Source and Channel Coding for Low-Bandwidth Speech Communication between Optoelectronic Devices

by

Daniel S. Perry

S.B. E.E. M.I.T. 2008

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

[June]
~~May~~ 2009

Author . . [illegible] . . . .
Department of Electrical Engineering and Computer Science
May 22, 2009

Certified by . . [illegible] . . . .
Yoel Fink
Associate Professor
Thesis Supervisor

Certified by . . [illegible] . . . .
Ofer Shapira
Postdoctoral Associate
Thesis Co-Supervisor

Accepted by . . . . . [illegible] . . . .
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Thesis

Source and Channel Coding for Low-Bandwidth Speech Communication between Optoelectronic Devices

by

Daniel S. Perry

Submitted to the Department of Electrical Engineering and Computer Science

May 22, 2009

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Optical communication is one solution to the communication problem that many military forces face in today's urban environments. The recent advances in optoelectronic fibers by the Photonic Bandgap Fibers and Devices Group allow optoelectronic fibers to be woven into standard military equipment. This thesis aims to design an optical communication system that would be capable of using these fibers. Since this project was undertaken with several collaborators this thesis focuses on the source and coding techniques used for an optical communication system. A freeware linear predictive vocoder developed by Speex was eventually adopted to compress speech to the bandwidth of the optoelectronic fibers. Several channel coding techniques were examined for an optical channel. This thesis describes the pulse modulation technique eventually used. Finally, this thesis explores how the optical communication system can be integrated into standard military equipment.

Thesis Supervisor: Yoel Fink
Title: Title Associate Professor

Thesis Co-Supervisor: Ofer Shapira
Title: Postdoctoral Associate

Acknowledgements

The author wishes to thank Yoel Fink for the opportunity to perform research. Also, Ofer Shapira deserves recognition as not only a co-supervisor but as a mentor for this research. Without Matthew Spencer's guidance and help on the previous versions of this system this project could never have been fully realized. My co-researchers played an equally important role in this research. I will never forget the countless hours spent with Agustya Mehta and Paul Yang assembling and testing the system in lab and I must thank them for all of their efforts. I would also like to thank the ISN for their support and funding of this project.

Finally, and most importantly, I would like to thank my friends and family for their continued support of all my endeavors. Without them, none of this would have been possible.

Contents

List of Figures

Chapter 1

Optical Communication System Overview

1.1 Introduction

Recent advances in optoelectronic fibers developed by the Photonic Bandgap Fibers and Devices Group (PBG) have provoked a wide array of different applications for these fibers [1]. One such application is an optical line-of-sight communication system for the Special Operations Forces (SOF). This optical communication system enables a soldier to extend the reach of voice to the range of vision in a safe, discriminate and far reaching manner. Line-of-sight communication between friendly forces is still a non-trivial problem, specifically as it pertains to dense multi-dimensional urban environments in which RF based capabilities may be challenged by multipath interference and jamming devices. By using the infrared band (850 nm) for interrogating, sensing and communicating the technology offers a covert capability that is impermeable to RF jamming and tracking [2].

The Institute for Soldier Nanotechnology (ISN) contracted the PBG group to implement a prototype of this system. Since the necessary optoelectronic fibers were still not available at the inception of this project, an emulator using a commercial photodiode was designed. However, this emulator was designed to be "fiber-ready", as the photodiodes could be replaced with the optoelectronic fibers once they become available. In order to effectively discuss the design for this optical communication system, it is necessary to first examine the system requirements and progress made prior to the start of this thesis.

1.2 System Requirements

There are several requirements that the vest must meet to make it suitable for an SOF in a hostile environment. First, the communication system must be discrete and difficult to detect. This is why infrared (*850 nm*) LED's were chosen as they are very difficult to spot with the unaided eye. Next, the soldier must be able use the system in any environment – urban, indoors, daylight and night operations. The system must also be practical for the different types of formations and tactical situations that arise in modern warfare.

One possible formation is the triangular formation shown in *Figure 1-1*. In this scenario soldiers maintain a distance of approximately *20 meters*, with the back soldier covering the front two as shown. The system should allow the back soldier to communicate with the front two soldiers while facing directly forward (the soldier should not have to aim the transmitter). This means that the system must be able to cover *60°* on both transmitting and receiving ends. A bidirectional system is needed as soldiers in the front must be able to respond to interrogations by the soldier in the back. Finally, to keep the vests consistent and allow both forward and backward communication, transmitters and receivers must be placed on both the front and the back of the vest.

