaussian low pass filter had been set to 0.3 of the data bandwidth. It should be noted that the bandwidth of the GMSK Gaussian filter may be reduced to roll the frequency spectrum off faster at the expense of increased ISI.

3.4 Data Demodulation

The digital demodulation of GMSK uses a delay and multiply technique [12] to recover the original digital signal from the sampled stream as illustrated in Figure 3.11. This technique is also known as differential decoding. The key component of the non-coherent demodulator is the $-\pi/2$ delay element.

When continuous phase is used to generate MSK, the simplified MSK transmitted signal $s(t)$ from Eq. 3.7 can be represented at the receiver by

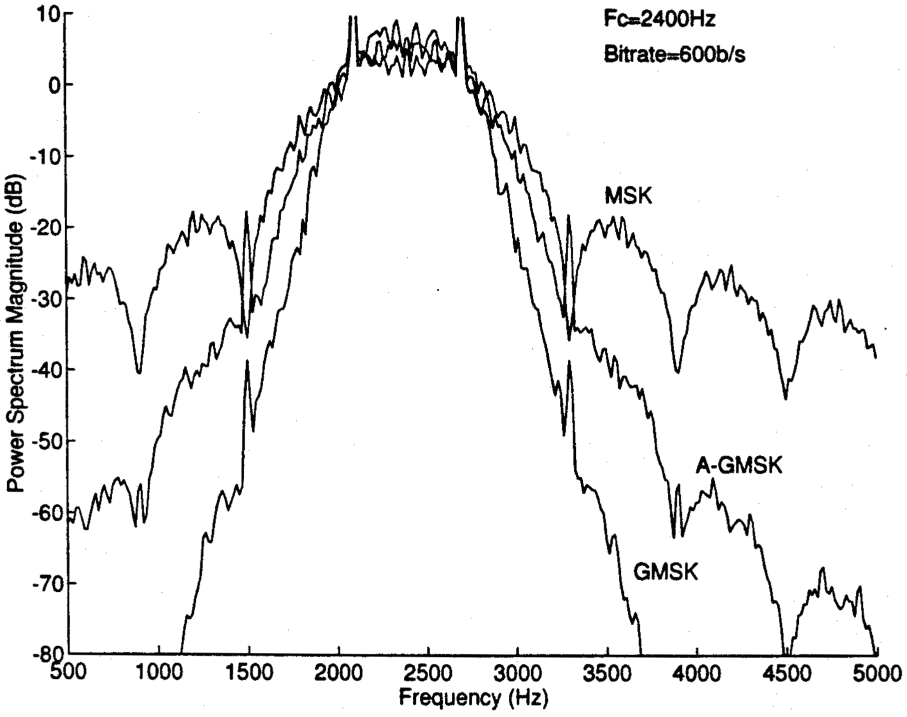


Figure 3.10 Power Spectral Density of GMSK (BT=0.3), MSK ($BT = \infty$), and GMSK approximation (A-GMSK)

$r(t)$:

$$r(t) = \cos[(\omega_c \pm \delta\omega)t + \theta] \tag{3.9}$$

where $\delta\omega$ represents the frequency deviation, and θ represents the phase offset. In a binary MSK system, the frequency of the signal is either $\omega_c - \delta\omega$ or $\omega_c + \delta\omega$ depending on whether a 0 or 1 was sent. Thus the two states of transmitting logic 1 or 0 are represented in the transmitted frequency $\omega_c \pm \delta\omega$

For differential detection, the received signal is delayed by $\frac{1}{\omega_c * 4}$ seconds and the resulting delayed signal $r(t-\tau)$ is multiplied by $r(t)$ to produce $p(t)$. The delayed input signal $r(t-\tau)$ is represented by:

$$r(t-\tau) = \cos[(\omega_c \pm \delta\omega)(t-\tau) + \theta] \tag{3.10}$$

where τ is the signal delay.

The multiplier output p(t), as seen in Figure 3.11, is then the product of Eq. 3.9 and Eq. 3.10 which results in:

$$p(t) = r(t)r(t-\tau) \tag{3.11}$$

$$p(t) = \cos[(\omega_c \pm \delta\omega)t + \theta]\cos[(\omega_c \pm \delta\omega)(t-\tau) + \theta] \tag{3.12}$$

$$p(t) = \frac{1}{2}\cos[2(\omega_c \pm \delta\omega)t - (\omega_c \pm \delta\omega)\tau + 2\theta] + \cos[(\omega_c \pm \delta\omega)\tau] \tag{3.13}$$

When τ is chosen such that $\omega_c\tau = \pi/2$, and when the remaining signal is filtered using a low pass filter to remove the double carrier frequency component, the signal can be simplified to

$$q(t) = \cos(\pi/2 \pm \delta\omega\tau) = \pm\sin(\delta\omega\tau) \tag{3.14}$$

The signal $q(t)$ can then be demodulated using a zero crossing detector. When $q(t)$ is a positive voltage, it represents a logic 1. When $q(t)$ is a negative voltage, it represents a logic 0. Timing signals must be reconstructed on the receiving end as differential-detection non-coherent demodulation provides no clocking information. As phase locked loops are difficult to implement on a digital signal processor, other timing recovery techniques are used. This is described in Chapter 4.

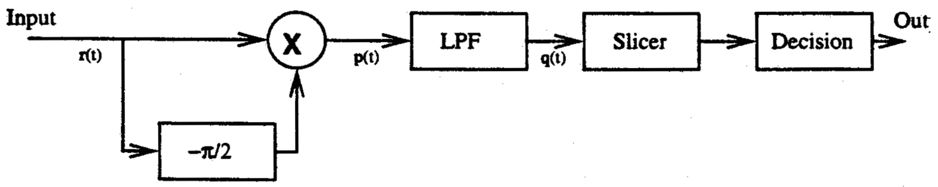


Figure 3.11 GMSK Non-Coherent Demodulator

As stated above, a delay element of length τ is used to create a $-\pi/2$ phase shift in the input signal with respect to the carrier frequency ω_c. The required delay, however, may or may not be an exact integer multiple of the sampling period. For this reason, we need to use a one-zero digital filter [4], as seen in Figure 3.12, to generate the appropriate delay.

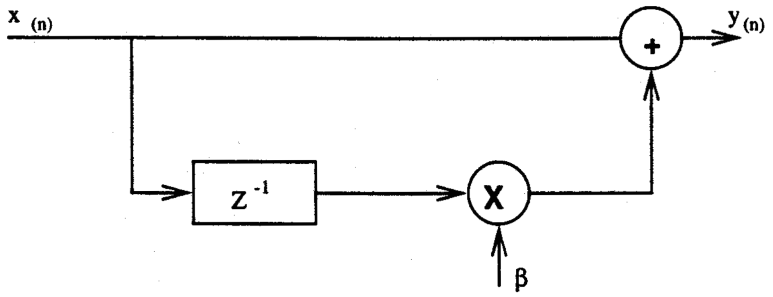


Figure 3.12 One Zero Phase Delay Filter

The response of the phase delay filter is given by:

$$y(n) = x(n) + \beta x(n-1) \tag{3.15}$$

This equation is used to generate an FIR filter with a frequency response of:

$$F'(\omega) = 1 + \beta e^{j\omega} \tag{3.16}$$

The purpose of this filter is to introduce a precise group delay τ to the received signal where τ is defined as:

$$\tau = \frac{-d\phi(\omega)}{d\omega} = \text{group delay} \tag{3.17}$$

Appendix A describes the development of the one zero phase delay filter and how it can be used to develop a group delay of $-\pi/2$.

After plotting the frequency and phase responses of Eq. 3.16, the resulting responses are shown in Figures 3.13 and 3.14. The coefficients are chosen so the phase response at the carrier frequency is equal to -90^o or $-\pi/2$ radians (as seen in Figure 3.14).

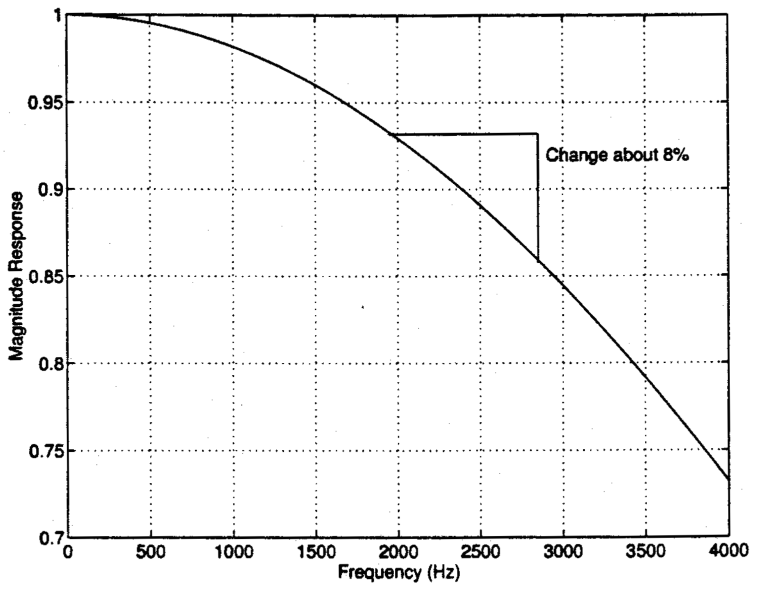


Figure 3.13 Magnitude Frequency Response of Single Zero Delay Filter

By proper coefficient scaling, a gain of approximately one for the frequency range of the filter can be achieved. Figure 3.13 shows that in our signal transmission band (f_c=2400Hz , BW=900Hz), the gain of the single zero delay filter changes by less than 8%, (or $0.6dB$). This will introduce amplitude differences in the transmission band and cause the introduction of amplitude modulation noise into the system.

3.5 The Requirement For Up-Sampling

Implementation of the system in software presented some additional problems. The sampling rate of the A-D/D-A was 8 kHz while the carrier fre-

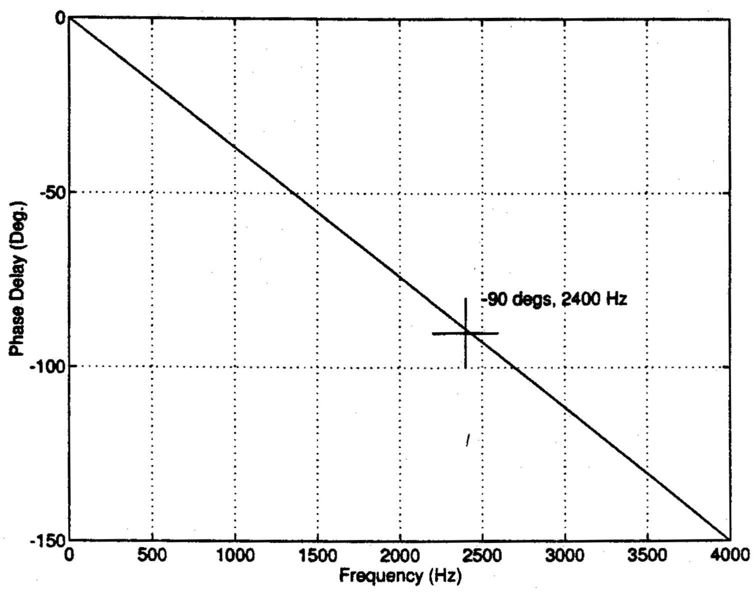


Figure 3.14 Phase Frequency Response of Single Zero Delay Filter

quency of the GMSK signal was in the range of 1600 Hz to 2400 Hz. When the received data signal reached frequencies above 2 kHz, the system did not function as desired. Equation 3.13 shows that a signal at twice the carrier frequency exists after multiplication. With a carrier frequency over 2 kHz, the double frequency content is aliased down into the low frequency range. This makes it very difficult to filter the double frequency signal. The simple solution would be to increase the sampling rate of the system. However, in order to maintain real time processing capabilities, the voice processing had to be maintained at a sampling rate of 8 kHz. Any increase in sampling rates above 8 kHz would not allow time for the DSP to finish the filter calculations required for each input sample.

To solve this problem, the filtered data input signal (i.e. no voice signal) was up-sampled using interpolation before multiplication took place. The interpolation formula (3.18) was used to convert the 8 kHz input signal up to the 16 kHz sampled signal that was used for multiplication. The differential detector demodulator then ran at a sampling frequency of 16 kHz while the voice bandpass filters ran at 8 kHz.

The interpolation formula is developed in Appendix B and can be simplified to

$$x(n-1) = \frac{1}{3}x(n) + x(n-2) - \frac{1}{3}x(n-3) \tag{3.18}$$

or in the Z-transform domain:

$$z^{-1}X(z) = \frac{1}{3}X(z) + z^{-2}X(z) - \frac{1}{3}z^{-3}X(z) \tag{3.19}$$

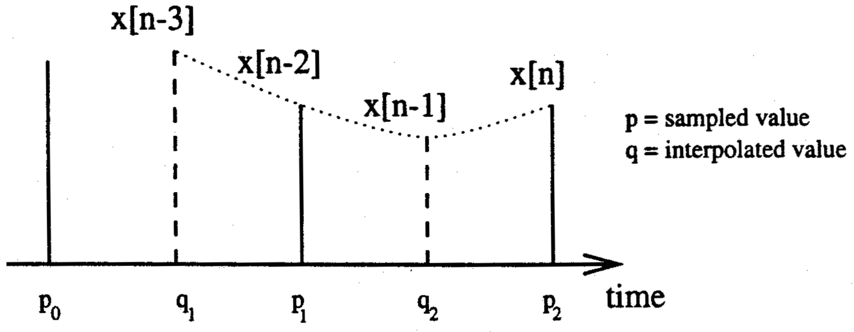


Figure 3.15 Interpolation of the Sampling Frequency from 8 kHz to 16 kHz

As seen in Figure 3.15, input values with even powers of Z come from the input A-D, while those with odd powers are generated by interpolation. The quadratic interpolation depends on the current value as well as the second and third previous values.

Because the differential detector is running at a clock speed equal to twice the sampling rate, it will process two data samples per sampling interval. Thus, when a new sample is received, the second newest sample can be calculated using the newest, third newest, and fourth newest sample. The fourth newest sample will be the interpolated value from the previous sampling period. It is the newest and second newest (interpolated) values that are then used in the differential detector.

This solution provides a good estimate of the missing value between samples. This second order approximation is fast to calculate and provides answers which are adequate for the demodulation scheme used. This interpolation calculation helps set the ground work for multirate digital signal processing. The technique can often be used to reduce the workload on the DSP by performing baseband functions at a lower sampling frequency.

3.6 Multirate Digital Signal Processing

As with all technology in our ever changing world, as time goes on, technology evolves to new and better levels of design. New techniques are discovered to do things better and faster. Such was the case with this project.

While multirate digital signal processing has been around for quite a few years, its use has been rather limited. While its use can have a dramatic impact on processor time and resources, it has quietly been gathering momentum and acceptance in the DSP world.

The key obstacle in this project was the lack of processor power to complete the entire computer algorithm in real time. Instead, short cuts were taken, compromising performance for real time operation. This chapter will demonstrate the power of multirate digital signal processing and its ability to cut operation cycles and boost the performance from the TMS320C31.

The first key component of miltirate DSP is the half-band filter. The half-band filter is truly a unique component with a lot of potential. The concept of the half-band filter is quite simple. If the digital signal of interest occupies a position in only one half of the frequency band, then it is possible to filter the other half of the frequency band and then reduce the sampling frequency. This deliberately uses the effects of signal aliasing to help move the interesting signal to a lower sampling frequency. Figure 3.16 shows the progression from the original signal to the final signal shown in Figure 3.18.

Figure 3.16 begins the sequence by showing the frequency spectrum of the original incoming signal. The half-sampling frequency is given by π.