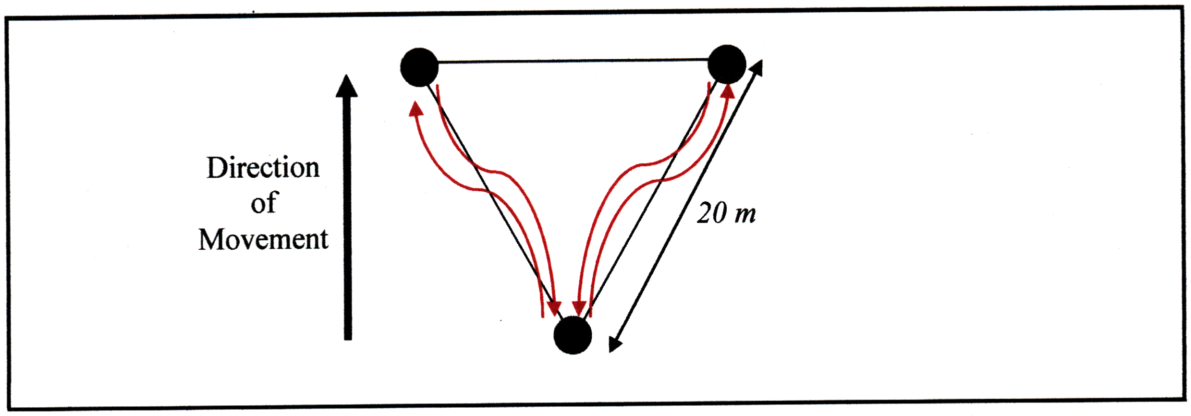


Figure 1-1: Diagram of a possible tactical formation that this optical system was designed to be used for. This formation requires a minimum range of 20 meters and at least a 60° of coverage. Also, because the vest is not designed to be used in stationary conditions, it must not be susceptible to normal military movements (i.e. walking, crouching, etc.)

The final requirement of this system is that is must discretely fit in the vest shown in *Figure 1-2*. All electronic components need to fit securely within the vest without adding unnecessary bulk. The soldier must not be able to notice a significant difference between this vest and a standard-issue vest.

Figure 1-2: Military vest used for testing purposes and final integration. There are several pockets inside that vest that can be used to house the electronics. Wires need to be routed through the shoulders to connect the front and back transmitter and receivers.

1.3 Previous Work

1.3.1 Analog, One-Directional Optical Communication System

Several optical communication systems had been developed within the PBG before work commenced on this thesis. The first such system used analog inputs from a microphone to modulate the amplitude of a *PEQ-2* laser designator to transmit speech. A silicon detector was mounted on the sides of a helmet as *Figure 1-3* shows and this allowed for one-directional communication [2].

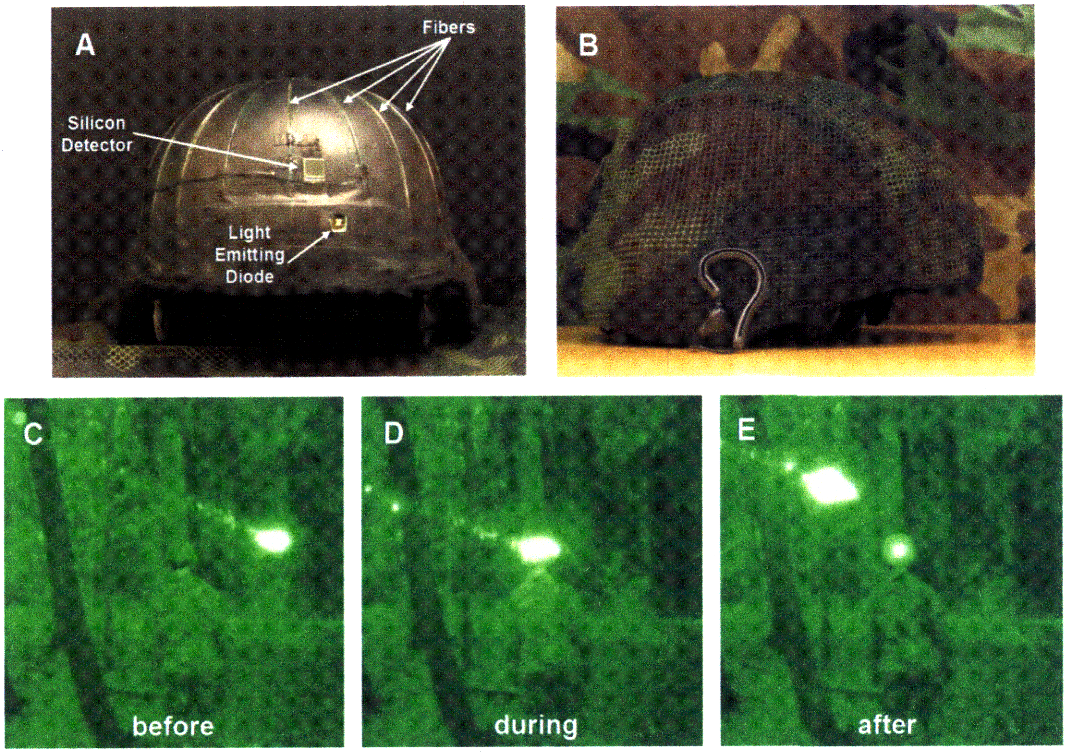


Figure 1-3: Optoelectronic fibers and silicon detectors embedded into the helmet for an analog communication system. This system also provided a means for friendly force identification via an infrared LED that remains illuminated after the helmet is illuminated by the laser designator [2].

There were several problems with this initial system. First, amplitude modulation is susceptible to noise from ambient light. The system was not functional during the day and even moonlight flooded the receiver so that the transmitted signal could not be detected. Also, the *PEQ-2* laser designator is very directional and requires precise aiming and so the soldier must divert his attention from his surroundings. Because of these problems it was decided that a digital communication system was necessary. So the group began simultaneous development on two different systems – a low-bandwidth tabletop system, and a high-bandwidth portable system.

1.3.2 Digital, Low-Bandwidth Communication System

A low-bandwidth digital communication system was designed so that the optoelectronic fibers could eventually replace the silicon detectors. Because the bandwidth of the fibers is substantially less than the silicon detector, digital voice compression was necessary. The group decided to purchase a commercially available vocoder. This vocoder, developed by Compandent (www.compandent.com) used a Mixed-Excitation Linear Predictive (MELPe) algorithm to compress speech to a rate of *2.4 kbps*. *Figure 1-4* shows the wired implementation of this system.

Like the original analog system, the MELPe vocoder has several drawbacks. First, the proprietary algorithm was expensive and so the group could only afford to use the evaluation boards. This allowed practically no hardware or software flexibility. The bulky evaluation boards could only be used for a tabletop demonstration to show that it was possible compress speech to the fiber bandwidth. Also, the evaluation boards communicated via a two-wire serial interface, which consisted of a data and clock signal. While a two-wire interface is acceptable for wired connections, it would not be possible with the optical system and so work with the MELPe vocoder was later abandoned. This project however showed that it was possible to obtain telephone quality speech with a data rate less than *10 kbps*.

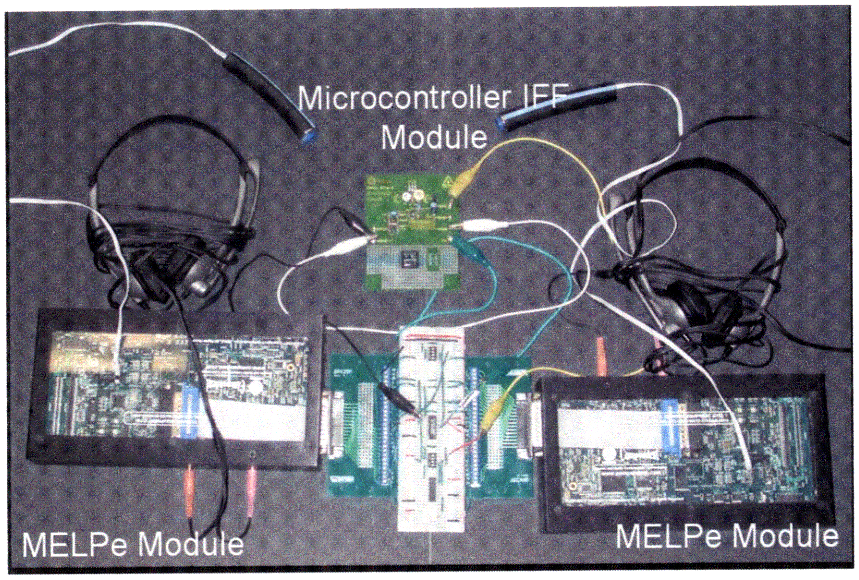


Figure 1-4: Digital communication link using Compandent's commercially available MELPe module. This system can transmit speech at a rate of 2.4 kbps via a two wire interface. Work on this system ceased because of cost and collaboration constraints.

1.3.3 Digital, High-Bandwidth Optical Communication System

In order to show that digital communication was possible across an optical channel, a high-bandwidth system was developed simultaneously. This system used a *10-bit* serial analog-to-digital converter (ADC) to sample a microphone at a rate of *8 kHz* and Microchip's *PIC18F4550* microcontroller was used to output a serial bit stream. *Figure 1-5* shows the block diagram and an image of the board that was developed. A return to zero encoding scheme was used for bit detection. The microcontroller added a start flag (logical one) and stop flag (logical zero) to the front and back of each bit, respectively. This ensured reliable detection of bits and minimized the probability of error. However, the functions responsible for encoding the data consumed most of the processing power of the PIC and so the maximum data rate that could be achieved was approximately *50 kbps*. Since audio transmission using this method required a bit rate of at least *240 kbps*, audio could not be optically transmitted with this system and so the audio components were not populated. Instead, a USB interface was added to the board which allowed the user to transmit a byte via a laptop. While this system could not transmit audio, it proved that digital optical communication was feasible.