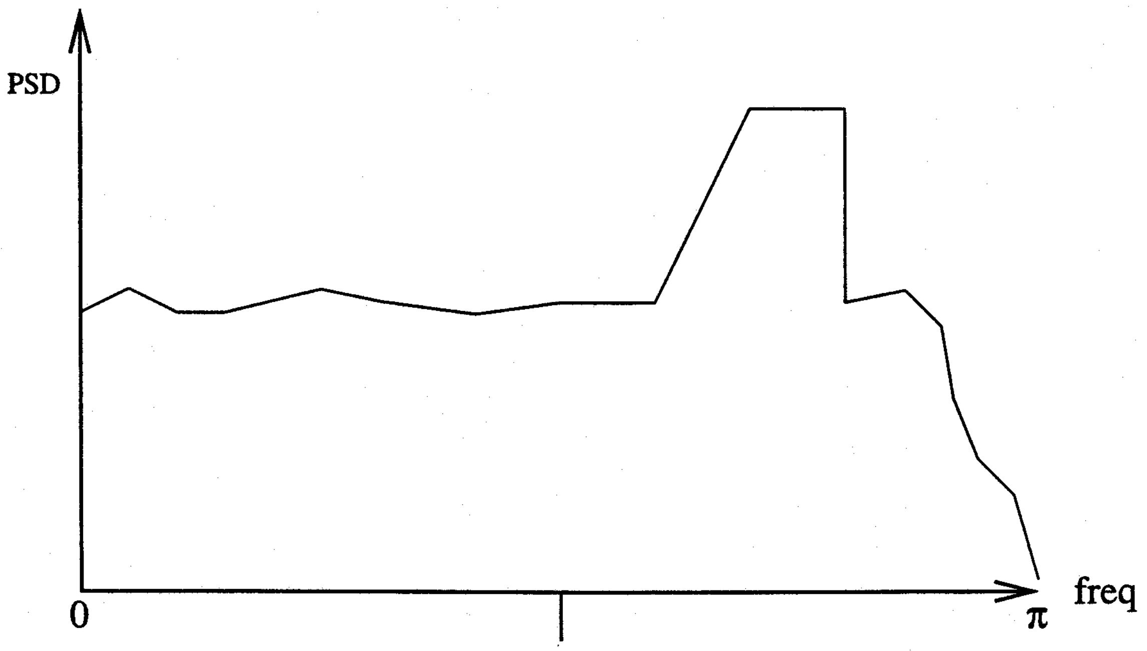


Figure 3.16 Original Signal Frequency Response

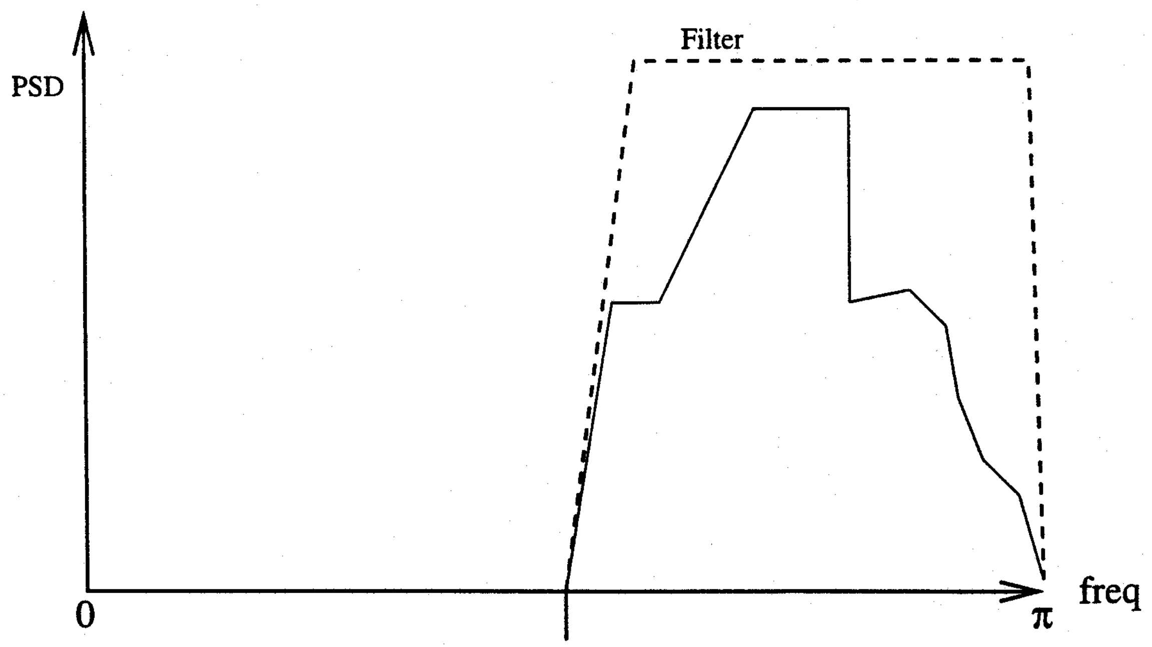


Figure 3.17 Frequency Response of Signal With Filtering

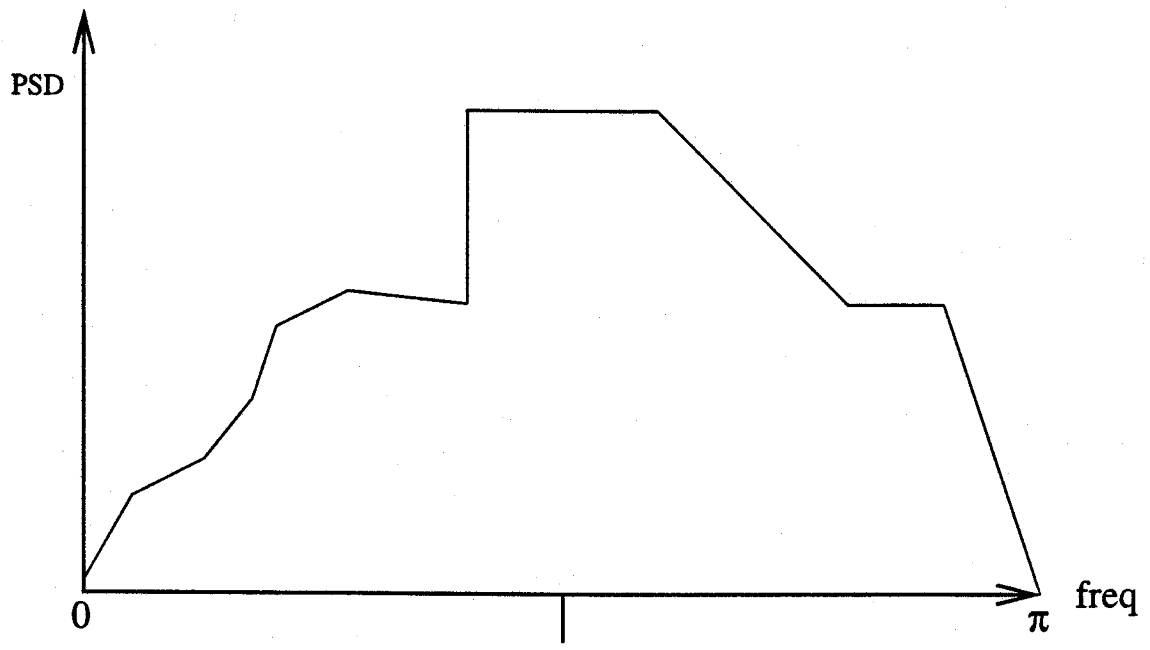


Figure 3.18 Frequency Response of Signal After Downsampling

The second Figure (3.17) shows the signal after a highpass filter is used to remove the frequency component between DC and one-quarter of the sampling frequency. Figure 3.18 completes the sequence by showing the result of lowering the sampling frequency by one-half. The signal is now reversed in frequency as a result of the frequency aliasing which took place during the downsampling. This frequency reversal is usually not a problem for modulation schemes which modulate around a common carrier. The demodulated signal will only have to be complemented to get the correct result. Incidentally, the downsampling algorithm is extremely simple, as we simply throw away every other sample.

The actual make-up of the half band filter is what really sets it apart from "normal" lattice style digital filters. The block diagram of a two-path signal stage half-band filter can be seen in Figure 3.19. The two-path filter uses two simple filter elements $P_0(Z^{-2})$ and $P_1(Z^{-2})$. It also uses a single delay element Z^{-1}. On exit from the half-band filter, the signal is downsampled to lower the sampling rate by a factor of two.

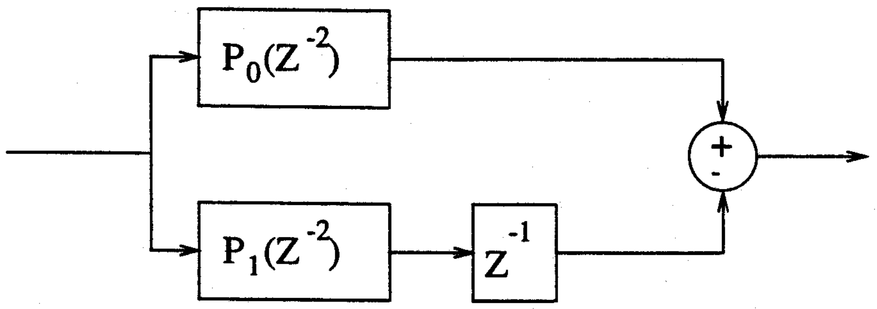


Figure 3.19 Block Diagram of Two Path Half-Band Filter

Each simple filter block $P_i(Z^{-2})$ has a transfer function given by:

$$\frac{1+\alpha Z^2}{Z^2+\alpha} \tag{3.20}$$

which when drawn in a schematic diagram can be shown as Figure 3.20

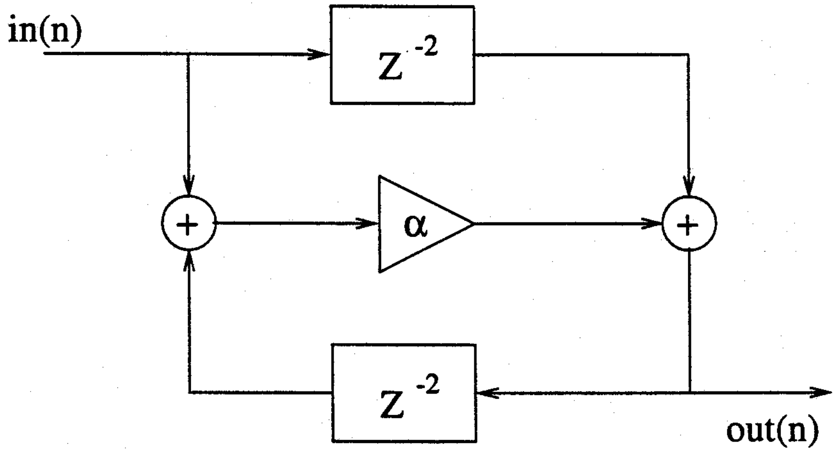


Figure 3.20 Block Diagram Of All-Pass Filter Structure

This structure results in an all pass filter with near linear phase response. With the insertion of the Z^{-1} delay element in one of the paths, the phase shift between the two paths is near 0^o in the pass band and near 180^o in the filter band.

When combined with the other simple filter block and the unit delay element, the transfer function of the half-band filter is given by:

$$H(Z) = \frac{1+\alpha_0 Z^2}{Z^2+\alpha_0} \pm \frac{1}{Z}\frac{1+\alpha_1 Z^2}{Z^2+\alpha_1} \tag{3.21}$$

After multiplication, this equation is represented by:

$$H(Z) = \frac{\alpha_0 Z^5 \pm \alpha_1 Z^4 + (1+\alpha_0\alpha_1)Z^3 \pm (1+\alpha_0\alpha_1)Z^2 + \alpha_1 Z + \alpha_0}{Z(Z^2+\alpha_0)(Z^2+\alpha_1)} \tag{3.22}$$

The transfer function exhibits 5 poles and 5 zeros when the values of the α's are chosen carefully. The α's must be chosen so that the equation remains stable. The function only requires 2 multiplications. This great response returned by this filter with only 2 multiplications allows for near linear phase filters with minimal computational processing.

The half-band filters will filter out either the upper or lower band of frequencies. The $\pm$ are to be set accordingly to which half of the band is of interest to the particular application.

The process can be made even more efficient if we take advantage of a relationship between resampling filters and resampling ratios known as the Noble Identity. To quote the expert in the field[6],

> "This relationship states that a filter formed as a ratio of rational polynomials in Z^M followed by an M-to-1 downsampler maintains the same input-output relationship when the two operations are commuted subject to changing the polynomial indeterminate from Z^M to Z^1".

In its simplest form, this means that we may put the downsampling before the filter providing we replace each Z^M with Z^1. The obvious advantage to this is that we now perform all our filtering at a lower sampling rate, and thus, require much less processing power to perform the same overall operation as before.

Figure 3.21 shows a step by step progression from the original half-band filter structure to the modified half-band filter. In the transition from Figure 3.22 to Figure 3.23, the two downsampling units and the unit delay are replaced with a simple commutator. It is obvious that the downsampling unit is operated by simply throwing away every second sample. With the delay, each half section would receive every second sample, offset by one.

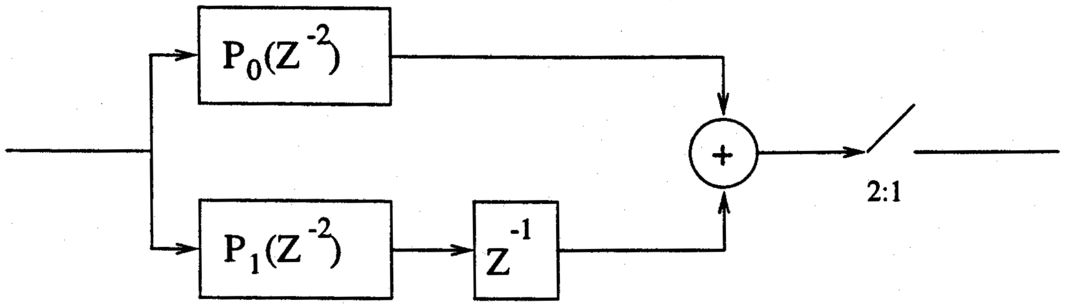


Figure 3.21 Original Structure of Half-Band Filter

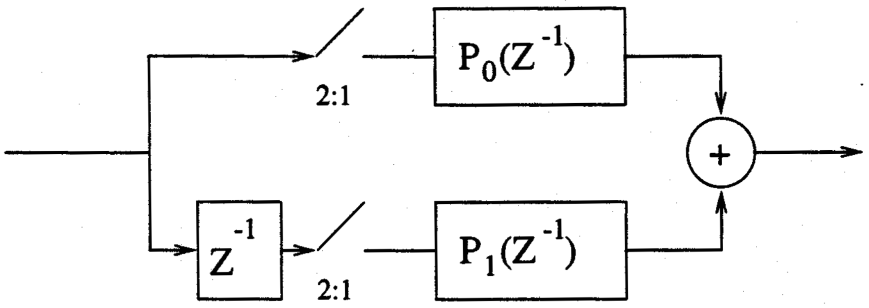


Figure 3.22 Half-Band Filter After Carrying Downsampling Through

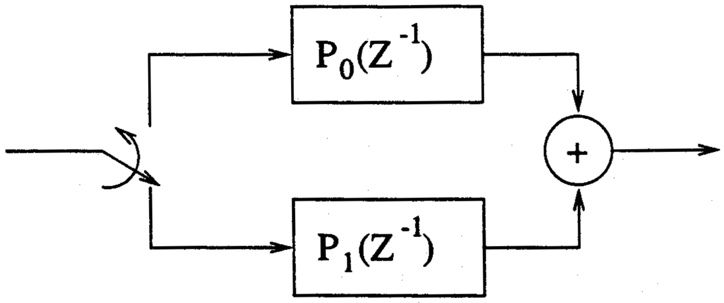


Figure 3.23 Half-Band Filter After Replacing Delay Element

The final result of this rearranging results in a filter structure which is extremely "instruction" efficient in generating the half-band filter. The process only takes one multiplication and 2 additions per input sample.

A typical frequency response from a half-band filter can be seen in Figure 3.24. The values of the filter in this example are set to $\alpha_0 = 0.153978$ and $\alpha_1 = 0.60945$.

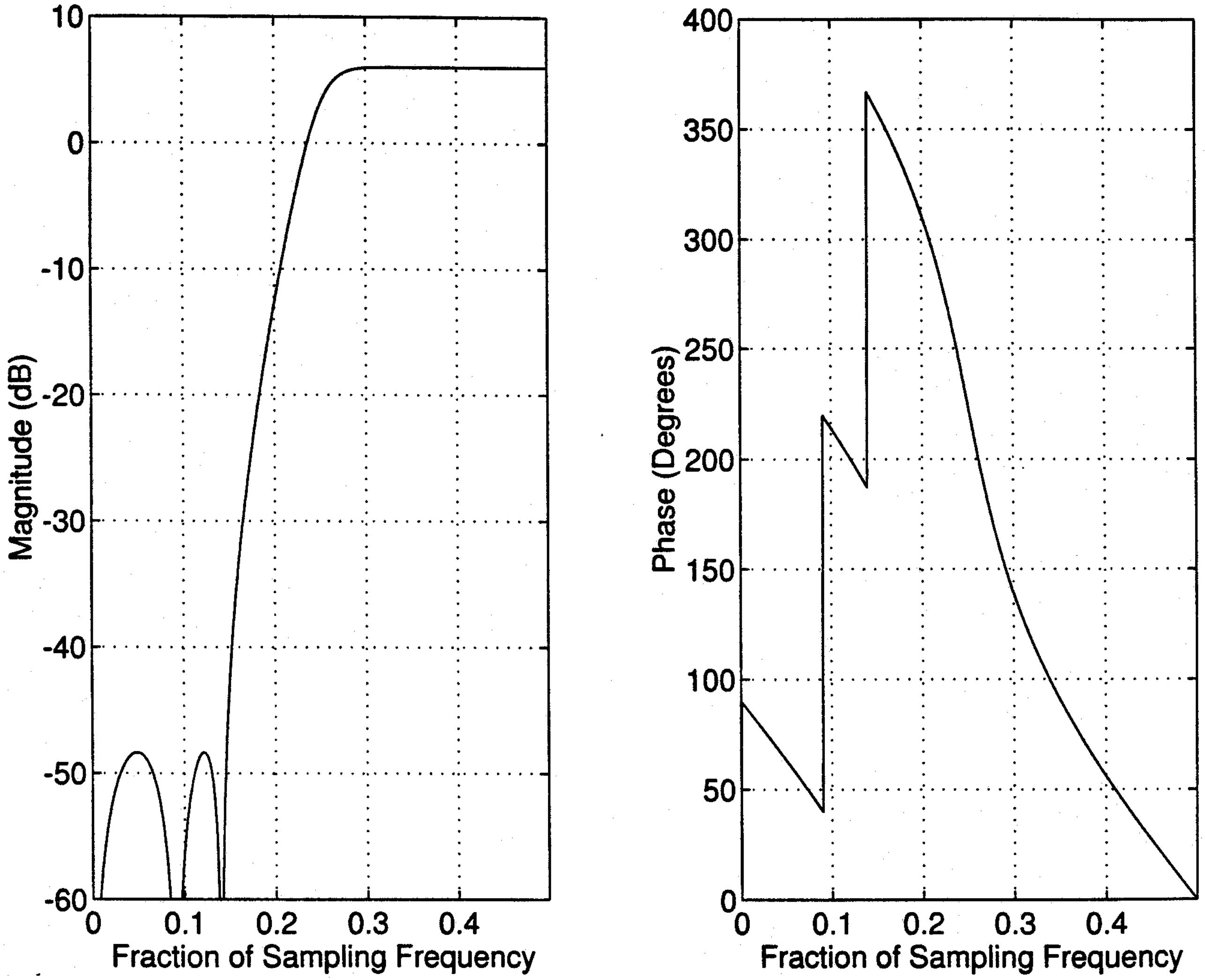


Figure 3.24 Typical Response of a Halfband Filter

The band limited signal of interest in this project is the in-band data which has a bandwidth of approximately 900 Hz centered around 2400 Hz. The sampling frequency of the system was 8000 Hz which put the π sampling frequency at 4000 Hz. The original system with its complicated demodulation scheme required approximately 2000 instructions per input sample to filter and demodulate the GMSK data signal.

In order to lower the instruction count, our data signal would have its sampling frequency lowered several times in order to take advantage of the multirate processing techniques. Figure 3.25 shows the progression of the input signal through the half-band filters and towards the demodulator.

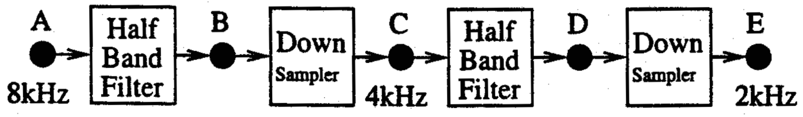


Figure 3.25 System Block Diagram for Half-Band Filters.

The letters in between each block can be used to compare the output of each block with the expected frequency response plots in Figure 3.26.

Our original signal enters the system at letter 'A'. The first half-band section filters off the signal between 0 Hz and 2 kHz. The sampling frequency is reduced by half after the letter 'B'. The sampling frequency is now at 4 kHz and the data signal now resides around 1.6 kHz. The signal has been frequency reversed by the effects of the downsampling. The signal continues through another half-band filter and sampling stage. At letter 'E' the signal has gone through another frequency reversal and is an exact copy of the original data signal, now centered at 400 Hz.

The sampling frequency at the output of the filtering is 2 kHz. The demodulation can now be accomplished without putting a large burden on the DSP. Because the demodulation of each bit is now spread across 4 sampling periods (8 kHz to 2 kHz sampling rate change), the demodulation scheme can be more complicated, and it can include some algorithms to help eliminate noise. This may include narrower bandpass filters. If the same demodulation scheme is used, the effective instructions per sampling period will change from the original 2000 to 800 instructions/sample.