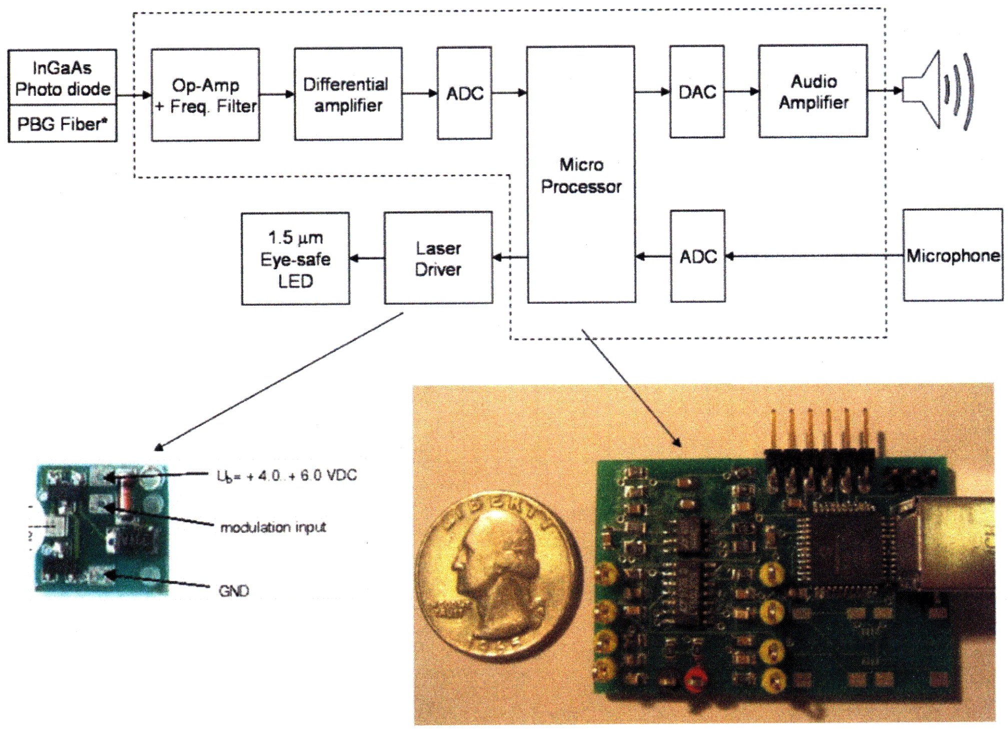


Figure 1-5: The first implementation of a digital optical communication system. The PIC18F4550 was not optimized to encode speech signals fast enough and so the USB connection was used to show that digital transmission was possible.

1.4 System Overview

At the inception of this thesis the aforementioned digital communication systems had been designed, tested, and demonstrated. It was evident from these demonstrations that many modifications were necessary in order to meet the system requirements set forth by the ISN. An electronics team of Agustya Mehta, Paul Yang, and I was commissioned with the task of modifying the system to meet these requirements. Work was divided as follows – Agustya and Paul developed the optical transmitters and receivers necessary to meet the range and angle requirements while I designed the encoding algorithms and associated hardware.

From the demonstrations of the previous systems , it was clear that a new method for encoding was needed. The MELPe vocoder, though effective, is impractical for an optical communication system as it needs to simultaneously transmit clock and data signals. The return-to-zero encoding algorithm prevents errors, but was not optimized for an optical channel, and is not bandwidth efficient. This thesis explains the theory behind the source and channel coding algorithm used for the final system and how it was implemented. It is divided as follows:

- Chapter two describes the source coding techniques available for speech compression.
- Chapter three outlines the optimal channel coding algorithm for an optical channel
- Chapter four describes the integration of the processing unit with the boards designed by other members of the team.

Chapter 2

Source Coding for Speech

2.1 Introduction

Source coding for speech signals is a topic that has been studied extensively and several different types of vocoders have already been proposed. This paper does not propose a novel source code for speech but rather explores the available options that meet the bandwidth requirements of the fibers. There are two types of vocoders that will be analyzed for this application; a channel vocoder and a linear predictive vocoder. The common element between these two vocoders is that they use the source filter model to describe speech.

A channel vocoder calculates the envelope coefficients for several different frequency bands. Because the envelope values contain mostly lower frequency content, they can be decimated and this is how speech is compressed. The next section describes the specific details of the channel vocoder.

On the other hand, an autoregressive linear predictive encoder fits an all-pole filter to the speech signal. This filter is designed to minimize the error between the speech signal and the filter. This can be accomplished by using the autocorrelation of the speech signal and the Yule-Walker equations. Initial tests were performed with Matlab, but ultimately the system was implemented using the freeware Speex vocoder.

2.2 Channel Vocoder

In 1928 Homer Dudley developed the channel vocoder, which was one of the first efforts to encode speech. Although this method is currently outdated, it provided the framework for several modern vocoders. The underlying foundation of the channel vocoder is the source filter model of speech. This model is shown in *Figure 2-1* and it is based on the idea that sound is produced when a pulsed stream of air is filtered by the vocal tract. The voicing source can be thought of as a pulse train corresponding to the pitch of the speaker. This pitch can vary from approximately *80 Hz* to *800 Hz* depending on the speaker. On the other hand, the noise source is typically represented as white Gaussian noise. This physical model can then be used as the basis for speech encoding. Only the pitch values and model for the vocal filter need to be transmitted – a synthesizer can then recreate the speech signal using the source filter model.

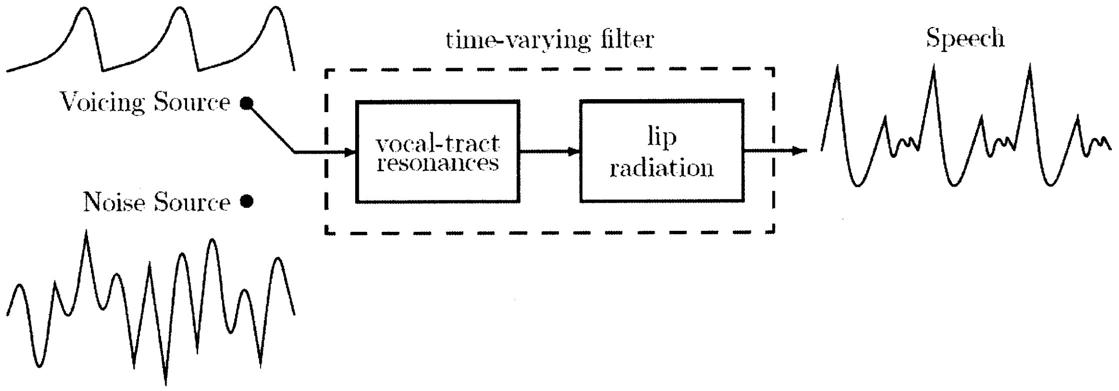


Figure 2-1: The source filter model for speech. This model assumes that either a pulse train or white noise is shaped by the vocal tract depending on the frame is voiced or unvoiced. In order to compress speech, the coefficients describing the filter are calculated and transmitted [3].

In order to implement a real time system, the speech must be broken into frames for processing. Selection of the frame or window size is critical. If the frame is too short, there is little advantage of using the source model. If the frame is too long (i.e. encompasses different types of sounds) then the filter will be inaccurate. Typical speech encoding methods use a frame size between *10* and *50 ms*. Next, the encoder must compute the coefficients for the filter. *Figure 2-2* shows the block diagram for the channel vocoder analyzer.

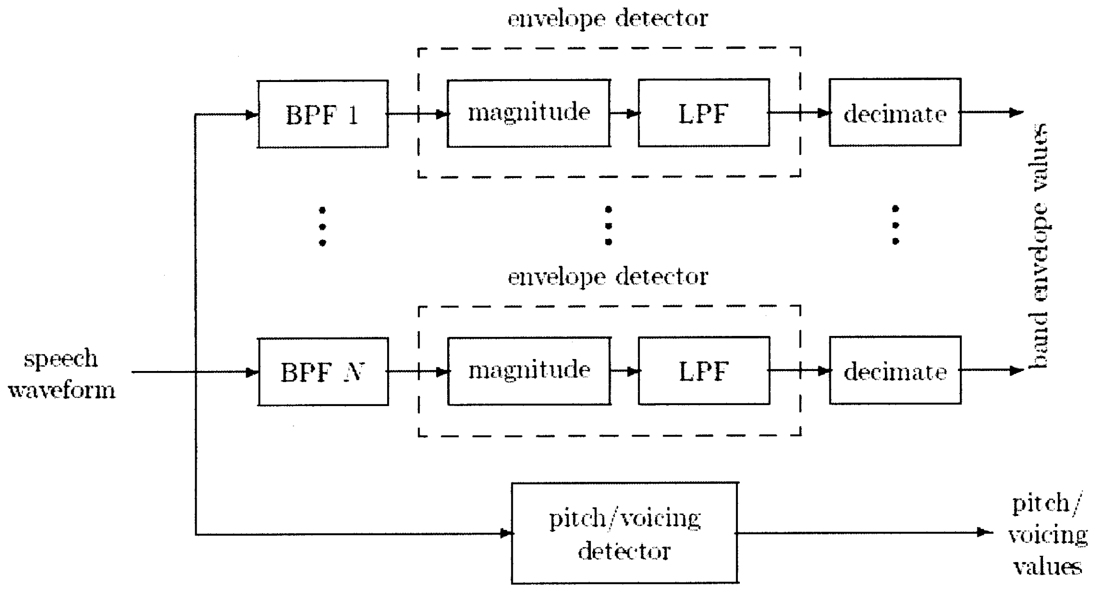


Figure 2-2: Block diagram for the analyzer for a channel vocoder. The speech is bandpass filtered and the envelope values for each frequency band are determined. These values are then decimated resulting in compression [3].