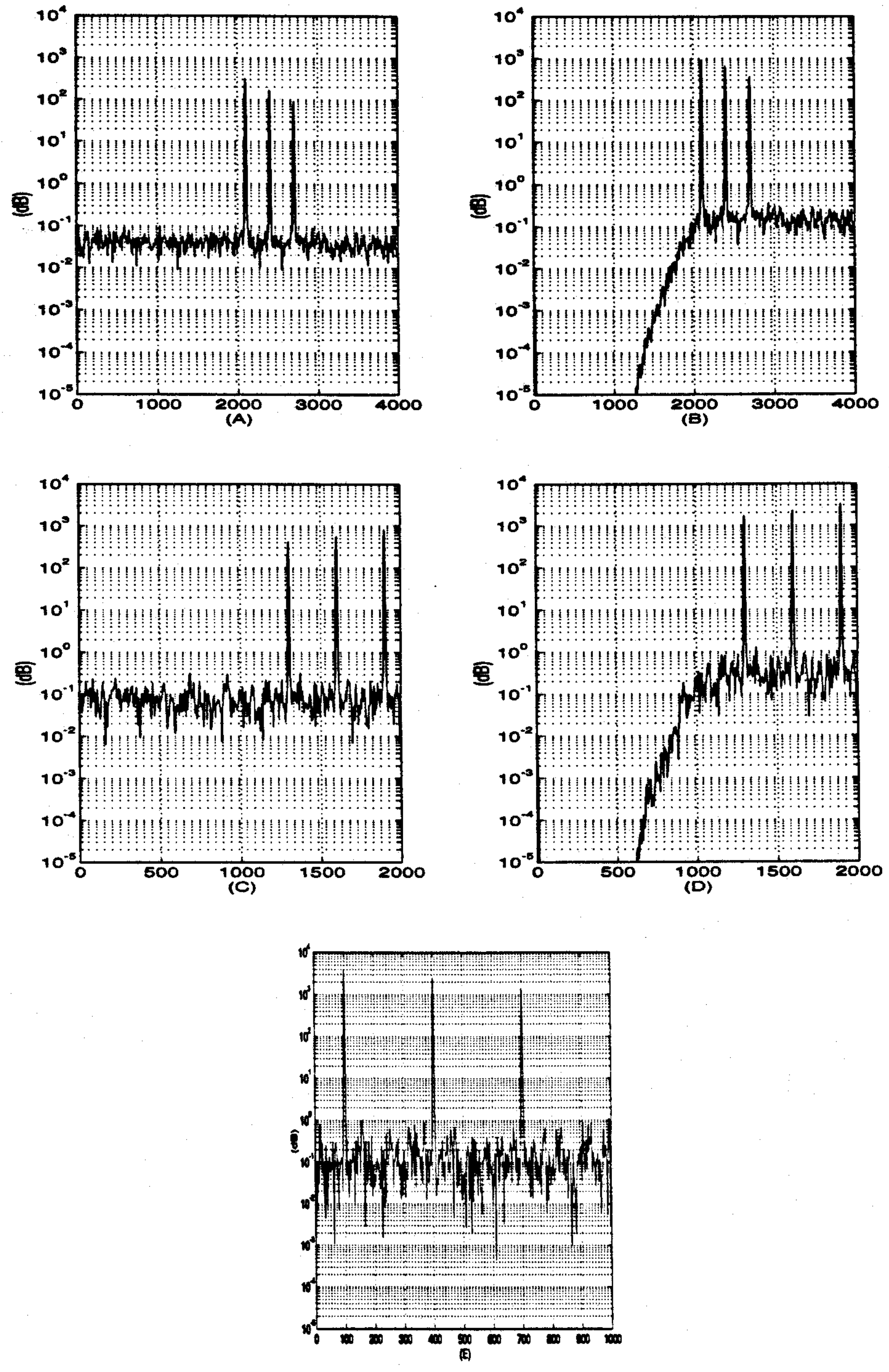


Figure 3.26 Frequency Response of Each Block of Half-Band System with 3 input sinusoidal tones

4. Implementation

4.1 The Elf Plug-In Card

The Elf card is a high speed multimedia DSP processor manufactured by Atlanta Signal Processors Inc.(ASPI), of Atlanta, Georgia, USA. The Elf card offers programmable solutions that can implement a wide variety of signal processing algorithms. The Elf DSP card includes a Texas Instruments TMS320C31 floating point DSP, 1 Mbyte of random access memory (RAM), and a telephone line RJ-11 interface. It uses a Crystal Semiconductor CS4215 stereo high-quality 16-bit A-D/D-A which can run at a speed between 5.5 kHz and 48 kHz. The A-D/D-A uses a delta-sigma modulation technique and can provide an 80 dB signal-to-noise ratio for large signals (this will be less for small signals).

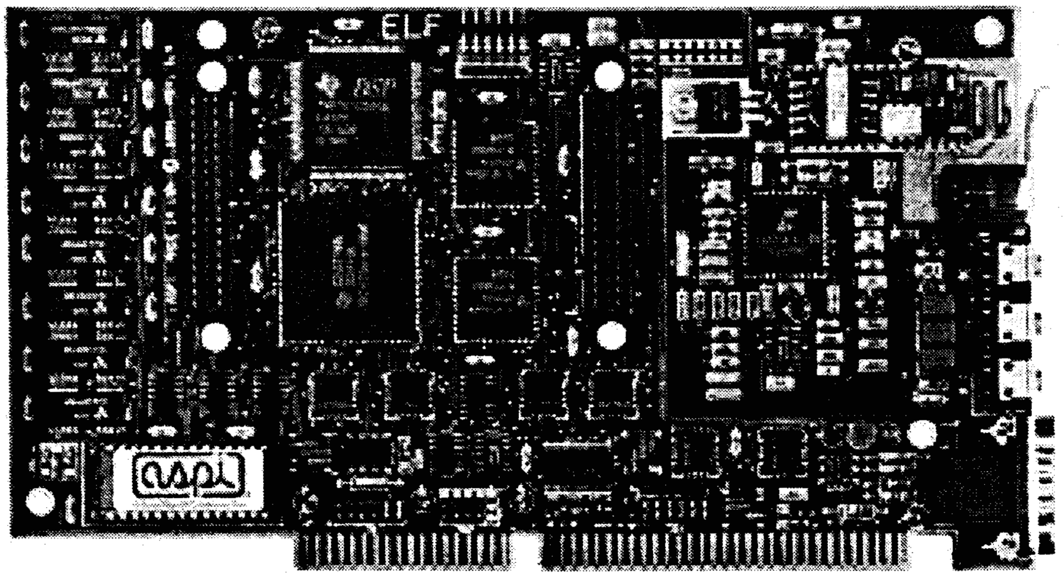


Figure 4.1 Elf Plug-In Card

The Elf card plugs into a 16-bit AT (ISA) bus expansion slot on a host Intel compatible PC. Communications with the host computer is provided by both an internal UART interface (COM port) located on the Elf board, and several high speed DMA data exchange registers which are accessible by the ISA bus.

A large portion of the TMS320C31's high performance is due to internal bus structure and parallelism. The separate program, data, and DMA buses allow for parallel program fetches, data accesses, and DMA accesses. The DMA controller is supported with both an address and data bus allowing it to perform direct memory addressing and is able to move data blocks in parallel with those occurring on the data and program buses.

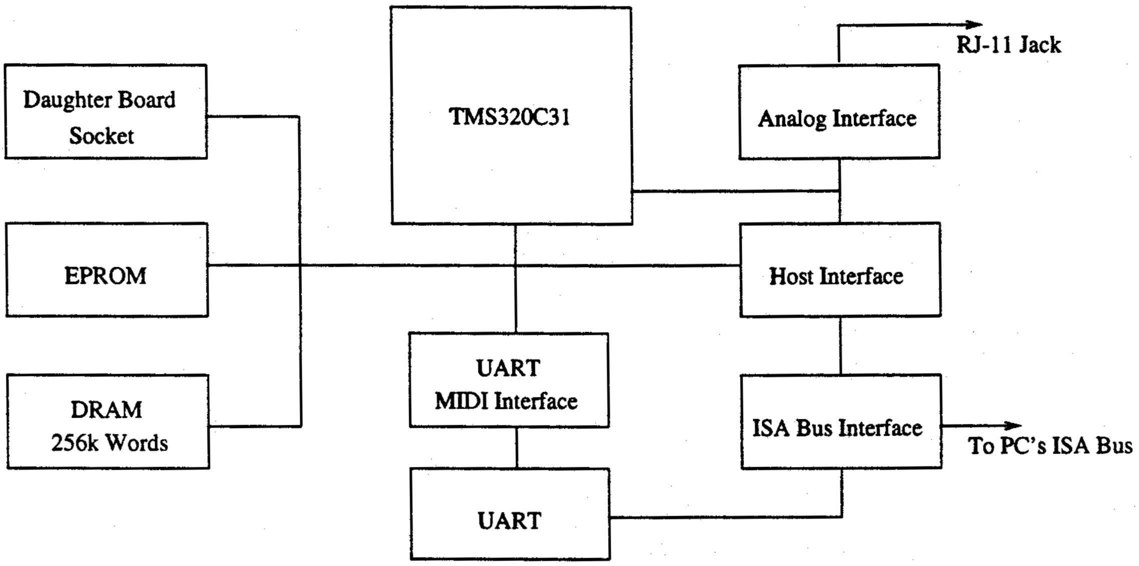


Figure 4.2 Elf Card Architecture

Figure 4.2 is a block diagram of the general architecture of the Elf card. The host interface consists of several port addressable registers for use by the DMA and debugging software. The on-card EPROM is programmed to monitor for loading, debugging, and running of Elf applications. The Elf card provides a convenient daughter board socket for easy attachment of extra peripherals.

The TMS320C31 DSP is clocked at 33 MHz and is capable of 33 million floating point operations per second.

4.2 Telephone Functions

The Elf card provides the necessary interface circuitry for a phone line connection. The hybrid coupler and DC loop circuitry of the telephone station set are provided by the circuitry on the card. The microphone and earphone connections are provided through the use of a mini-DIN stereo headphone connector as seen in Figure 4.3. The connections to a standard telephone handset RJ-12 connector is also seen in Figure 4.3.

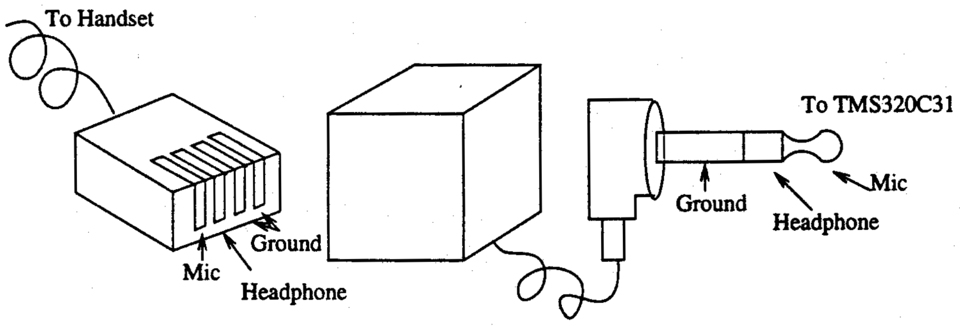


Figure 4.3 Connection of Handset to Elf

By using the telephone handset, a look and feel of a real telephone can be maintained. The underlying circuitry is however quite different from that found in a normal telephone. Figure 4.4 gives a block diagram overview of the Elf "telephone". The immediate difference recognized between the Elf telephone and a standard residential telephone is the internal use of digital signals. Standard telephones leave all signals in their analog form. The other major difference is the lack of a phone cradle and dialing pad. These components have been replaced with "on-screen" equivalents. The number to be dialed is read from the computer keyboard while the phone is taken off-hook and returned on-hook by key strokes from the computer's keyboard.

The experimental system supported two way voice communications, but

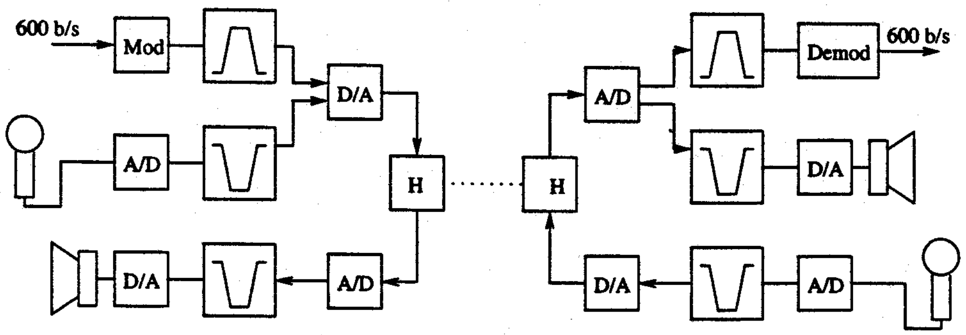


Figure 4.4 Block Diagram of TMS320C31 Telephone

only one way data transmission. The system also only allowed for origination of phone calls from one end. This setup was deemed sufficient to test the concept of the entire system.

4.3 The Receiver

The receiver was implemented in two parts. One high level control program operated on the PC, while another ran at a lower level on the Elf card.

The Elf card is responsible for filtering the incoming signal from the telephone line and splitting it into the voice and data channels. It handles the demodulation of the incoming data signal and sends the demodulated data to the host PC. It also returns the outgoing voice to the telephone network. When the telephone is on-hook, the Elf card also monitors the line for a ringing signal, and alerts the host if the phone was ringing.

The PC displays the incoming data and interprets the user's keyboard inputs. It sends control signals to the Elf board instructing it on which action to take.

4.3.1 The Receiving PC

Upon boot-up, the host PC program clears the screen and provides the user with a window-based environment seen in Figure 4.5. It initializes the data link between the host and the Elf card by setting up the COM port to

9600 b/s, and turning on the TMS320C31 ring detector. The program then enters a loop waiting for the phone to ring. This software loop is shown in Figure 4.6.

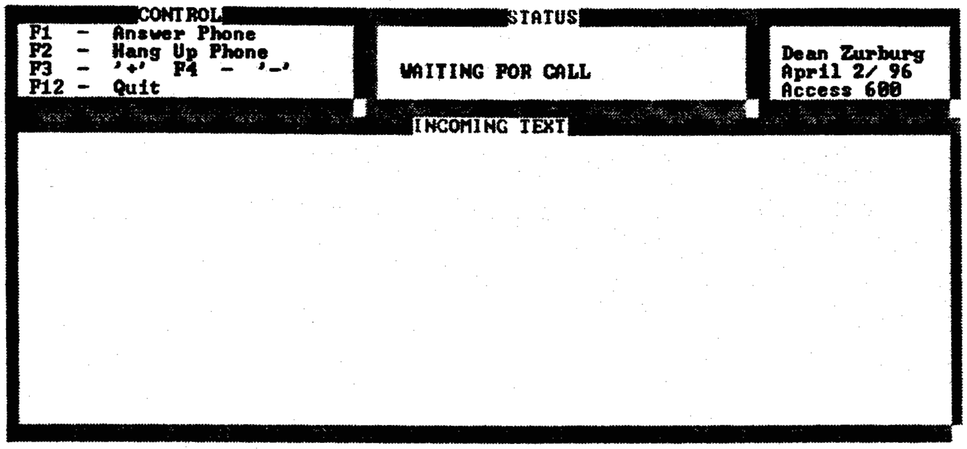


Figure 4.5 Receiving PC's User Interface

If a ring indication is passed to the host PC from the TMS320C31, the user is alerted to the presence of a ring. It then waits for the user to answer the phone by pressing **F1** on the keyboard.

When the user answers the phone, the control character 0x07 is sent by the PC to the Elf card to tell it to answer the phone. Each character is sent using the internal COM port. The host monitors a control bit called the CTS (Clear To Send) and waits for it to go high. If it does, the phone is considered to be off-hook. If it does not go high after a certain amount of time, an error message is displayed indicating that no serial port activity can be detected.

With an off-hook indication, the program goes into a loop in which it waits for either keyboard input, or characters from the Elf card. The characters from the Elf card are displayed in the text window exactly as transmitted. The only keyboard inputs accepted by the program are keys corresponding to **hang-up**, **clear screen**, or **quit**.

If the key for **hang up** is pressed, the program sends a 0x09 to the Elf

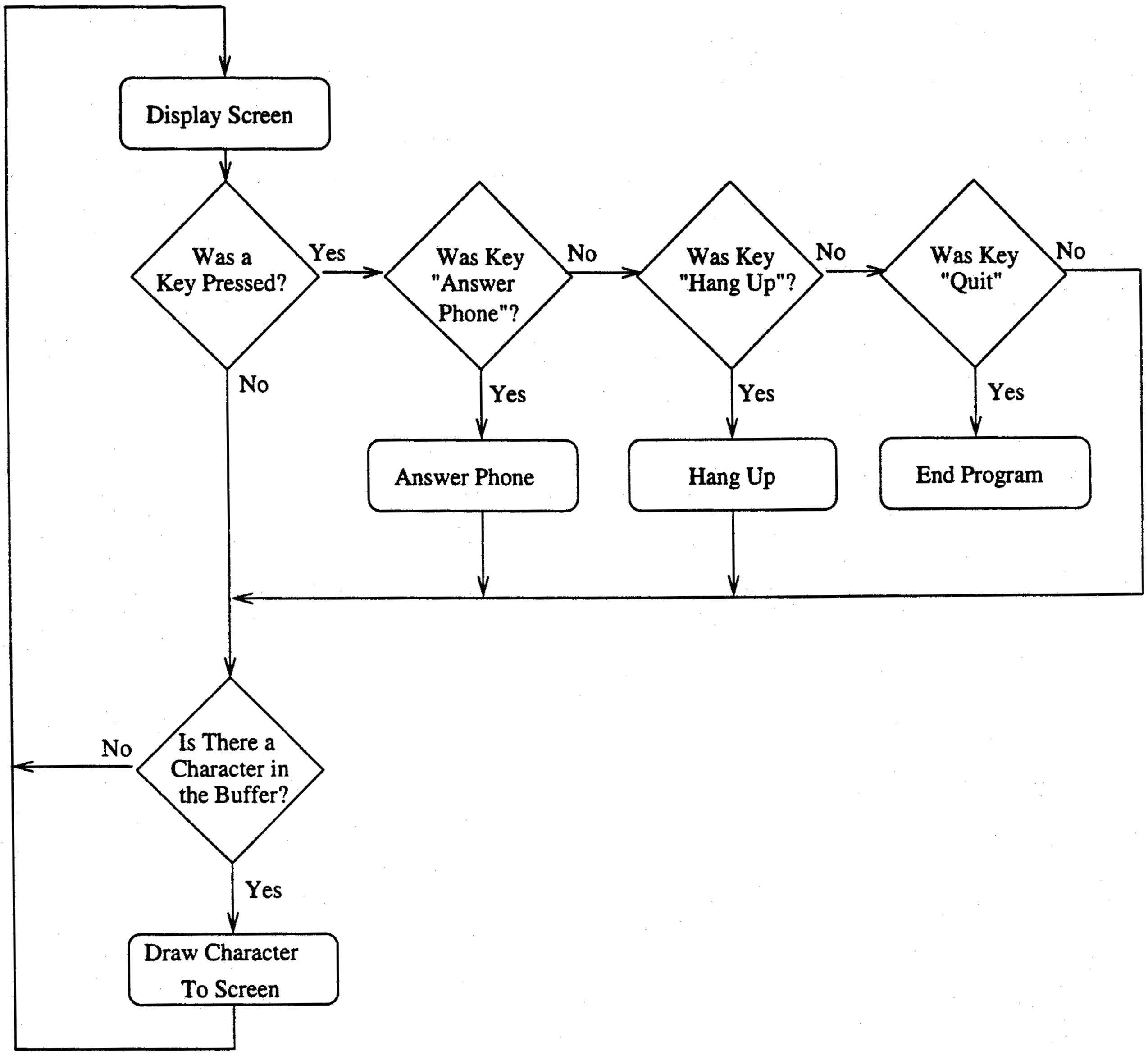


Figure 4.6 Software Flow Diagram for Receiving PC

card, indicating that it wants to hang up the phone. The phone is put on-hook, and the program resets to initial conditions and waits for another phone call. If the key for **quit** is pressed, the system exits back to the DOS prompt.

4.3.2 The Receiving Elf Card

The role of the program on the Elf card in the receiver is considerably more complex then the program running on the PC. It's complexity requires that the TMS320C31 DSP run at close to the maximum load.