For the channel vocoder, the speech is filtered with several bandpass filters to determine the magnitude envelope of the speech signal for each frequency region. Since the rate of change of the envelope is slow compared to the sampling rate, the envelopes values can be decimated. The amount of decimation largely determines the amount of compression. The synthesizer is shown in *Figure 2-3*. It interpolates the envelope values for each frequency band and

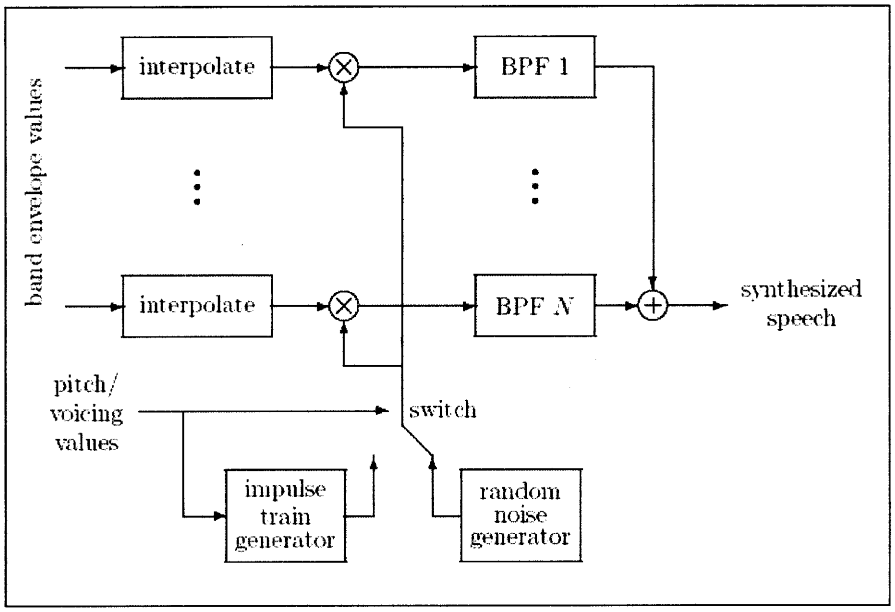


Figure 2-3: Block diagram for the channel vocoder synthesizer. First, the band envelope values are interpolated and are used to shape an impulse train or random noise. Then they are bandpass filtered and summed to create speech [3].

synthetically generates a pulse train or noise pattern depending on if the speech segment is voiced or unvoiced.

Matlab was used to assess the performance of a channel vocoder. The speech segment in *Figure 2-4* was encoded using a channel vocoder. First, the speech was broken into *30 ms* frames, or *240* samples. The bandpass filters used are shown in *Figure 2-5*. They were convolved with each frame of speech in order to determine the band envelope coefficients. In order to compress the frame, the band envelope values were decimated at various rates. The decimation rates experimented with ranged from *10* to *100*. With a decimation rate of *100* the speech quality was barely intelligible. It was determined that the minimum acceptable decimation rate for the desired speech quality was *50*.

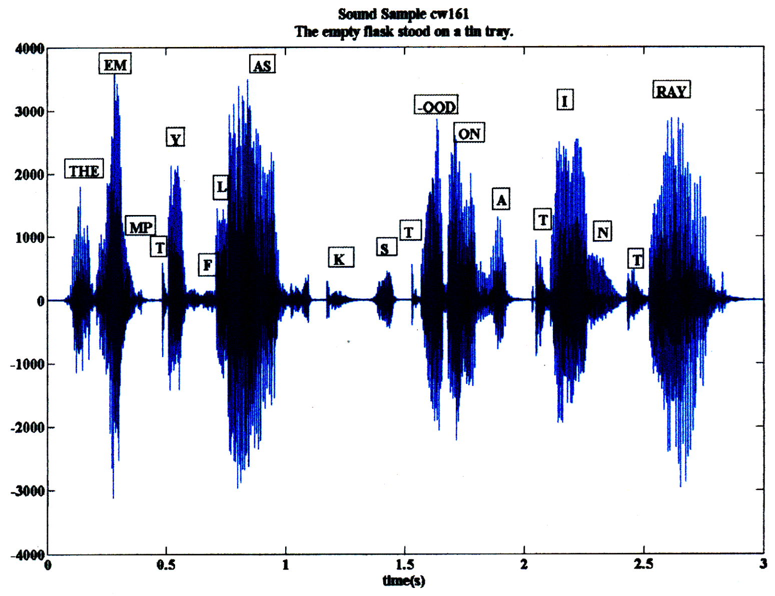


Figure 2-4: An example of a speech segment. This phrase, "The empty flask stood on a tin tray", contains most phonemes. This phrase was used to test both a channel vocoder and an autoregressive linear prediction vocoder.

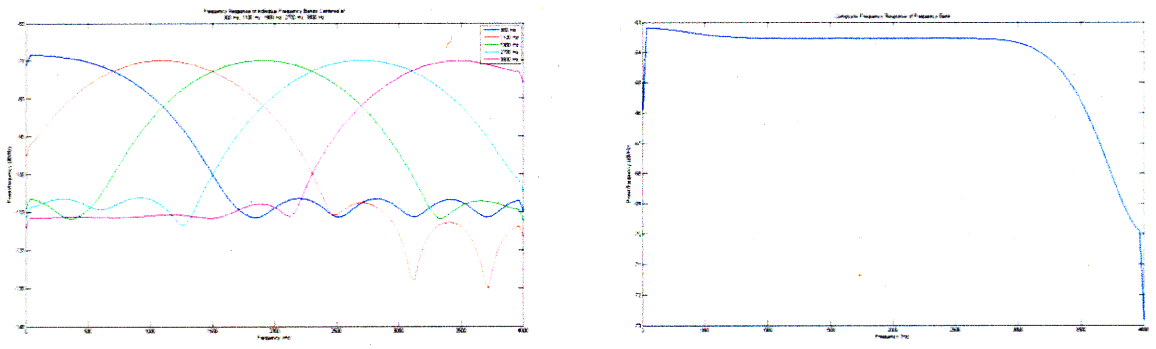


Figure 2-5: Theoretical bandpass filters for a channel vocoder. The left figure shows the frequency response of each filter while the right figure shows the composite frequency response. These filters were used to divide the 4 kHz bandwidth into 5 different frequency bands for the channel vocoder.