A simplified software flow diagram is shown in Figure 4.7. Each block can be broken into many smaller pieces for examination. The demodulation process will be covered later.

After initialization of the data exchange registers, the Elf board begins by waiting for the PC to raise the RTS (Request To Send) flag on the COM port. This indicates the PC wishes to go off-hook. The Elf card complies by raising its CTS flag and taking the phone interface off-hook. It initializes its A-D/D-A chip for 8 kHz sampling, and clears all interrupt flags.

The Elf card enters into a continuous loop executed once every sample period. The loop sits and waits until the A-D raises its flag indicating that a sample is ready in the buffer. The sample is read in and the calculations begin.

The A-D sends the sample to the Elf using a 32 bit buffer. The top 16 bits of the buffer represents the incoming phone line channel (referred to as the right channel), while the lower 16 bits represents the incoming microphone channel (referred to as the left channel). A conversion from 16 bit words to 32 bit words is necessary to continue calculations with the 32 bit DSP processor. Table 4.1 shows how the four 16-bit data bytes are packed into the incoming and outgoing data words.

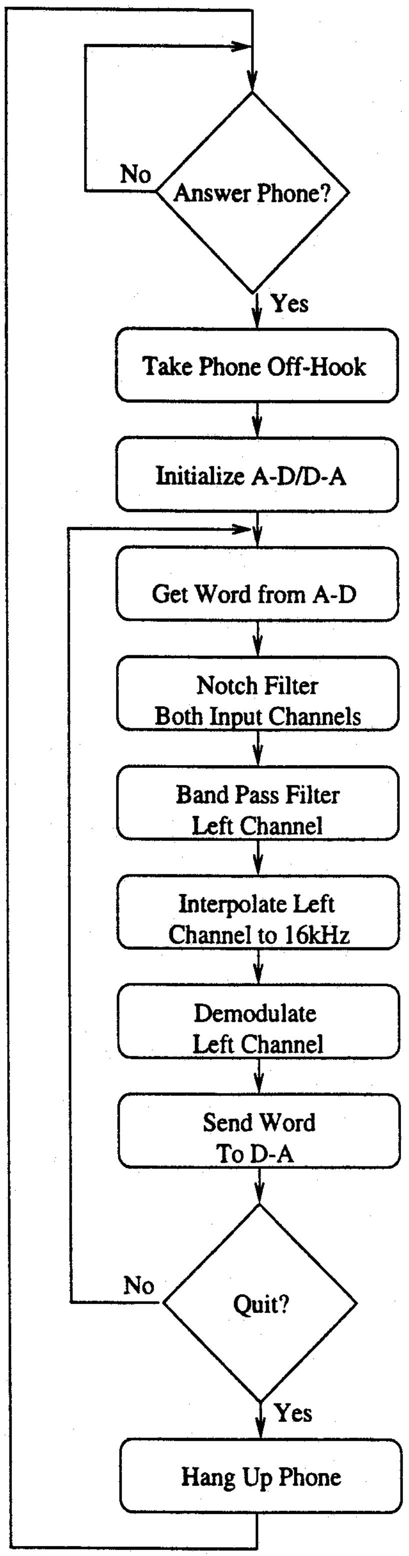


Figure 4.7 Software Flow Diagram for Receiving Elf Card

Table 4.1 Byte Packing of 32-bit A-D/D-A Word

	32 bit word	
	Upper 16 Bits	Lower 16 Bits
From A-D	phone line	microphone
To D-A	phone line	headphone

The right channel is recovered by right shifting the data down 16 times. This preserves the sign bit which is extended into the data bits as necessary. The left channel is recovered by first left shifting the buffer 16 times to remove the upper 16 bits, and then right shifting 16 times to sign extend bringing the sample back to its original value. While it would be easier if we could simply mask off the top 16 bits, the DSP has no instruction for converting a 16 bit signed integer into a 32 bit signed integer. It was therefore necessary to use the shifting method to get the correct 32 bit signed result.

With the "phone line in" and "handset microphone" samples separated, they are converted from integers into floating point numbers to allow the processor more range in its calculations. Normally, with 32 bit accumulators and only 16 bit input numbers there is no need to convert to floating point numbers. The 32 bit accumulators would allow enough room to calculate our filter equations. However, since the choice was made to use short IIR filter lengths, the coefficients were large when compared to the input values. Thus, large changes in the value of the accumulator while calculating the filtered data meant that we needed more then 32 bits to represent the intermediate values. The choice was made to use floating point math as the processor was able to perform all floating point operations in the same amount of time as the fixed point math operations.

Each channel is first sent through a notch filter. This removes the digital data signal preventing it from reaching the ears of the users. A copy of the "phone line in" sample is also filtered with a band-pass filter to remove the users voices. This filtered sample is then passed on to the digital demodulation section.

The physical sampling occurs at a rate of 8 kHz. To up-sample to the desired rate of 16 kHz, an intermediate sample is calculated using the interpolation equation:

$$x(n-1) = \frac{1}{3}x(n) + x(n-2) - \frac{1}{3}x(n-3) \tag{4.1}$$

As a result, for every sampling period, there are two samples to process. The demodulation routine is then used twice to sequentially process each sample separately.

After the data signal is run through the demodulator, it is compared to a threshold to give a resulting digital signal of either 0 or 1. The TMS320C31 program is expecting to see a serial digital data stream consisting of data bytes which are 8 bits long, have a start bit equal to 1 at the beginning, and a stop bit equal to 1 at the end. This UART data stream totalled 10 bits in length and was made to mimic the operation of a computer's UART.

Following the flow chart for the UART seen in Figure 4.8, the program checks for a value of 1. If a 1 occurs, the program marks this as a possible start bit. It sets a timer for half of a bit period, and then continues on. When the timer reaches zero, the program checks again to see if a value of 1 is read from the demodulator. If it is, then this data is considered a start bit and that 8 bits of data will follow. The timer is reset to one bit period and the program samples the demodulator data every time the timer reaches zero. After 8 bits are read, the program again checks to see if a 1 is received. If a one is received, the data byte is considered to be complete and is sent to the host PC through the COM port for display on the screen.

If an error occurs at either the start or stop bit, then the byte is considered to be garbage, and the Elf discards the data. It resets its timers and starts looking for another start bit. Errors that would be detected would include a

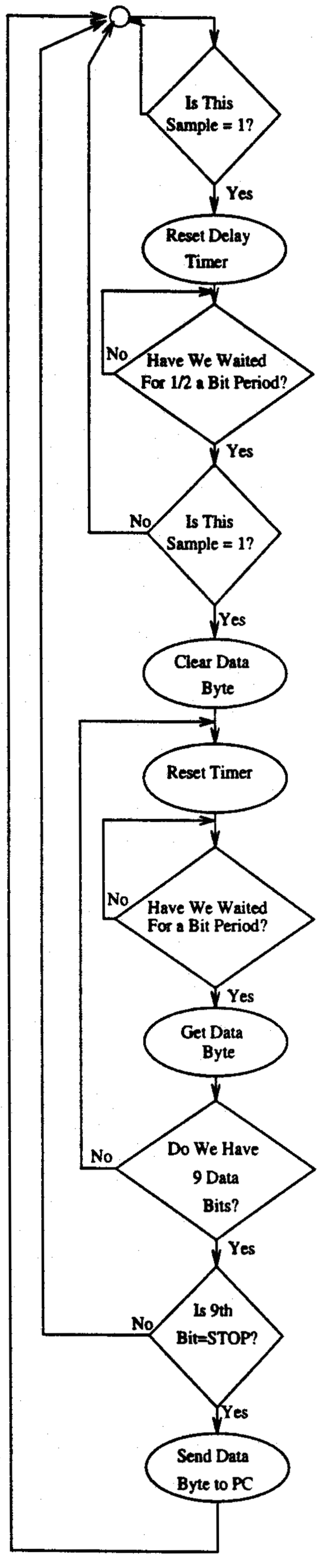


Figure 4.8 Flow Chart of the UART Process

start bit that was not equivalent to one, a stop bit that was not equivalent to a one, and a stop bit which was longer than one bit time.

Before returning to wait for another sample from the A-D, the sample from the microphone and the sample from the phone line are converted back into integers and then packed into a 32 bit word. This word is sent to the D-A for transmission. The original phone line sample minus the digital data is sent to the handset's ear speaker, while the voice from the handset's microphone is transmitted back down the phone line.

This cycle continually repeats itself until a **Hang Up** signal is received from the host PC. With this, the Elf board hangs up the phone line, shuts down the A-D/D-A, and resets for another phone call.

4.4 The Transmitter

The transmitter was also implemented in two parts. One part on the PC, and the other on a Elf card. The PC is responsible for generating a user display and interface for the user, while the Elf card is responsible for the filtering of the telephone line signals, generation of the digital modem, and for generating DTMF tones as necessary.

4.4.1 The Transmitting PC

A simplified flow diagram of the software implemented on the transmitting PC is given in Figure 4.9. The PC on the transmitter side provides the user with a text based interface to input his digital data. Upon start-up, the program clears the screen and provides a window-based text environment like that seen in Figure 4.10. The program then waits for the user to push the **Dial** button. When pushed, the user is prompted to enter the telephone number which is to be dialled.

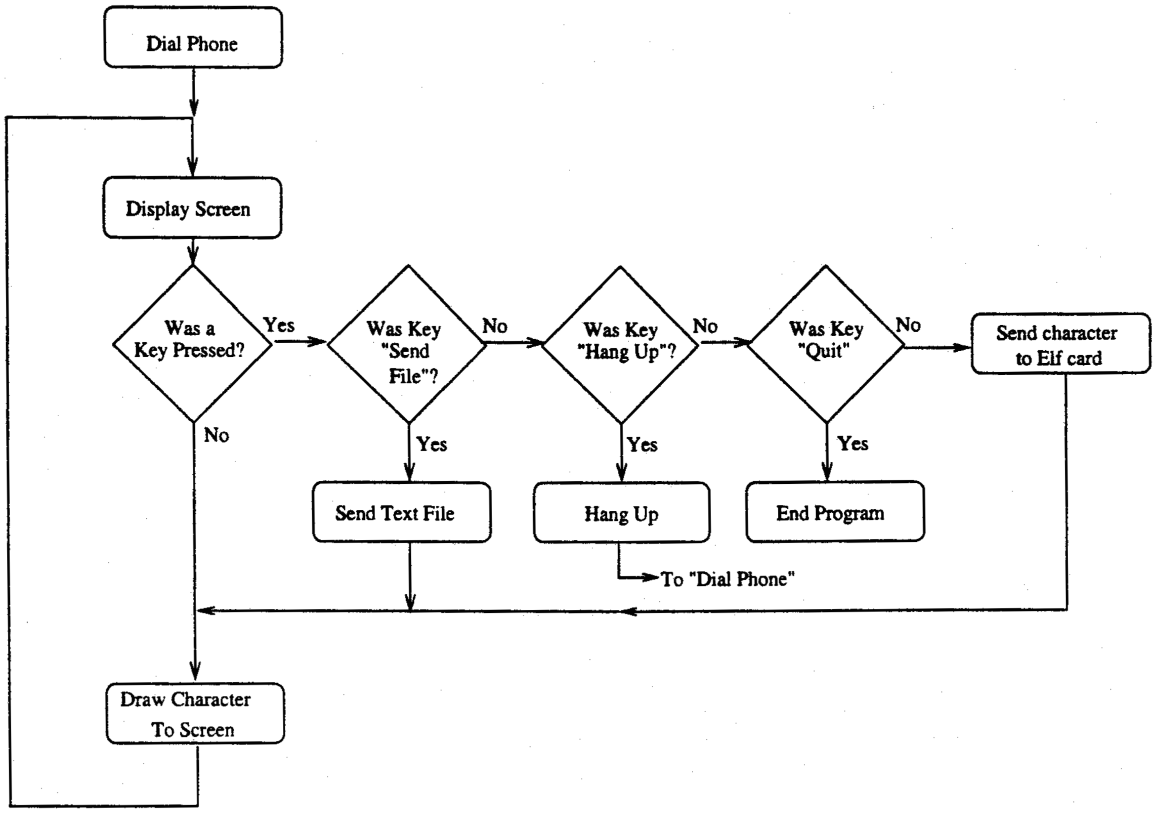


Figure 4.9 Software Flow Diagram for the Transmitting PC

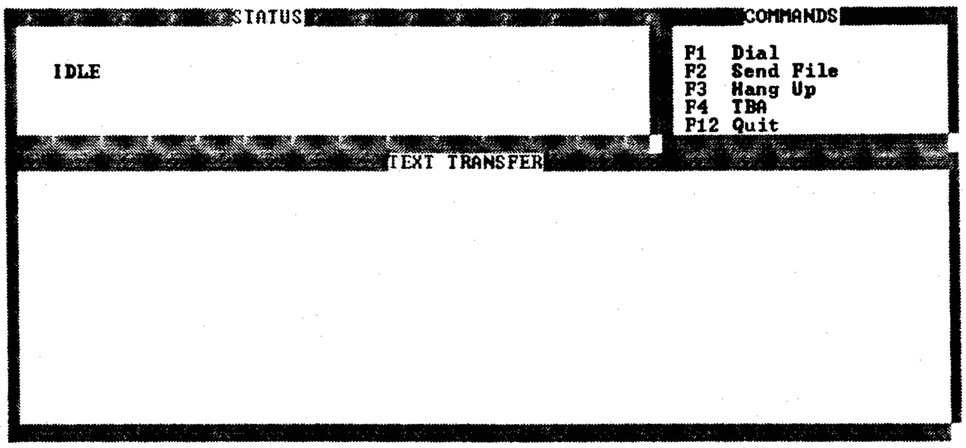

```
STATUS                                        COMMANDS
   IDLE                                       F1  Dial
                                              F2  Send File
                                              F3  Hang Up
                                              F4  TBA
                                              F12 Quit
                        TEXT TRANSFER
```

Figure 4.10 Transmitting PC's User Interface

Once the phone number is entered, the program uses the COM port to send the number to the Elf card. The character 0xF1 is sent first followed by the number of digits. The digits of the phone number are sent next, left to right, followed by the end character 0xF0.

Handshaking is used for all COM traffic to the Elf card. As seen in Figure 4.11, the host PC begins a transfer by raising its RTS (Ready to Send) flag. If the Elf responds by raising its CTS (Clear to Send) flag, then the PC may transmit its byte. Upon finishing, the PC lowers it's RTS flag and waits for the Elf to lower its CTS flag. If the Elf does not raise or lower its CTS flag within a certain amount of time, the PC alerts the user to a possible COM port error.

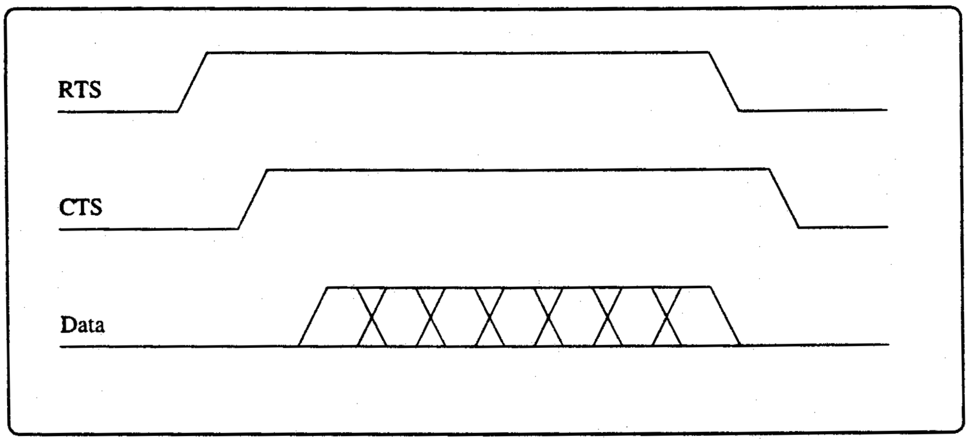


Figure 4.11 Handshaking between the PC and Elf card

Having dialled the phone, the program goes into a loop where it waits for input from the user on the keyboard. If a printable character is pushed, the key is passed to the Elf card for transmission.

Other options available to the user include **Hang Up**, **Send File**, and **Quit**.

The **Send File** redirects characters from a file to the Elf allowing you to pre-record typed messages to send to the other user.

The **Hang Up** option resets the Elf card, hangs up the phone, and begins again waiting for the user to enter a telephone number.

The **Quit** option sends the user back to the DOS prompt.

4.4.2 DTMF Generation In The Transmitting Elf Card

The flow diagram labelled Figure 4.12 gives a simplified view of the transmitting Elf board software sequence. After initialization, the Elf card waits for the PC to send it a phone number to dial. It checks the transmitted number for the correct start and end characters and also ensures that the numbers transmitted are between 0 and 9.

Once the phone number has been correctly received, the Elf card initializes its A-D/D-A and dials the phone number.

The dual tone multi-frequency (DTMF) tones necessary for dialling are generated using a simple sine addition. A single tone can be generated by using:

$$y[i] = \sin(\frac{2\pi f_d i}{f_s}) \tag{4.2}$$

where i is incremented by one each sample period, and f_s is the sampling frequency. Since DTMF is made up of two separate frequencies, the output will simply be an addition of two sinusoids $y_{out}[i] = y_1[i] + y_2[i]$. By combining two frequencies f_{d_1} and f_{d_2}, and then scaling to a useful amplitude, the DTMF signal can be generated. The DSP could generate each sinusoid using its built in sin function as the DSP would not be burdened by other calculations or modulation at that time.

The DTMF tones necessary for the transmission of each digit can be seen in Figure 4.2. Frequencies also exist for the four keys labelled A, B, C, and D. Although these keys are rarely found on telephone sets, frequencies for them are defined. The two tones necessary to generate the DTMF for each digit are the two which intersect on the number in question.

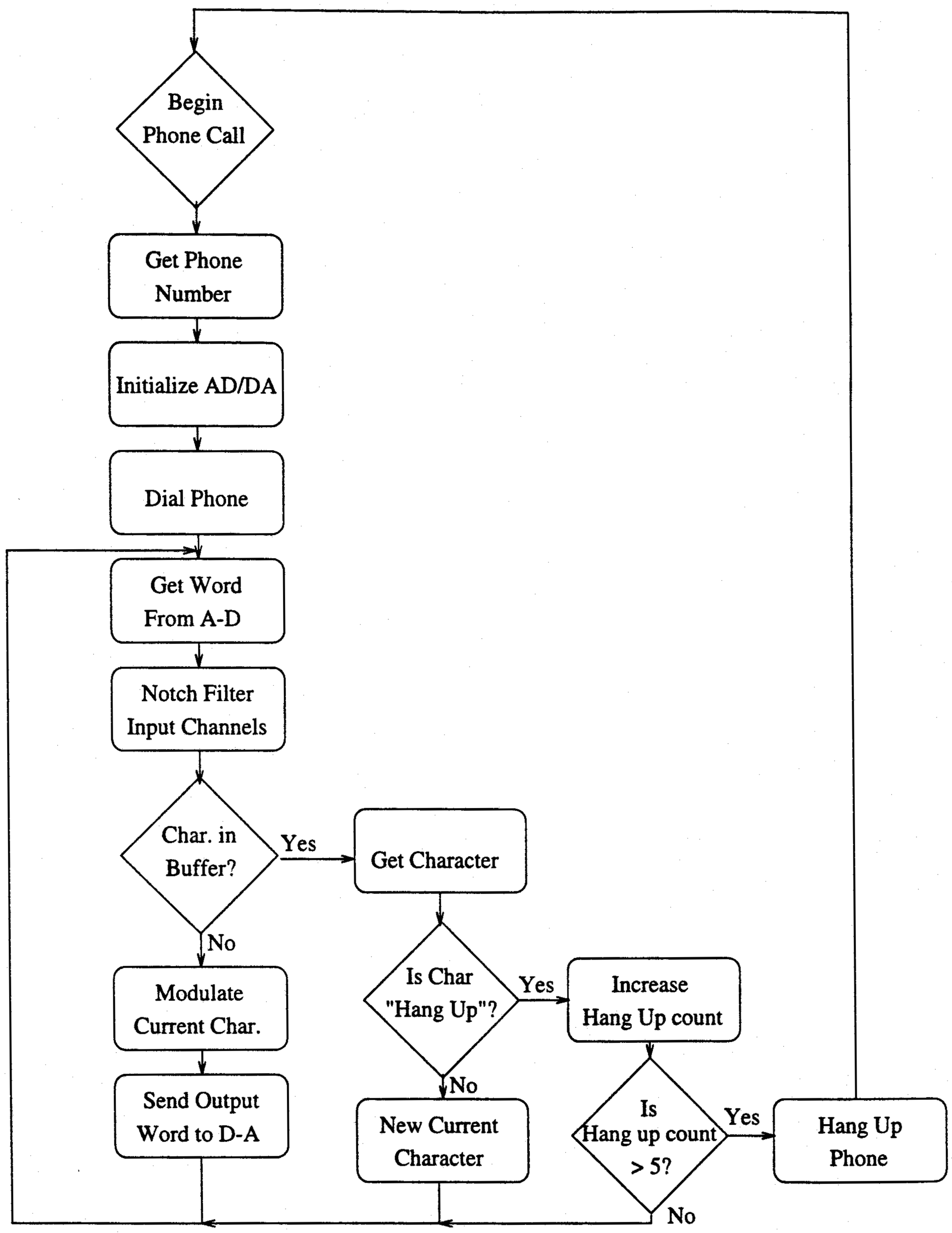


Figure 4.12 Software Flow Diagram for Transmitting Elf Card

Table 4.2 DTMF Frequency Allocation

		f_{d_1} (Hz)			
		1209	1336	1477	1633
f_{d_2} (Hz)	697	1	2	3	A
	770	4	5	6	B
	852	7	8	9	C
	941	*	0	#	D

The Elf program uses a look-up table to determine which frequencies are needed to generate the DTMF signal for the given input digit. As the sampling frequency of the Elf board and the frequencies for DTMF are known in advance, the look-up table becomes a series of constants, one for each digit of the phone dial pad.

After the DTMF calculation, the sample is passed to the A-D/D-A for transmission down the phone line. After sending the phone number, the Elf card enters the main modulation loop which is executed once every sampling period.

4.4.3 GMSK Data Transmission With The Elf Card

The DSP program monitors the COM port for any transmissions from the PC. Handshaking, as seen in Figure 4.11, is used to prevent overflow of the COM port buffer.

On receiving a byte of data from the PC, the program loads the byte into its transmission buffer. A timer is set up to shift each bit out of the buffer, once every bit period.

The modem signal is generated by taking the cosine of a running sum phase offset. By adding a constant phase to the sum at each sample period, a sine wave with a constant frequency is generated. The advantage to this technique is the sinusoidal output remains continuous and free from the high frequency components associated with discontinuous modulation techniques.

At each bit period, a new bit is shifted into the modulator. Depending on the value of the previous and current bit, four phase offset waveforms as shown in Figure 3.8 are used. If no change in the bits occurs (i.e., $0 \rightarrow 0$), then the phase offset remains constant and the resulting frequency also remains constant. If however, the bits are in transition (i.e., $1 \rightarrow 0$), then the phase offset changes from one to another gradually over the course of the bit period, and as the frequency follows the phase offset change, it also changes.

Look-up tables, generated using the formulas seen in equation 3.6, are used to generate all of the waveforms. The first step is to determine from the previous two data bits which transition the modulator is in and use the appropriate row in the first look-up table. The position within the first table is governed by the bit period timer. The table is configured to cover exactly one bit period. This table will return a value between -1 and 1 with each extreme representing one of the two carrier frequencies of a simple MSK modulation system. But instead of instantly changing from one frequency to the next, a more gradual system is used.

This value is next changed into a phase offset using the second look-up table. This phase offset is added to the current running offset sum and is wrapped if it exceeds 360^o. The final step is to use the third look-up table to take the cosine of the running sum. This answer is scaled and combined with the out-going voice.

After transmitting the voice and data down the phone line and the incoming voice to the ear, the program does one final check to see if the user wants to hang up the phone. When the user wants to hang up, the PC sends five "0xF2" characters to the Elf card. If the Elf card receives these characters, the Elf card hangs up the phone, disables the A-D/D-A, and resets itself to wait for another telephone number.

5. Performance Results and Discussion

5.1 Bit Error Rate (BER) with Added Gaussian Noise

Bit Error Rate testing was done using two subscriber telephone lines each approximately 400 meters in length. Each line is connected to the local exchange switch and each is capable of supporting normal telephone operations. As seen in Figure 5.1, a Gaussian noise generator was used to add noise directly to the telephone wire. The line driver provided line balancing and impedance matching in order to minimize the effects of echoes resulting from discontinuities of the line impedance. A differential line receiver converted the balanced signal back to unbalanced where it could be fed into the spectrum analyzer. While exact values of N_o and E_b could not be measured by this method, their ratio E_b/N_o could be determined. Since the ratio E_b/N_o becomes a subtraction of the logs of each, the ratio can be measured on the spectrum analyzer. By using the spectrum analyzer to measure the noise floor before the modem is turned on, and by measuring the combined signal after, the ratio can be measured.

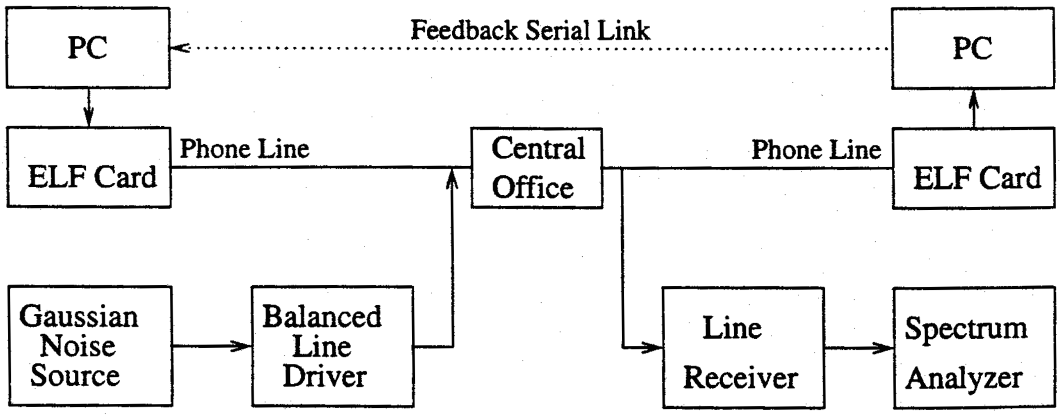


Figure 5.1 BER Test Configuration

In order to compare results from this system with those from theory, the ratio of E_b/N_o was used. By using the energy per bit sent as a reference, systems with different modulation schemes can be compared. For a given error rate, the ratio can be used to calculate the necessary signal power (E_b) given the channel noise power spectral density (N_o). The Gaussian noise generator was used to add noise to the line. The noise was considered to have a uniform spectral density (white) in the frequency band of the pass band. The transmitter generated a GMSK signal of constant amplitude. Therefore, as the noise amplitude was changed, the error rate was measured, and the P_e vs. E_b/N_o[2] curve could be generated.

In evaluating the performance of the experimental system, non-coherent FSK was used as a reference. The approximation to GMSK which is used by the transmitter has an expected bit error rate pattern similar to non-coherent FSK [2].

An experiment would begin after the transmitting Elf card established a data connection with the receiving Elf card. DTMF tones, generated by the transmitting Elf card was used to send signalling information to the telephone company's central office, while a ring detector on the receiving Elf card alerted the program to an incoming call. After transmission data lock-up, the transmitting PC and Elf card began to send a series of random characters. The characters were uniformly distributed between 0 and 255. The generation of the random character was done by the PC while the GMSK approximation modulation was done by the Elf card. The Elf card also added a set level of Gaussian noise. The generated noise signal and the modulated signal were then combined and sent onto the telephone network. The receiving Elf card demodulated the incoming signal. The demodulated characters were sent to the receiving PC through a series of data transfers. The receiving PC then completed the loop by passing the received character back the transmitting PC through a connecting serial link. The transmitting

PC was able to compare the received character with the original transmitted character and determine if an error occurred. The generated results included number of bits sent and number of errors. When combined with the externally measured E_b/N_o, the P_e vs. E_b/N_o curve could be determined.

Figure 5.2 shows the results of 17 days of testing. The tabulation of the raw data from the tests can be found in Appendix C. For reference, the probability of error curve for non-coherent FSK is shown.

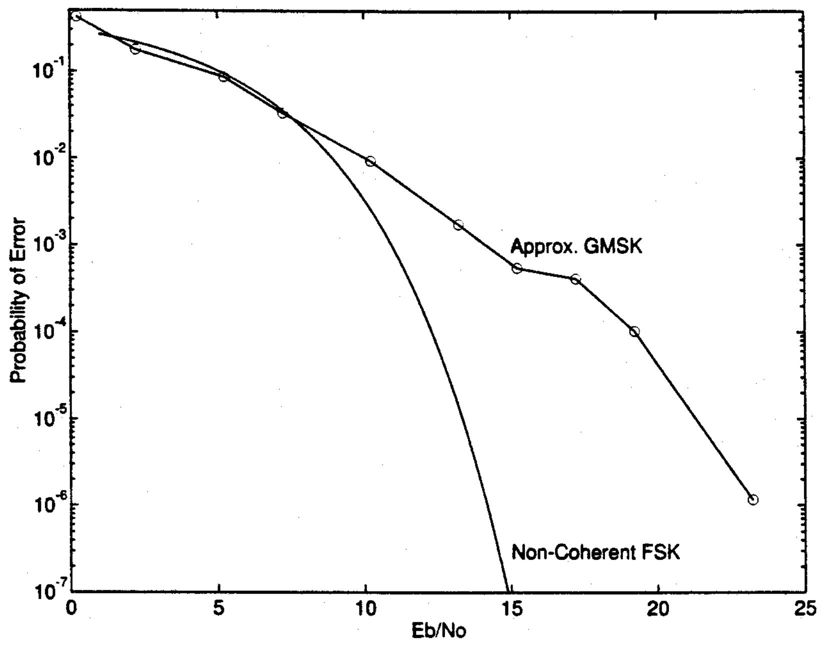


Figure 5.2 BER Test Results

Some sources of implementation loss in the system include the non-coherent demodulator, and non-equalization of the data channel. The probability of error curve for the GMSK approximation is compared against that of non-coherently demodulated FSK. However, the GMSK approximation may not be particularly suited for non-coherent demodulation due to the introduction of phase non-lineararity. This situation is only made worst by the introduction of a non-ideal transmission channel. It may have then been optimistic to expect the system to meet ideal standards.

The largest effects on the probability of error curve occurred as the noise signal was reduced. This could most likely be explained as a LPCM problem. As was discussed earlier, the Elf system used a 16 bit LPCM codec to capture samples from the telephone line. As was also explained earlier, the LPCM signal to noise ratio decreases nearly linearly as the input amplitude is reduced. Thus, for small levels of added noise in the GMSK approximation system, a significant part of the overall noise in the system would be added by the LPCM codec, and not by the gaussian noise generator.

5.2 BER with Gaussian Noise Added to the Microphone

In order to determine how the system will operate with large voice inputs, the system was again tested using the Gaussian noise generator to add noise at the microphone input. The rms voltage of the noise was varied up to a maximum of 1.5V. At levels above this, the peak noise voltage frequently exceeds the $\pm 2.5V$ range of the input quantizer.

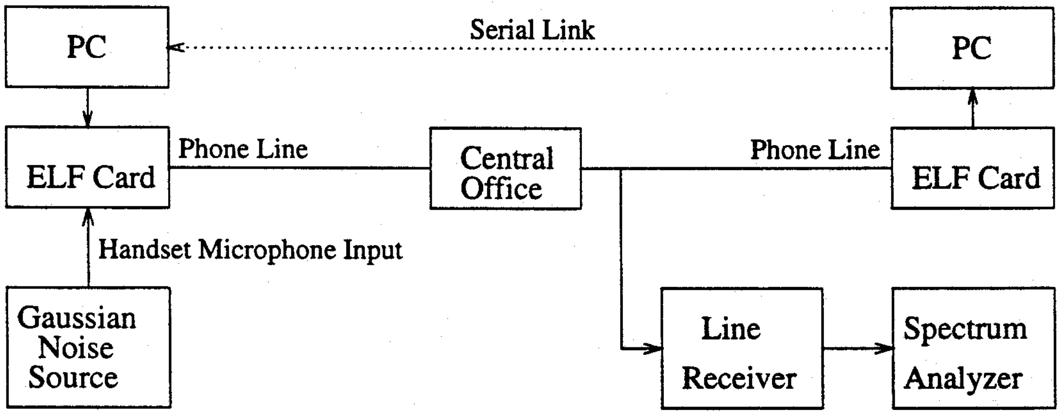


Figure 5.3 BER Test with Microphone Noise

Figure 5.3 shows the set up used to test for microphone noise tolerance. The generated noise was gradually increased from zero to 1.5V rms, a point where saturation of the A-D/D-A becomes extreme. At this case, the audio channel of the system was unusable. Despite the excessive noise in the audio channel, no effects were measured in the data channel. The BER stayed roughly constant regardless of the noise power in the noise channel. The input bandstop filter prevented the noise from entering the frequency band occupied by the data channel.

5.3 Subjective Sound Quality Assessment

Because the quality of received voice is not easy to quantify, it was decided to use subjective evaluation. Both male and female voices were filtered through the system. The voices were both high and low pitched. Different speaking accents were used to demonstrate the systems ability to handle speakers of different linquistic backgrounds.

A panel of eight unbiased observers were asked to listen to the many voices and rate the overall system in terms of its ability to generate good quality sound. The observers were asked to use a normal telephone connection as their base of comparison in order to draw their conclusions.

The results from the observations were all about the same. The conclusions drawn from the panel were that while the voice quality was not as good as the telephone company's original signal, the voice quality was good enough to be used for short periods of time in a normal conversation. They agreed that the voice was acceptable for time periods up to ten seconds or so. They also concluded that any time period longer than that would make the listener strain too hard to understand the total conversation.

6. Conclusions

Research was performed on simultaneous voice and inband data over telephone wires. The system was constructed using a TMS320C31 ELF digital signal processing card placed in an Intel compatible personal computer. The system delivered a 600 b/s digital data service while maintaining a fully working voice channel.

Bit error rate tests were performed with varying amounts of Gaussian noise added. As expected, the BER increased as more noise was injected onto the phone line. Results compared the performance of approximate GMSK transmission against the ideal response of non-coherent FSK transmission. While the system appears to have close to a 8 dB implementation loss at high E_b/N_o values, the results are still very encouraging. In systems where the designer has control over the entire system, an implementation loss of 3-6dB over the ideal is common.

Part of the implementation loss can be blamed on the codecs located in the central office. They have a less than ideal frequency response and are not optimized for data traffic. The sampling and reconstruction done by the codecs add noise and can create a none constant amplitude envelope on signals with frequencies near 4 kHz. Modern modems use a series of frequency tests in order to determine several characteristics of the line, including the frequency response. With the help of digital signal processing, the modems are able to compensate for the irregularities in the frequency response of the line. As no such testing was performed by the ELF DSP software, similar compensation was not done.

The non-ideal nature of the delay and multiply demodulator has also added a large part of the implementation loss. When an error is made in the first or last bit, the timing recovery information for the entire word may be lost. This error may cause the loss of 8 bits of data. As a result, the error rate of the system is heavily swayed by the correct detection of the start and stop bits.

The bit error rate was also examined when noise was added to the handset's microphone. This noise emulated someone speaking into the microphone. Depending on the noise amplitude, voice from a whisper to a scream was emulated. The results indicated that the system had an exceptional immunity to microphone noise. The system performed so well that no bit error rate difference could be detected between the full load noise case and the no-noise case. The filters on the input from the microphone filtered the noise signal well and let practically no noise escape into the demodulator circuit.

If we impose an acceptance error rate of 1 error in 1000000 ($1 * 10^{-6}$), the E_b/No ratio must exceed 23dB in order to meet the requirement. Keeping this in mind, the telephone companies have a quantizing noise ratio ranging between 32 and 38 dB. Using this number and the modem signal power, one can conclude that the system will operate within acceptable error rates on virtually all telephone connections.

6.1 Potential System Improvements

The largest factor governing the quality of the digital receiver was the processing power of the TMS320C31 ELF board. Lack of sufficient processing power to implement a fully featured digital receiver meant that certain corners had to be cut in order to get the system to operate in real time. As this project neared completion, a new ELF TMS320C31 board was introduced to the market with twice the clock rate of the board used in this project. With the faster clock rate, several improvements to the demodulator could be accomplished.

By the use of the delay and multiply digital receiver, the receiver could implement the system in real time. It was, however, quite susceptible to line noise. By only sampling at 8kHz and interpolating up to 16kHz, it became apparent that each sample was very sensitive in the demodulation process. The 2.4 kHz GMSK carrier meant that each cycle would only be sampled 3 to 4 times. Adding noise to these few samples will make the interpolation perform very poorly. The use of a Fast Fourier transform method of demodulation would provide better noise immunity. The FFT would average the noise spike out allowing for a cleaner signal to be passed onto the decoder.

The addition of an automatic gain control may have also helped the system. While the ideal telephone circuit has a flat frequency response in the voice band, this is rarely true for real circuits . The different amount of attenuation between the two extremes of the modulating signal can have a negative effect on the demodulator. The addition of a hard-limiter would help eliminate the amplitude modulation effects on the incoming signal.

A more professional looking front end screen display could be developed. The text based system used was for testing and small demonstration purposes only. The system could be run in a Microsoft Windows environment and use one of the many commercial desktop conferencing packages already available. The system would only have to imitate the responses of a modem in order to work seamlessly with the desktop software.

6.2 Future Work

In order to have proper desktop conferencing, the system will have to be made bi-directional. While this would not be a problem for a higher speed digital signal processor, one would have to develop error correcting codes and handshaking protocols.

The modulation method used could be changed to one that has a higher bit rate to bandwidth ratio. By using a method that gets 4 bps/Hz [1], a very usable data rate of 2400b/s could be obtained. As expected, as the bps/Hz increase, so must the processing power necessary to demodulate the signal. It is unknown what processor speed would be necessary to implement such a design.

[1] A modulation scheme known as 256 level QAM has a spectral efficiency of $\eta = 4bits/sec/Hz$. The commercial modems running at speeds of 28.8kb/s employ techniques similar to this.

References

[1] John Bellamy. Digital Telephony. New York, NY:John Wiley & Sons, Inc.,1991.

[2] Leon W. Couch II. Digital and Analog Communication Systems. New York, NY:Macmillian Publishing Company, 1993.