The original frame contained *3840 bits* (*16* * *240*), while the compressed frame consists of *872 bits* for a compression rate of *4.4*. The resolution of the filter does not have to be as fine as that of the speech samples and so only eight bits are required per sample. A filter length of *65* (for the bandpass filters) tended to produce desirable results.

$$\frac{bits}{frame} = (bands * \frac{(samples + length\,filter - 1)}{decimation} + 1) * \frac{bits}{sample}$$

The resulting data rate of *30 kbps*, while substantially lower than the original data rate, still exceeds the bandwidth of the optoelectronic fibers. In order to compress the speech segment to fit within the bandwidth of the fibers a decimation rate greater than *150* is needed and this renders the speech unintelligible. Thus different methods of source coding were explored, namely linear prediction.

2.3 Linear Prediction

Another source coding method for speech that also uses the source filter model described in the previous section is linear prediction Essentially a linear predictor fits an all-pole, or autoregressive, filter to the speech signal. The goal of a linear prediction filter is to deconvolve the source and the filter. *Figure 2-6* shows a block diagram for both the encoder and decoder of a linear predictor [3]. This linear prediction algorithm is not novel and is known as the autoregressive linear predictor and the all-pole filter is designed to minimize the total energy of the error signal, $e[n]$.

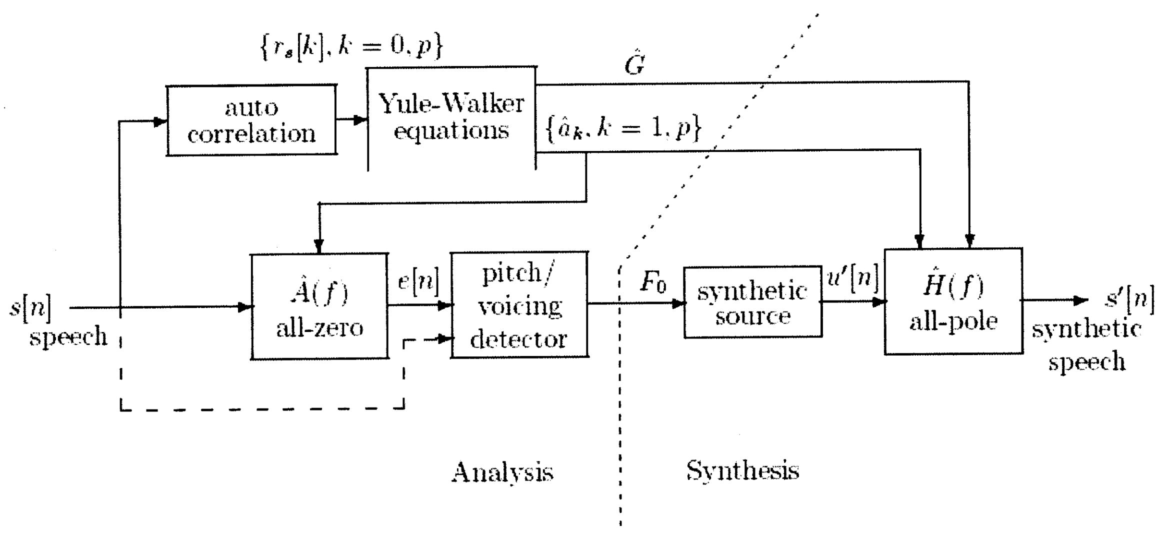


Figure 2-6: Block diagram for an autoregressive linear prediction encoder. The autocorrelation of the speech signal and Yule-Walker equations are used to determine the coefficients of an all-pole filter that minimizes the energy of the error signal [3].

The autocorrelation function defined as follows is used to generate the pole locations for the filter.

$$R_x[k] \triangleq \sum_{-\infty}^{\infty} x[n]x[n-k]$$

The speech signal is convolved with the inverse of the all-pole filter in order to obtain the error signal which is then used as the synthetic source for the all-pole filter. *Figure 2-7* shows the autocorrelation of a stressed "*a*". The number of samples between peaks can be used to determine the frequency of the pitch. The pole locations can be calculated using the linear Yule-Walker equations.