[3] David E. Dodds. EE816.3 Telephony Class Notes. University of Saskatchewan, January 1995.

[4] Phil Evans, Al Lovrich. Implementation of an FSK Modem Using the TMS320C17. Texas Instruments - Semiconductor Group, Ottawa, ONT, 1990.

[5] Dr. Kamilo Feher. Advanced Digital Communications. Englewood Cliffs, NJ:Prentice-Hall, 1987.

[6] Fred Harris. Filter Structures Using Recursive Allpass Filters. In Procedings *Internation Symposium on Signal Processing and I.T. Applications* Gold Coast, Australia, August 16, 1992.

[7] Fred Harris. Multirate Digital Filter And Applications Course Notes. San Diego State University, 1997.

[8] Dale A. Heatherington. A 56 kilobaud RF Modem. In Proceedings *ARRL 6th Computer Networking Conference* August 29, 1987.

[9] John A. Kuecken. Talking Computers and Telecommunications. New York, NY:Van Nostrand Reinhold Company, 1983.

[10] Leslie Lamport. Latex - A Documentation Preparation System. Readings, Massachusetts:Addison-Wesley Publishing, 1994.

[11] Jeffery D. Laster. Robust GMSK Demodulation Using Demodulation Diversity and BER Estimation. Virginia Polytechnic, Blacksburg, Virgina, March 1997.

[12] Panos Papamichalis. Digital Signal Processing Applications with the TMS320 family Volume 2. Texas Instruments, 1990.

[13] John G. Proakis, Dimitris G. Manolakis. Digital Signal Processing, Principles, Algorithms, and Applications. New York, NY:Macmillian Publishing Company, 1992.

[14] John G. Proakis. Digital Communication. New York, NY:McGraw-Hill, Inc, 1989.

[15] Mischa Schwartz. Information Transmission, Modulation, and Noise. New York, NY:McGraw-Hill, Inc, 1980.

[16] Dean Zurburg. Simultaneous Voice and In-band Data. In Proceedings *IEEE Wescanex conference* Winnipeg, Manitoba, May 16, 1995.

A. Calculation of Phase Delay Filter Coefficients

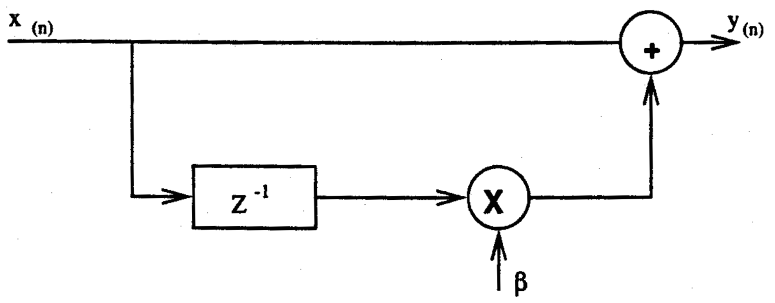


Figure A.1 One Zero Phase Delay Filter

The response of the phase delay filter is given by:

$$y(n) = x(n) + \beta x(n-1) \tag{A.1}$$

The corresponding transfer function in the frequency domain is:

$$Y(z) = X(z) + \beta z^{-1} X(z) \tag{A.2}$$

The transfer function $F(z)$ of the filter is then given by:

$$F(z) = \frac{Y(z)}{X(z)} = (1 + \beta z^{-1}) \tag{A.3}$$

Evaluating $F(z)$ at $z = e^{j\omega}$ to obtain the frequency response leaves:

$$F'(\omega) = F(e^{j\omega}) = 1 + \beta e^{-j\omega} \tag{A.4}$$

By dividing Eq. A.4 into its real $R(\omega)$ and imaginary $I(\omega)$ parts, a polar expression $A(\omega)e^{j\phi(\omega)}$ is obtained with $A(\omega)$, $\phi(\omega)$, $R(\omega)$, and $I(\omega)$ all being real functions of ω. Solving for $A(\omega)$ and $\phi(\omega)$, we get:

$$A(\omega) = \mid F'(\omega) \mid = [R(\omega)^2 + I(\omega)^2]^{\frac{1}{2}} \tag{A.5}$$

and

$$\phi(\omega) = \arctan(I(\omega)/R(\omega)) \tag{A.6}$$

Given

$$e^{-j\omega} = \cos\omega - j\sin\omega \tag{A.7}$$

we can substitute Eq. A.7 into Eq. A.4 to obtain

$$F'(\omega) = 1 + \beta \cos \omega - j\beta \sin \omega \tag{A.8}$$

Combining Eq. A.6 and Eq. A.8 results in

$$\phi(\omega) = (\arctan \frac{-\beta \sin \omega}{1 + \beta \cos \omega}) \tag{A.9}$$

The purpose of this filter is to introduce a precise group delay τ to the received signal where τ is defined as

$$\tau = \frac{-d\phi(\omega)}{d\omega} = \text{group delay} \tag{A.10}$$

Substituting Eq. A.9 into Eq.A.10 yields

$$\tau = \frac{\beta(\beta + \cos \omega)}{1 + 2\beta \cos \omega + \beta^2} \tag{A.11}$$

By solving Eq. A.11 for β with ω set to ω_c, a value for β is found which will introduce a $-\pi/2$ phase delay with respect to the carrier frequency.

By combining Eq. A.5 and Eq. A.8 we obtain the amplitude response $A(\omega)$ of the filter

$$A(\omega) = \sqrt{(1 + \beta \cos \omega)^2 + (\beta \sin \omega)^2} \tag{A.12}$$

$$= \sqrt{1 + 2\beta \cos \omega + \beta^2 \cos^2 \omega + \beta^2 \sin^2 \omega} \tag{A.13}$$

$$= \sqrt{1 + 2\beta \cos \omega + \beta^2} \tag{A.14}$$

If we plot the frequency and phase responses of eq. A.14, we get the responses shown in Figures A.2 and A.3. The coefficients are chosen so the phase response at the carrier frequency is equal to -90^o or $-$ *pi*/2 radians. By scaling the two IIR coefficients (α and β), a DC gain of 1 can be achieved.

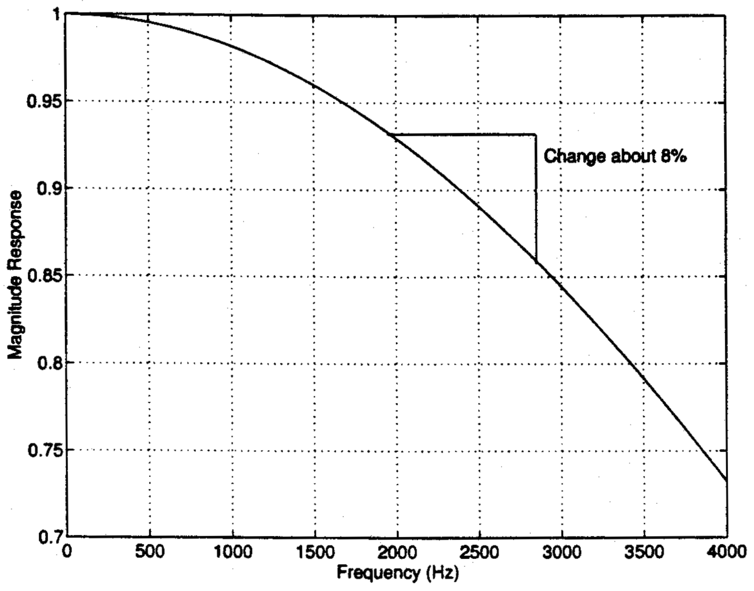


Figure A.2 Magnitude Frequency Response of Single Zero Delay Filter

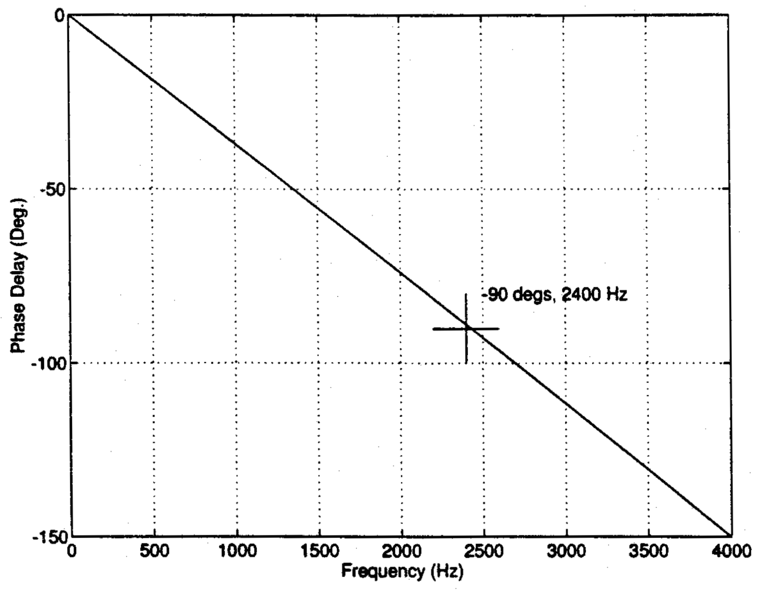


Figure A.3 Phase Frequency Response of Single Zero Delay Filter

For small ω, the Eq. A.14 simplifies to

$$A(\omega) \approx \sqrt{1 + 2\beta + \beta^2} \tag{A.15}$$

For coefficient scaling, a gain product is added.

$$A(\omega) \approx G * \sqrt{1 + 2\beta + \beta^2} \tag{A.16}$$

By proper coefficient scaling, a gain of approximately one for the effective frequency range of the filter can be achieved. Figure A.2 shows that in our signal transmission band (1950 Hz - 2850 Hz), the gain of the single zero delay filter changes by less than 7%, (or $0.6dB$). This gain difference in the passband will introduce amplitude modulation effects into the system.

B. Quadratic Equation for Interpolation

The quadratic interpolation formula is used to generate a second order approximation for interpolation. It has the advantage of being quick to calculate, saving previous DSP processing time.

To solve for the interpolation formula, we begin with the basic quadratic equation $x = at^2 + bt + c$. A series of simultaneous equations can then be used to represent the system. First, let x_0, x_1, x_2, and x_3 represent the output values of the system. The value x_0 would represent the output at time$=0$ and x_3 at time$=3$. As there will be four unknowns in the equation (a, b, c and x_1), we will need four equations. The first four equations generated from the basic quadratic equation with time equal to 0, 1, 2, and 3, can be written as follows

$$x_3 = c \tag{B.1}$$

$$x_2 = a + b + c \tag{B.2}$$

$$x_1 = 4a + 2b + c \tag{B.3}$$

$$x_0 = 9a + 3b + c \tag{B.4}$$

Solving for a:

$$x_2 = a + b + x_3 \tag{B.5}$$

$$a = x_2 - b - x_3 \tag{B.6}$$

Solving for b:

$$x_0 = 9(x_2 - b - x_3) + 3b + c \tag{B.7}$$

$$x_0 = 9x_2 - 9b - 9x_3 + 3b + x_3 \tag{B.8}$$

$$b = -\frac{x_0}{6} + \frac{9x_2}{6} - \frac{8x_3}{6} \tag{B.9}$$

Thus, solving for x_1:

$$x_1 = \frac{4x_0}{6} - \frac{12x_2}{6} + \frac{8x_3}{6} - \frac{2x_0}{6} + \frac{18x_2}{6} - \frac{16x_3}{6} + x_3 \qquad \text{(B.10)}$$

With simplification and substitution of $x(n-k) = x_k$, x_1 equates to

$$x(n-1) = \frac{1}{3}x(n) + x(n-2) - \frac{1}{3}x(n-3) \qquad \text{(B.11)}$$

or in the Z-transform domain:

$$z^{-1}X(z) = \frac{1}{3}X(z) + z^{-2}X(z) - \frac{1}{3}z^{-3}X(z) \qquad \text{(B.12)}$$

C. Tabulated Results from BER Tests

Table C.1 Test Results - August 12 to 27, 1996

E_b/N_o	Bits Sent	Errors	BER
15	$1.522 * 10^5$	63893	$4.2 * 10^{-1}$
17	$2.245 * 10^5$	40200	$1.79 * 10^{-1}$
25	$1.058 * 10^6$	9934	$9.39 * 10^{-3}$
32	$5.586 * 10^5$	307	$5.50 * 10^{-4}$
34	$9.804 * 10^5$	150	$1.53 * 10^{-4}$
38	$2.059 * 10^7$	20	$9.71 * 10^{-7}$
38	$2.22 * 10^6$	44	$1.98 * 10^{-5}$

Table C.2 Test Results - August 28 to 30, 1996

E_b/N_o	Bits Sent	Errors	BER
20	$2.903 * 10^5$	25114	$8.65 * 10^{-2}$
22	$2.645 * 10^5$	8765	$3.31 * 10^{-2}$
25	$2.258 * 10^5$	2078	$9.20 * 10^{-3}$
28	$2.072 * 10^6$	3600	$1.74 * 10^{-3}$
30	$4.889 * 10^5$	144	$2.95 * 10^{-4}$
30	$1.407 * 10^6$	895	$6.36 * 10^{-4}$
32	$4.644 * 10^5$	115	$2.48 * 10^{-4}$
34	$2.846 * 10^6$	245	$8.61 * 10^{-5}$
38	$2.077 * 10^7$	32	$1.54 * 10^{-6}$

Table C.3 Combined Test Results

E_b/N_o	Bits Sent	Errors	BER
15	$1.522 * 10^5$	63893	$4.2 * 10^{-1}$
17	$2.245 * 10^5$	40200	$1.79 * 10^{-1}$
20	$2.903 * 10^5$	25114	$8.65 * 10^{-2}$
22	$2.645 * 10^5$	8765	$3.31 * 10^{-2}$
25	$1.284 * 10^6$	12012	$9.36 * 10^{-3}$
28	$2.072 * 10^6$	3600	$1.74 * 10^{-3}$
30	$1.896 * 10^6$	1039	$5.48 * 10^{-4}$
32	$1.023 * 10^6$	422	$4.13 * 10^{-4}$
34	$3.826 * 10^6$	395	$1.03 * 10^{-4}$
38	$6.19 * 10^7$	72	$1.16 * 10^{-6}$

D. ELF Board GMSK Demod C Source Code

```
/*
 *  ======== msk_rx.c ========
Receive data from the serial port using sinusoidal
preshaped MSK, or GMSK with LPF bandwidth set to about
0.5 .

Read and write to the A/D,D/A without the use
of the spox SIG_IO functions

March 27/96
 */

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



#define OFF                     1
#define ON                      0
#define SCALE_FACTOR            2
#define SCALE_FACTOR2           1
#define BACKGROUND
/* #define FOREGROUND */
#define FREQ1                             /* fc = 2400Hz  */
#define FREQ_COUNT             28
#define HALF_FREQ_COUNT        15
#define HYSTOR                 15
#define TMS320
```

```
/*prototypes */
void  INITIALIZE_ANALOG(void);
float FILTERIIR(float *, float *, float, int);
void  PHONE_HOOK(int);
void  DELAY(void);



/*  ======== smain ========*/

void smain(argc, argv)
  int argc;
  char *argv[];
{

    register int       status;
    register int       i ;
    int                temp, mic, temp2, temp5;
    float              temp3, temp4;
    volatile unsigned  *serial_data_rx,
       *serial_data_tx; /* addresses of ADAC */
    volatile unsigned  *uart_data;      /* address of PC uart data reg      */
    volatile unsigned  *uart_flag;      /* address of PC uart flag reg      */
    volatile unsigned  *uart_modem;     /* address of PC uart modem reg     */
    volatile unsigned  *uart_interrupt; /* address of PC uart int. reg      */
    volatile unsigned  *uart_status;    /* address of PC uart status reg    */
    volatile unsigned  *c31_control;    /* address of C31 control register */
    volatile unsigned  *c31_data;       /* address of C31 data exch. reg    */
    float              outputL, outputR;/* Voice output                     */
    float              inL, inR;        /* voice input                      */
    volatile unsigned  *flag_ptr;       /* ADAC serial flag register        */
```

```
float               delay1[32],      /* delay elements for filtering    */
   delay2[32],
   delay3[32],
   delay4[32];       /* filter delay elements          */
float               sine_lookup[360];/* sine table                       */
unsigned            a,b;             /* temp variables                   */
int                 c,d;
float               e,f1,f2;
unsigned         adaptive = 0;
int                 parity;          /* parity bit                       */
int                 count1;          /* determine which bit we are on    */
int                 count2;          /* determine which sample we are on*/
int                 data_byte;       /* byte we are currently receiving */
int         data_bit;
int                 delayed_modem_out[FREQ_COUNT+1];
float               phase_lookup[360] ;      /* voltage to phase look-up */
float               past_in[3] = {0, 0, 0}; /* past input samples        */
float               modem_in[4] = {0,0,0,0};/* used to convert modem     */
   /* signal from 8kHz to 16kHz*/
float               modem_multiply;          /*                          */
float               modem_out[2] = {0, 0};   /*                          */
float         dc;               /*                                */
int                 min_modem;       /* current minimum of modem_out     */
int                 max_modem;       /* current maximum of modem_out     */

float               coeff1 = 3.6055; /* coefficents for one-pole phase   */
float               coeff2 = 1 ;     /* delay filter needed for          */
float               coeff3 = 4.6055 ;/* GMSK demodulation                */

#ifdef FREQ1
```

```
   float              bpf_coeff[] = { 0.00189579,
 1.19802,   0.981091,  1,  1.64423,
 1.11188,   0.931084,  1,  1.40706,
 0.91496,   0.854986,  1,  1.33253,
 0.588231,  0.798492,  1, -0.899243,
 0.253182,  0.836531,  1, -0.385451,
 0.0439949, 0.916374,  1, -0.248527,
-0.0424509, 0.976320, -1,  0.00000
       };
/* BW = 850Hz Center = 2.4k */


    float              notch_coeff[] = { 0.106254,
 1.43445,   0.725281, 1,  1.36394,
 1.44868,   0.908635, 1,  1.30033,
 1.45004,   0.978395, 1,  1.11387,
-0.645596,  0.585120, 1,  0.725765,
 0.275408, -0.241055, 1,  0.262529,
-0.428033,  0.866431, 1, -0.144518,
-0.340391,  0.968871, 1, -0.0303619
};     /* BW = 1000Hz Center = 2.4K */


/*    float              lpf_coeff[] = { 0.00868928,
-1.11745, 0.324257, 1, -1.88692,
-1.8478 , 0.93397 , 1, -1.86194,
-1.76684, 0.860033, 1, -1.78515,
-1.60991, 0.726727, 1, -1.51588,
-1.37094, 0.530293, 1, 0.147904
}; */
```

```
/* LPF Fc = 750, -40dB @ 900, Fs=16k */

/*     float lpf_coeff[] = { 0.00898657,
     -1.62385, 0.662665, 1, -1.97441,
     -1.95419, 0.970979, 1, -1.96448,
     -1.76771, 0.795468, 1, -1.92131,
     -1.86212, 0.882019, 1, -1.44015
     };*/
     /* LPF Fc = 300, -35dB @ 350, Fs=16k */

/*      float lpf_coeff[] = { 0.0637,
      -1.54679, 0.603607, 1, -1.96763,
      -1.94602, 0.968630, 1, -1.95510,
      -1.86580, 0.892899, 1, -1.90082,
      -1.73084, 0.769770, 1, -1.31621
      };*/
      /* LPF Fc = 350, -35dB @ 450, Fs = 16k */
float lpf_coeff[] = { 0.022632,
-1.893066, 0.9130249, 1, -1.935356,
-1.716187, 0.7408142, 1, -1.650861
};

#endif

      /* address of ADAC serial registers*/
    flag_ptr       = (volatile unsigned *) 0x808040;
    serial_data_rx = (volatile unsigned *) 0x80804C;
    serial_data_tx = (volatile unsigned *) 0x808048;
    uart_data      = (volatile unsigned *) 0xA00000;
    uart_flag      = (volatile unsigned *) 0xA00005;
    uart_modem     = (volatile unsigned *) 0xA00004;
    uart_interrupt = (volatile unsigned *) 0xA00001;
    uart_status    = (volatile unsigned *) 0xA00006;
    c31_control    = (volatile unsigned *) 0x900002;
    c31_data       = (volatile unsigned *) 0x900000;

    asm("   AND 0, IE ");       /* turn off serial ports int     */
    asm("   LDI 0, IF ");          /* clear interrupts              */
    asm("   OR 2000h, ST ");       /* turn on interrupts            */
    *uart_interrupt = 0x0;               /* com port  no interrupt        */
    UART_init(12, 0x0003, 0);            /* 9600 baud, 1 stop bit         */



BEGIN:
    temp           = *serial_data_rx ;   /* clear sample from ADAC        */
    *uart_modem    = 1;                  /* set CTS low                   */

    for (i=0; i<(FREQ_COUNT+1); i++)     /* zero elements of              */
{                                   /* delayed_modem_out          */
delayed_modem_out[i] = 0;
}

    for (i=0; i<32; i++)                 /* zero filter delay elements    */
{
delay1[i] = 0;
delay2[i] = 0;
delay3[i] = 0;
delay4[i] = 0;
}


#ifdef BACKGROUND
    a = 0;
    while (!a)
      {
      a = (*uart_status & 0x010); /* check CTS bit                     */
      }                           /* wait for host PC to send          */
  /* a byte indicating to pick up phone   */
    *uart_modem = 3;              /* raise RTS                         */

    b = 0;
    while(!b)
      {
      b = *uart_flag & 0x01 ;     /* check data ready flag             */
      }                           /* wait for data in buffer           */
    inL = *uart_data;             /* clear buffer, lower RTS and       */
    *uart_modem = 1;              /* get ready to start                */
#endif

    PHONE_HOOK(OFF);              /* take phone off-hook               */
    for (i=0; i<10; i++)
{
DELAY();
}

    INITIALIZE_ANALOG();                   /* begin ADAC processing
    DELAY();
    DELAY();
    DELAY();



    asm("         PUSH R0  ");
    asm("         PUSH R1  ");
    asm("         PUSH AR0 ");
    asm("         PUSH AR1 ");
    asm("         TRAP 4   ");
    asm("         POP  AR1 ");
    asm("         POP  AR0 ");
    asm("         POP  R1  ");
    asm("         POP  R0  ");

    i  = 0;
    e  = 0;
    c  = 0;
    dc = 200;
    f1 = e + HYSTOR ;
    f2 = e - HYSTOR ;


    while(1)
{
while( (*flag_ptr & 0x01) == 0 );
      /* wait for data on serial port coming from A/D */
temp = *serial_data_rx ;  /* get ADAC word from serial port        */
mic  = temp;
temp = temp >> 16;        /* upper 16 bits is left channel         */
inL  = (float) temp ;
mic  = mic << 16;         /* get rid of top 16 bits                */
mic  = mic >> 16;           /* sign extend                              */
inR  = (float) mic ;        /* lower 16 bits is right channel           */

     /* preform IIR filtering on the two channels */
outputR      = FILTERIIR(notch_coeff, delay1, inL, 6);
outputL      = FILTERIIR(notch_coeff, delay4, inR, 6);
/* outputL = inR ;  */

modem_in[3] = modem_in[1]; /* oldest sample                             */
modem_in[2] = modem_in[0]; /* next oldest                               */
modem_in[0] = FILTERIIR(bpf_coeff, delay2, inL, 6) - dc;
modem_in[1] = (modem_in[0] * 0.3333333) +
       modem_in[2] -
      (modem_in[3] * 0.3333333) ;
   /* quadratric interpulation                 */

if (adaptive)
   {
   temp5 = ( (int) modem_out[0] ) >> 2;

   if (temp5 > max_modem)   { max_modem = temp5; }
   if (temp5 < min_modem)   { min_modem = temp5; }

   c++;
   if (c > 32000)
    {
    c=0;
    max_modem = -32000;
    min_modem =  32000;
    }
   else if (c == 18000)
    {
        *c31_data = ( ((int)(e*0.25)) & 0x0FFFF) | 0x20000 ;
        }
   else if (c == 12000)
    {
          *c31_data = (max_modem & 0x0FFFF) | 0x10000 ;
        }
       else if (c == 6000)
        {
    *c31_data = min_modem & 0x0FFFF;
    }
       }




for (i=1; i>(-1); i--)
   {
     /* one-pole delay filter to create a phase delay     */
   modem_multiply = modem_in[i] *
      ( (modem_in[i+2] * coeff1 / coeff3)
      + (modem_in[i+1] * coeff2 / coeff3) );

    /* LPF to recover FSK baseband signal */
   modem_out[i] =
      FILTERIIR(lpf_coeff, delay3, modem_multiply, 1);

  /* decision  logic 0 or 1        */
  /* This decision contains        */
```

```
 /* hysteresis. The data bit only*/
 /* changes if the threshold     */
 /* values are reached           */
 /*
+--+--- f1 = e + HYSTOR
|  |
|  |
    f2 = e - HYSTOR  ---+--+                    */

  if ( modem_out[i] < f2 )
     {
     status = status & 0x0E;   /* data bit = 0                   */
     data_bit = 0;
     }
  else if ( modem_out[i] > f1 )
     {                         /* data bit = 1                   */
     status = status | 0x01;
     data_bit = 0x0200;
     }                          /* else data bit stays as is     */


  if ( (status & 0x07) == 1)    /* !start_bit AND data_bit       */
     {                          /* AND !possible_start           */
     status = 5;                /* set 'possible start' bit      */
     count2 = HALF_FREQ_COUNT ; /* wait 1/2 bit to check         */
     }                          /* start bit again               */


  if  (count2 > FREQ_COUNT-1)
     {                          /* if count2 > FREQ_COUNT        */
```

```
        status = status | 0x08;      /*  set count2 flag               */
        }
     else
        {
        status = status & 0x07;      /*  unset count2 flag             */
        }


     if ( (status & 0x0E) == 0x0C ) /* if   !start bit AND             */
        {                            /* possible start AND count2      */
        if (status & 0x01)           /* if 'data bit' = 1              */
  {                          /*                         */
  count1     = 0;            /* then we have a valid    */
  count2     = 0;            /*  'start bit'            */
  data_byte = 0;             /*                         */
  parity     = 0;            /* set 'parity bit' low    */
  status     = 2;            /* set 'start bit' flag    */
  }                          /* lower 'possible start' flag */
        else                         /* and lower 'count2' flag        */
  {                          /* else mis-trigger        */
  status     = 0;            /* lower all flag bits     */
  }
        }

     if ( (status & 0x0A) == 0x0A)  /* if    count2 > FREQ_COUNT       */
        {                            /*   AND  start_bit               */
        data_byte = data_byte >> 1; /* We have data, shift down        */
        data_byte = data_byte | data_bit;
        count1++;                    /* increment bit count            */
        count2 = 0;                  /* reset bit length counter       */
```

```
    if ( count1 == 10 )         /* have we received byte?       */
  {
  if ( data_byte & 0x200 )
  {   /* if valid stop bit            */
  *uart_data = data_byte & 0xFF;
  }               /* send byte to PC dos level    */
    /* wait till the end bit passes */
    /* so we don't retrigger on it  */
  count2 = FREQ_COUNT - HALF_FREQ_COUNT ;
  }
    if ( count1 > 10 )
        {
        status = 0;   /* lower all flag bits          */
        }
    }
  count2++;
  }

  /* output voice loudness        */
/* temp = (int) (outputR * SCALE_FACTOR2) ;
*/
temp = (int) (modem_out[0] * 0.95) ;
if (temp > 32255 )  { temp =  32255; }
if (temp < -32255)  { temp = -32255; }
  /* -32255 < temp < 32255        */
        temp = temp & 0x0FFFF;        /* only 16 bit channel          */


        mic = (int) (outputL * 0.2);
if (mic > 32355 ) { mic = 32255; }/*    -32255 < mic < 32255      */
if (mic < -32255) { mic =-32255; }/*    16 bit ADAC                  */
mic = mic << 16;


  /*  ______ L __________ R _____ */
                    /*  | to phone line | to ear  | */
*serial_data_tx = (temp | mic);   /*  |   "mic"       | "temp"  | */
  /*  ___________________________ */



if (*uart_flag & 0x01)              /* if the PC sends us a "9"      */
 {                           /* on the serial port, then     */
 b = *uart_data;             /* hang up the phone            */
 b = b & 0x0FF;


 if ( b == 0x09)
 {
 PHONE_HOOK(ON);
 goto BEGIN ;
 }
 else if ( b == 0x10)  { e=e-100;}/*threshold fine tuning */
 else if ( b == 0x11)  { e=e+100;}
 else if ( b == 0x14)  { adaptive = (adaptive ^ 0x001) ;}
 else if ( b == 0x12)
 {                    /* fine tuning for delay filter */
 dc = dc - 25;
 }
 else if ( b == 0x13)
 {
 dc = dc + 25;
 }
 if ( e >  32000)  { e = 32000; }
 if ( e < -32000)  { e =-32000; }


 f1 = e + HYSTOR ;
      f2 = e - HYSTOR ;



 asm("        PUSH R0  ");
 asm("        PUSH R1  ");
 asm("        PUSH AR0 ");
 asm("        PUSH AR1 ");
 asm("        TRAP 4   ");
 asm("        POP  AR1 ");
 asm("        POP  AR0 ");
 asm("        POP  R1  ");
 asm("        POP  R0  ");
 }  /* end if */

}  /* end while(1) */


}



/* ----------AHHHHHHHHHHhhhhhhh!!!!!!!!!-----------------------------------*/

void INITIALIZE_ANALOG(void)                /* Initialize Analog A/D and D/A port
{

/*------> ANALOG CONTROL WORD #1
     |       |       |       |       |       |
0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
  \___/ | \_/     \_/ |                     |
 ___________|   |  |        |  |______reserved______|
    DFR /                 |  |      MCK
    001 - 16kHz     Stereo |      01 - internal XTAL 1
    000 - 8 kHz            |
      with                DF
      MCK=01              00 - 16bit linear
  01 - 8 bit mu law
  10 - 8 bit A law

--------> ANALOG CONTROL WORD #2
       0000000000000000  reserved

--------> ANALOG CONTROL WORD #3                        0x400010F0
  |       |       |       |       |       |       |       |       |
  0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 0 0 0 0
  | | |_________| | | |_________| |_| | | |_____| |_____| |_____|
  | |     |     res|        |      | ovr| L. Channel |    R. Channel
  | |  left out    |        |     resv | input gain |    input Gain
  | |  attenuation | right out         |       Monitor Path
  | |              | attenuation   Input       Attenuation
  | LE             |               Select
  | 1 - line out   extern          0 - line in
  |     enable     speaker         1 - phone and mic
  |
  HE   1 - headphone and phone out enable
*/
  asm("   TRAP 0          ");             /* Call GET_CONFIG                  */
/* returns addr of MRSS in AR2    */
```

```
  asm("   LDI *+AR2(1), R4");             /* R4=addr of ANALOG_INIT func     */
  asm("   LDI 0004h , R5  ");             /* top 1/2 of Analog cntrl word 1 */
  asm("   LDI 1200h , R6  ");             /* bottom 1/2                      */
  asm("   LSH 16,R5       ");             /* roll upper half                 */
  asm("   ADDI3 R5, R6, R0");             /* result in R0                    */
  asm("   AND 0h, R1      ");             /* Analog control word #2          */
  asm("   LDI 8003h , R5  ");             /* top 1/2 of Analog cntrl word 3 */
  asm("   LDI 10F0h , R6  ");             /* bottom 1/2                      */
  asm("   LSH 16,R5       ");             /* roll upper half                 */
  asm("   ADDI3 R5, R6, R2");             /* result in R2                    */
  asm("   LDI 0 , R3      ");             /* disable serial port interrupts */
  asm("   PUSH AR3        ");
  asm("   CALLU R4        ");             /* call MRSS ANALOG_INIT func      */
  asm("   TRAP 5          ");             /* Wait for register to be sent    */
  asm("   TRAP 7          ");             /* Turn interrupts off             */
  asm("   POP AR3         ");

}

/*-----------------------------------------------------------------*/
void PHONE_HOOK(int i)
{
asm("   LDI *-FP(2), R0  ");    /* get integer "i" off stack      */
asm("   TRAP 3    ");           /* on or off-hook accord to i     */
asm("   OR 2000h, ST ");

}

/*-----------------------------------------------------------------*/
```

```
float FILTERIIR(float *coeff_addr, float *delay_addr, float input, int stages)
/*
____ { A0i + A1i z(-1) + A2i z(-2) }
H(z)= K  || ------------------------------
 ||   { 1 + B1i z(-1) + B2i z(-2) }


     in---<>-----+---------o--------+-----  ..... next section
   K       |         |         |
  |       [z-1]        |
  +--<]-----|----[>--+
  |  -B1i   |   A1i   |
  |       [z-1]        |
  |          |         |
  +--<]-----o----[>--+
     -B2i       A2i

coeff variable goes [ K,   B11, B12, A12, A11,   B21, B22, A22, A21, .... ]

*/
{
  asm("   LDI *-FP(3), AR0          ");  /* address of delay elements      */
  asm("   LDI *-FP(2), AR1          ");  /* address of coefficients        */
  asm("   LDF *-FP(4), R3           ");  /* input sample                   */
  asm("   MPYF R3, *AR1++, R3       ");  /* R3 = (in * K)                  */
  asm("   MPYF *AR0, *AR1++, R0     ");  /* R0 = x[z-1] * B1i              */
  asm("   LDI *-FP(5), RC           ");  /* load repeat counter            */
  asm("   RPTB ENDFLT               ");  /* block repeat                   */
  asm("   MPYF *++AR0, *AR1++, R1   ");  /* R1 = x[z-2] * B2i              */
  asm("|| SUBF R0, R3               ");  /* R3 = R3 - R0                   */
  asm("   MPYF *++AR0, *AR1++, R0   ");  /* R0 = x[z-2] * A2i              */
```

```
    asm("|| SUBF R1, R3, R2            ");  /* R2 = R3 - R2                          */
    asm("   LDF *AR0++, R3             ");  /* R3 = x[z-1]                           */
    asm("   STF R2, *-AR0              ");  /* update new [z-1] temp                 */
    asm("|| STF R3, *AR0               ");  /* shift old [z-1] to new [z-2]          */
    asm("   MPYF *AR0++, *AR1++, R0    ");  /* R0 = x[z-1]*A1i                       */
    asm("|| ADDF R0, R2                ");  /* R2 = R2 + x[z-1]*A2i                  */
    asm("ENDFLT MPYF *AR0, *AR1++, R0");    /* R0 = next B1i*x[z-1]                  */
    asm("|| ADDF R2, R0, R3            ");  /* R3 has stage output                   */
    asm("   LDF R3 ,R0                 ");
}


/*---------------------------------------------------------------------------*/
void DELAY(void)
{
unsigned i;

for (i=0;i<60000;i++)              /* wait and do nothing */
{
asm("   NOP     ");
asm("   NOP     ");
asm("   NOP     ");
}
}
```

E. ELF Board GMSK Modulator C Source Code

```
/*
 *  ======== msk_tx.c ========
Transmit data from the serial port using sinusoidal
preshaped MSK, or GMSK with LPF bandwidth set to about
0.5 .

Read and write to the A/D,D/A without the use
of the spox SIG_IO functions

March 26/96

V3.0 July 9 / 96
 */

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

#define off             1
#define on              0
#define SCALE_FACTOR    2
#define SCALE_FACTOR2   0.25
#define SAMPLE_FREQ     16000
#define ONECYCLE        28
#define BAND_WIDTH      600
#define FREQUENCY1      2250
#define MODEM_AMPLITUDE 750
/**#define MODEM_AMPLITUDE 0 */
#define BIG_PRIME 2147483647
```

```
/*prototypes */
void  ANALOG_INIT(void);
float FILTERIIR(float *, float *, float, int);
void  PHONE_HOOK(int);
void  delay(void);
void  DIAL_PHONE(int);
void  GET_PHONENUMBER(int *) ;



/*  ======== smain ========*/

void main(int argc, char *argv[])
{

    int                 i, status, in;
    int                 int_phase;        /* rounded current phase of modem   */
    int                 modem_count=0;    /* counter for 300Hz modem signal   */
    register int        temp, mic;
    float               phase=0;
    register int        pc_serial_data;  /* current data word register       */
    float               outputL, outputR;/* Voice and data output            */
    float               inL, inR;         /* voice input                      */
    volatile unsigned   *flag_ptr;        /* ADAC serial flag register        */
    volatile unsigned   *serial_data_rx,
                        *serial_data_tx; /* address of ADAC                  */
    volatile unsigned   *uart_data;       /* address of PC uart data register */
    volatile unsigned   *uart_flag;       /* address of PC uart flag register */
    volatile unsigned   *uart_modem;      /* address of PC uart modem register*/
    volatile unsigned   *uart_interrupt; /* address of PC uart int. register */
    volatile unsigned  *uart_status;    /* address of PC uart status reg.    */
    volatile unsigned  *data_ex_reg;    /* address of Data exchange reg.     */
    volatile unsigned  *ex_status_reg;  /* address of exchange status reg.   */
    float              delay1[24],
            delay2[24];     /* filter delay elements          */
    float              sine_lookup[360];/* sine table                        */
    float          noise_amplitude; /* user def. white noise amplitude  */
    int                current_bit = 0; /* bit of data currently sending    */
    int                previous_bit = 0;/* last state                        */
    int                hang_up_count = 0;
    int                parity;          /* partiy bit                        */
    unsigned           a,b;
    int                c,d;
    float              msk[4][ONECYCLE] ;
    int                shift_count;     /* # of bits shifted out             */
    int                phone_number[12] = {7,6,6,8,8,2,1,1};
    float              phase_lookup[360] ;
    float              bit_amplitude;
    float          noise, noise2;
    unsigned         signal_on = 1;

/*   float                notch_coeff[] = { 0.418455,
-0.705841,   0.16486,   1,   -1.45052,
-1.49359 ,   0.863056,  1,   -1.38197,
-1.51534 ,   0.976271,  1,   -1.21189,
-0.734349,   0.96589 ,  1,   -1.00872,
-0.524041,   0.789581,  1,   -0.902578
}; */
/* BW=850Hz Center=2.4kHz Trans=200Hz elliptic */
```

```
    float              notch_coeff[] = { 0.410044,
-1.34719,    0.630340,  1,   -1.47343,
-1.50188,    0.913784,  1,   -1.42857,
-1.48718,    0.94337,   1,   -1.30367,
-0.712273,   0.94169,   1,   -1.09657,
-0.607374,   0.871343,  1,   -0.921930,
-0.308051,   0.435529,  1,   -0.845701
};

  /* address of ADAC serial registers     */
flag_ptr       = (volatile unsigned *) 0x808040;
serial_data_rx = (volatile unsigned *) 0x80804C;
serial_data_tx = (volatile unsigned *) 0x808048;
  /* address of UART interface registers  */
uart_data      = (volatile unsigned *) 0xA00000;
uart_flag      = (volatile unsigned *) 0xA00005;
uart_modem     = (volatile unsigned *) 0xA00004;
uart_interrupt = (volatile unsigned *) 0xA00001;
uart_status    = (volatile unsigned *) 0xA00006;
  /* address of data exchange registers   */
data_ex_reg    = (volatile unsigned *) 0x900000;
ex_status_reg  = (volatile unsigned *) 0x900002;

for (i=0;i<360;i++)       /* generate voltage to phase lookup     */
{
phase_lookup[i] =
     ( (i*BAND_WIDTH/720) + FREQUENCY1 ) * 360 / SAMPLE_FREQ;
}

for (i=0;i<360;i++)       /* generate sine look-up table          */
```

```
{
sine_lookup[i] = (sin(i*2*3.141592654/360));
}

for (i=0;i<ONECYCLE;i++)  /* generate sinusoidal pulse shaping    */
{                     /* for GMSK approximation waveforms     */
msk[0][i] = 1 ;                               /* 0 -> 0       */
msk[1][i] = cos(3.1415927*i/ONECYCLE);        /* 0 -> 1       */
msk[2][i] = -cos(3.1415927*i/ONECYCLE);       /* 1 -> 0       */
msk[3][i] = -1 ;                              /* 1 -> 1       */
}

temp           = *serial_data_rx ;
pc_serial_data = 0;
*flag_ptr      = 0x00;

asm("             AND 0, IE ");/* turn off serial ports int     */
asm("             LDI 0, IF ");    /* clear interrupts             */
asm("             OR 2000h, ST "); /* turn on interrupts           */
UART_init(12, 0x0003, 0);          /* 9600 baud, 1 stop bit        */
*uart_interrupt = 0x0;
*uart_modem = 0x01;


BEGIN:  /* let's begin */
hang_up_count = 0;
 /* Get phone number from User    */
GET_PHONENUMBER(&phone_number[0]);
PHONE_HOOK(off);                     /* take phone off-hook            */
for (i=0;i<10;i++)
```

```
     {
     delay();
     }

ANALOG_INIT();                    /* begin ADAC processing          */
delay();
delay();

for (i=0; i < phone_number[0]; i++)
{
asm(" PUSH R0  ");
asm(" PUSH R1  ");
asm(" PUSH AR0 ");
asm(" PUSH AR1 ");
asm(" TRAP 4   ");
asm(" POP  AR1 ");
asm(" POP  AR0 ");
asm(" POP  R1  "); /* Dial phone, 1 digit at a time  */
asm(" POP  R0  ");
DIAL_PHONE(phone_number[i+1]);
}

asm(" PUSH R0  ");
asm(" PUSH R1  ");
asm(" PUSH AR0 ");
asm(" PUSH AR1 ");
asm(" TRAP 4   ");
asm(" POP  AR1 ");
asm(" POP  AR0 ");
asm(" POP  R1  ");
asm(" POP  R0  ");

i=0;
noise_amplitude = 0;

while(1)
{
while( (*flag_ptr & 0x01) == 0 );
/* wait for data on serial port*/

temp = *serial_data_rx ; /* get ADAC word from serial port */
mic  = temp;
temp = temp >> 16;        /* upper 16 bits is left channel  */
inL  = (float) temp ;     /* convert to floating point      */
mic  = mic << 16;         /* get rid of top 16 bits         */
mic  = mic >> 16;         /* sign extend                    */
inR  = (float) mic ;      /* lower 16 bits is right channel */

 /* preform IIR filtering on the    */
 /* two channels to notch out the   */
 /* modem carrier                   */
outputR = FILTERIIR(&notch_coeff[0], &delay1[0], inL, 5);
outputL = FILTERIIR(&notch_coeff[0], &delay2[0], inR, 5);



temp = (int) (outputR * SCALE_FACTOR) ;
 /* convert back to integer         */
if (temp > 32255 )        /* preform hard limiting to        */
{                  /* ensure a 16 bit channel        */
```

```
temp = 32255;
}
if (temp < -32255)
{
temp = -32255;
}                  /* -32255 < temp < 32255          */
temp = temp & 0x0FFFF;   /* only 16 bit channel              */


a = (*uart_status & 0x010);
 /* UART CTS bit raised when       */
 /*    the host PC wants to send   */
b = (*uart_flag & 0x01); /* UART Data Ready flag              */

if ( a && (!b) )
{                  /* if CTS and no data in buffer   */
*uart_modem = 3; /* raise RTS line(ready 2 receive)*/
}
if (b)                      /* if there is data in the buffer */
{
pc_serial_data = *uart_data;
 /* get 8 bits of data from the    */
 /* serial port                    */
pc_serial_data = pc_serial_data & 0xFF ;

if ( pc_serial_data == 0xF2)
     {             /* If the special character 0xF2  */
 /* is sent, add 1 to hang_up_count*/
     hang_up_count++;
     }
```

```
else
     {
     hang_up_count = 0;
     }           /* otherwise, reset hang_up_count */

parity = 0;

pc_serial_data = pc_serial_data << 1;
pc_serial_data = pc_serial_data | 0x400 ;
 /* shift serial data up one and   */
 /* begin with a 'start' bit       */
 /* add 'end' bit                  */
modem_count    = 0;
current_bit    = 1;
previous_bit   = 0;
shift_count    = 0;
}

if (modem_count > (ONECYCLE - 1) )
{
 /* Send bits out telephone modem  */
 /* one bit at a time. Shift them  */
 /* out every time we pass         */
 /* ONECYCLE time                  */
 /* Also calculate parity bit      */
pc_serial_data = pc_serial_data >> 1;
previous_bit   = current_bit;
current_bit    = pc_serial_data & 0x01;
parity         = parity + current_bit;
shift_count++;   /* increment # of bits shifted    */
```

```
modem_count    = 0;
 /* reset clock for modem counting */

/* if ( shift_count == 8 )
{
pc_serial_data = (int) (parity % 2) ;
pc_serial_data = pc_serial_data << 1;
}
*/
if ( shift_count > 12 )
{       /* lower RTS (ready for next char)*/
*uart_modem = 1;
}
}
 /* add phase to create a sine     */
 /* wave at the frequency          */
 /* necessary depending on the     */
 /* current bit transmitting       */

modem_count++;
c = current_bit + (previous_bit * 2) ;
 /* calculate which transition we  */
 /* are in (0->0,0->1,1->0,1->0)    */
bit_amplitude = msk[c][modem_count];
 /* GMSK amplitude at given time   */
d = (int) ((bit_amplitude * 178) + 180);
 /* scale to integer between 0-360 */
phase = phase + phase_lookup[d];
 /* phase to add depending on      */
 /* given amplitude                */
```

```
if ( phase > 359 )
     {                    /* sine lookup table only goes     */
     phase = phase - 360;/* to 360 deg                       */
     }
int_phase = (int) phase;

 /* introduce noise using a    */
 /* pseudo-Random number generator  */
 /* At 16kHz,it repeats every 34hr */
 /* It's amplitude is determined   */
 /* by the value passed to it      */
 /* from DOS through the data      */
 /* exchange register              */
         in = *ex_status_reg;
         in = in & 0x02;
 if (in)
 {
 in = (int) *data_ex_reg;
 in = in << 16;
 in = in >> 16;
 if (in == 0x01)
 {
 signal_on = signal_on ^ 0x01 ;
 }
 else
 {
 noise_amplitude = (float) in ;
 /* noise_amplitude = noise * noise * 2 ;*/
 }
 }
```

```
noise = ((float)rand() / BIG_PRIME) - 0.5;
noise = noise * noise_amplitude ;

/* noise = cos(2 * 3.141593 * noise ) * noise_amplitude;
noise2 = log( BIG_PRIME / (float)rand() );
noise2 = sqrt(noise_amplitude * noise2);
noise = noise * noise2 ;
*/

if (signal_on)
  {
  outputL = (outputL * SCALE_FACTOR2) +
   noise +
(sine_lookup[int_phase] * MODEM_AMPLITUDE);
 /* add modem signal to voice      */
  }  /* signal, scale voice amplitude  */
else
  {
  outputL = (outputL * SCALE_FACTOR2) + noise ;
  }
 /* for test purposes, turn off    */
 /* the carrier and only add noise */

mic = (int) outputL ;    /* convert to integer                */

if ( mic > 32255 )
{
mic = 32255;
}
```

```
if ( mic < -32255 )
{
mic = -32255;
}
                      /* -32255 < mic < 32255              */
 /* 16 bit ADAC                        */
 /*                                    */
 /*   ______ L __________ R _____      */
         mic = mic << 16;          /*  | to phone line | to ear   |    */
 /*  |    "mic"        | "temp"  |     */
 /*  __________________________        */
         *serial_data_tx = temp | mic;
 /* output voice and data signals  */
 /* to DAC                             */

         if (hang_up_count > 4)    /* if user sent hang up code        */
   {
   PHONE_HOOK(on);
   asm("    LDI 7F3Fh , R5  ");  /* top 1/2 of Analog word 3*/
   asm("    LDI 10F0h , R6  ");  /* bottom 1/2              */
   asm("    LSH 16,R5 ");        /* roll upper half         */
   asm("    ADDI3 R5, R6, R0 "); /* result in R0            */
   asm("    LDI 0FFFFh , R5  "); /* top 1/2 of bit mask     */
   asm("    LDI 0FFFFh , R6  "); /* bottom 1/2              */
   asm("    LSH 16,R5 ");        /* roll upper half         */
   asm("    ADDI3 R5, R6, R1 "); /* result in R1            */
   asm("    TRAP 2");            /* hang up, mute volume    */
   *uart_modem = 1;
   goto BEGIN ;                  /* and restart             */
   }
```

```
       }


}



/* ----------AHHHHHHHHHHhhhhhhh!!!!!!!!!-----------------*/

void ANALOG_INIT(void)   /* Initialize Analog A/D and D/A port */
{
/* analog control word 1
     |       |       |       |       |
0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
  \___/ | \_/     \_/ |                     |
 -----------|   |  |       |  |_______reserved______|
    DFR /               |  |      MCK
    001 - 16kHz     Stereo |      01 - internal XTAL 1
    000 - 8 kHz            |
      with                DF
      MCK=01              00 - 16bit linear
  01 - 8 bit mu law
  10 - 8 bit A law

  analog control word 2

0000000000000000  reserved

  analog control word 3                   0x400010F0
 |       |       |       |       |       |       |       |       |
  0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 0 0 0 0
  | | |_________| | | |_________| |_| | | |_____| |_____| |_____|
| |        |      res|       |         | ovr| L. Channel |     R. Channel
| |  left out       |       |       resv  | input gain |     input Gain
| |  attenuation    | right out          |          Monitor Path
| |                 | attenuation    Input        Attenuation
| LE                |                Select
| 1 - line out   extern              0 - line in
|     enable     speaker             1 - phone and mic
|
HE   1 - headphone and phone out enable

*/


asm("   TRAP 0 ");                /* Call GET_CONFIG                   */
 /* returns addr of MRSS in AR2    */
asm("   LDI *+AR2(1) , R4 ");    /* R4=addr of Analog_Init function*/

asm("   LDI 000Ch , R5  ");      /* top 1/2 of Analog word #1         */
asm("   LDI 1200h , R6  ");      /* bottom 1/2                        */
asm("   LSH 16,R5 ");            /* roll upper half                   */
asm("   ADDI3 R5, R6, R0 ");     /* result in R0                      */

asm("   AND 0h, R1      ");      /* Analog control word #2            */

asm("   LDI 8008h , R5  ");      /* top 1/2 of Analog word #3         */
asm("   LDI 10E0h , R6  ");      /* bottom 1/2                        */
asm("   LSH 16,R5 ");            /* roll upper half                   */
asm("   ADDI3 R5, R6, R2 ");     /* result in R2                      */

asm("   LDI 0 , R3 ");           /* disable serial port interrupt  */
asm("   PUSH AR3 ");
asm("   CALLU R4   ");             /* call MRSS ANALOG_INIT function */
asm("   TRAP 5 ");
asm("   TRAP 7 ");
asm("   POP AR3  ");

}

/*-----------------------------------------------------------------------*/
void PHONE_HOOK(int i)
{
asm("   LDI *-FP(2), R0  ");      /* get integer "i" off stack      */
asm("   TRAP 3    ");             /* on or off-hook accord to i     */
asm("   OR 2000h, ST ");

}


/*-----------------------------------------------------------------------*/
float FILTERIIR(float *coeff_addr, float *delay_addr, float input, int stages)
/*                ^                 ^                 ^          ^
                  |                 |                 |          |
                  |                 |                 |  #stages - 1
       filter coeff.   delay elements     input sample


---- { B0i + B1i z(-1) + B2i z(-2) }
H(z)= K  ||  -----------------------------
 ||   { 1 + A1i z(-1) + A2i z(-2) }
```

```
     in---<>-----+---------o--------+----- ..... next section
   K      |          |         |
   |        [z-1]       |
   |          |         |
   +--<]-----o----[>--+
   |  -A1i    |   B1i  |
   |        [z-1]       |
   |          |         |
   +--<]-----o----[>--+
      -A2i        B2i

coeff variable goes [ K,   A11, A21, B21, B11,   A21, A22, B22, B12, .... ]

   */
{

     asm("   LDI *-FP(3), AR0        ");  /* addr of delay element          */
     asm("   LDI *-FP(2), AR1        ");  /* addr of coefficients           */
     asm("   LDF *-FP(4), R3         ");  /* input sample                   */
     asm("   MPYF R3, *AR1++, R3     ");  /* R3 = (in * K)                  */
     asm("   MPYF *AR0, *AR1++, R0   ");  /* R0 = x[z-1] * B1i              */
     asm("   LDI *-FP(5), RC         ");  /* load repeat counter            */
     asm("   RPTB ENDFLT             ");  /* block repeat                   */
     asm("   MPYF *+AR0, *AR1++, R1  ");  /* R1 = x[z-2] * B2i              */
     asm("|| SUBF R0, R3             ");  /* R3 = R3 - R0                   */
     asm("   MPYF *+AR0, *AR1++, R0  ");  /* R0 = x[z-2] * A2i              */
     asm("|| SUBF R1, R3, R2         ");  /* R2 = R3 - R2                   */
     asm("   LDF *AR0++, R3          ");  /* R3 = x[z-1]                    */
     asm("   STF R2, *-AR0           ");  /* update new [z-1] temp          */
     asm("|| STF R3, *AR0            ");  /* shift old [z-1] to new [z-2]   */
```

```
    asm("   MPYF *AR0++, *AR1++, R0 ");  /* R0 = x[z-1] * A1i              */
    asm("|| ADDF R0, R2              ");  /* R2 = R2 + (x[z-1] * A2i)       */
    asm("ENDFLT MPYF *AR0,*AR1++, R0");  /* R0 = next B1i * x[z-1]         */
    asm("|| ADDF R2, R0, R3          ");  /* R3 has stage output           */
    asm("   LDF R3 ,R0               ");
}

/*----------------------------------------------------------------------------*/
void delay(void)
{
int i;
for (i=0;i<60000;i++)                /* wait and do nothing               */
{
asm("   NOP     ");
asm("   NOP     ");
asm("   NOP     ");
}
}

/*----------------------------------------------------------------------------*/
void DIAL_PHONE(int number)
{
 /* generate dual tone multi-       */
 /* frequency for dialing the       */
 /* phone  An integer 0-9 is sent  */
 /* to the procedure and the tone  */
 /* is sent to the DAC              */
   int                      i,j, int_out, k ,l, int_out1;
   float                    out;
   volatile unsigned        *flag_ptr1;
```

```
  volatile unsigned      *serial_data_rx1,
  *serial_data_tx1; /* address of ADAC*/

/*            freq hi (Hz)
| 1209  1336  1477  1633
   ------------------------------
   697 |  1     2     3     A
      freq lo (Hz) 770 |  4     5     6     B
   852 |  7     8     9     C
   941 |  *     0     #     D   */
/* normalized frequencies        */
  float                   dial_freq_lo[4] =
  { 0.0436, 0.04813, 0.05325, 0.05881 } ;
  float                   dial_freq_hi[4] =
  { 0.07556, 0.0835, 0.0923, 0.10206 };
  int                     number_conversion[2][10] =
  {  { 3,0,0,0,1,1,1,2,2,2 },
     { 1,0,1,2,0,1,2,0,1,2 } };

  flag_ptr1       = (volatile unsigned *) 0x808040;
  serial_data_rx1 = (volatile unsigned *) 0x80804C;
  serial_data_tx1 = (volatile unsigned *) 0x808048;


  for (j=0; j<3000; j++)
     {
     while( (*flag_ptr1 & 0x01) == 0);
     i = *serial_data_rx1;
     *serial_data_tx1 = 10;
     }
   for (j=0; j<3000; j++)
      {
      while( (*flag_ptr1 & 0x01) == 0);

      i = *serial_data_rx1;
      k = number_conversion[0][number];  /* choose the 2 frequencies that */
      l = number_conversion[1][number];  /* go with the digit dialed      */
      out = sin(2*3.141*j*dial_freq_lo[k])+sin(2*3.141*j*dial_freq_hi[l]);
      out = out * 5000;                  /* generate sinusiodal waveform  */
      int_out1 = (int) out;
      int_out = int_out1 | (int_out1 << 16);
      *serial_data_tx1 = int_out ;       /* convert to int and sent to DAC*/
      }



}

/*------------------------------------------------------------------------*/
void GET_PHONENUMBER(int *phone_num)
{
     int      size,in, stat = 0;
     int      a,b,c,d;
     volatile unsigned *uart_data;     /* address of PC uart data register */
     volatile unsigned *uart_flag;     /* address of PC uart flag register */
     volatile unsigned *uart_modem;    /* address of PC uart modem register*/
     volatile unsigned *uart_status;   /* address of PC uart status reg    */
     uart_data    = (volatile unsigned *) 0xA00000;
     uart_flag    = (volatile unsigned *) 0xA00005;
     uart_modem   = (volatile unsigned *) 0xA00004;
     uart_status  = (volatile unsigned *) 0xA00006;




     c=0; d=0;
     in = *uart_data;

     while (c == 0)
{
a = (*uart_status & 0x010);      /* UART CTS bit                    */
 /* raised when host wants to send */
b = (*uart_flag & 0x01);         /* UART Data Ready flag            */

if ( (a>0) && (b==0) )
{                           /* if CTS and no data in buffer   */
*uart_modem = 3;            /* raise RTS line(ready 2 receive)*/
}
if (b)                           /* if there is data in the buffer */
{
in = *uart_data;            /* get digit from serial port     */
in = in & 0x0FF;
phone_num[d] = in;          /* add it to phone number list    */
*uart_modem = 1;
if ( stat == 1)
     {                      /* if confirmed phone number start*/
     d++;                   /* then add 1 number of digits    */
     if (d>12)              /* if we already have 12 numbers  */
 {
 c = 1;          /* then end getting digits       */
 }
     }
if (in == 0xF1)           /* 0xF1 is the code for begin     */
     {                    /* phone number transfer          */
     stat = 1;
     }

if ( (d>0) && (in == 0xF0) )
     {                    /* 0xF0 is the code for end       */
     c = 1;               /* phone number transfer          */
     }
}


}
}
```