$$f_p = \frac{samples}{f_s}$$

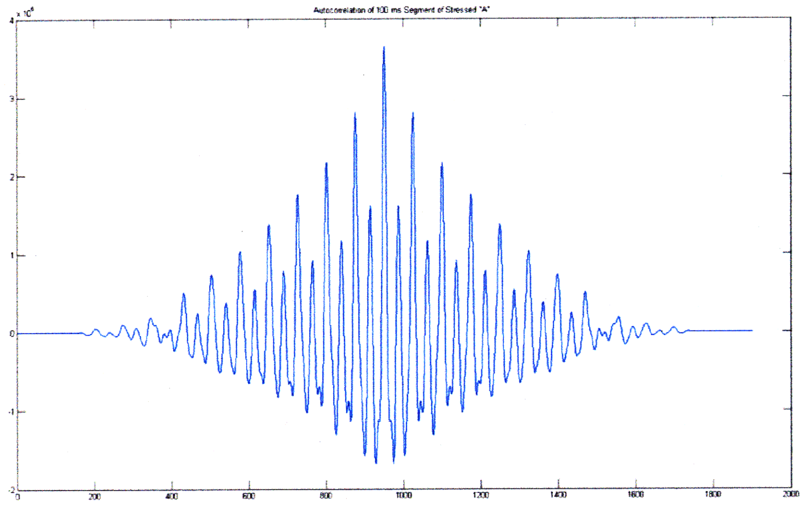


Figure 2-7: Autocorrelation of a stressed "a". The distance between peaks can be used to determine the frequency of the pitch of the speaker.

In order to see how autocorrelation produces different filters for different sounds it is best to compare the autocorrelation of a few values. *Figure 2-8* shows the autocorrelations of a stressed "*a*" and a stressed "*e*". Even though these two vowels sound similar, their respective autocorrelations clearly show the differences. Thus, the autocorrelation encoder can detect and encode subtle differences between sounds.

The model order (number of poles) is largely responsible for determining the bit rate of the compressed signal. For speech a *14th* order system provides acceptable results. If *12 bits* are used to encode each pole location, pitch value, and gain then the minimum acceptable bit rate for this encoder is *9.6 kbps* (assuming *20 ms* frames). So a linear prediction encoder was chosen to perform the source coding for this optical system.

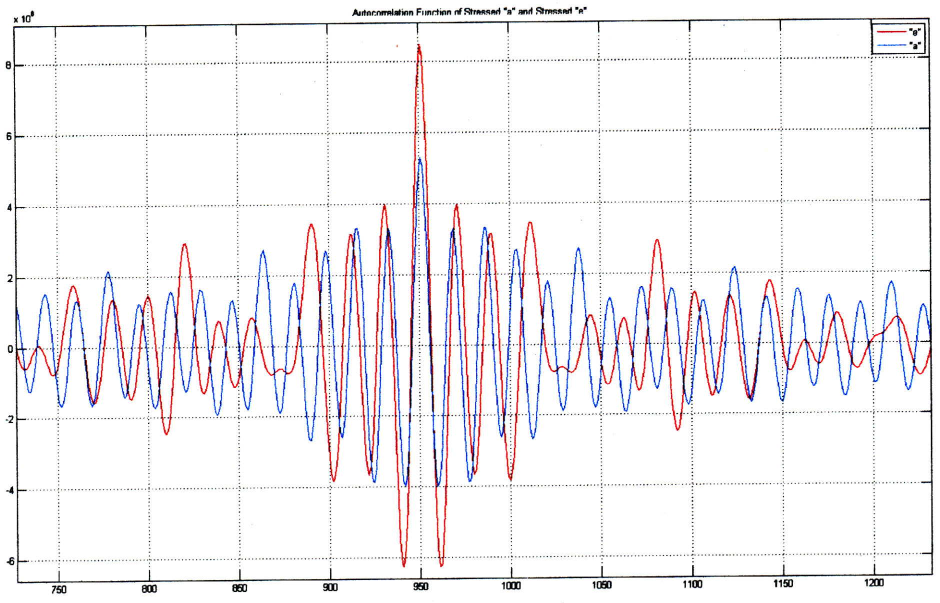


Figure 2-8: Autocorrelation of a stressed "a" and a stressed "e" vowel. The first few peaks are in approximately the same location indicating that the pitches are consistent for each vowel. However, there is clearly a difference between the spoken vowels as the autocorrelation indicates.

2.4 Speex – Code Excited Linear Prediction (CELP)

Even though the linear predictor designed in the previous section would be a feasible encoder for the optoelectronic fibers, implementing it on a DSP is a non-trivial task. Matlab offers several utilities for calculating the autocorrelation and linear prediction coefficients of a data vector, Libraries written in C and optimized for a DSP are not as available. Since both vocoders are not recent developments, several freeware vocoders exist that are based on linear prediction. One such project, Speex, was eventually used as the vocoder for this communication system.

The Speex vocoder uses a code excited variant of linear prediction (CELP). Instead of using the error signal to excite the filter, this vocoder uses a fixed codebook to generate the synthetic source [4]. *Figure 2-8* shows the modified block diagram for the CELP encoder. Each frame is broken into subframes and a closed-loop search is used to determine each codebook gain. The Speex libraries were originally designed for Linux-based systems, however they recently introduced a version optimized for Microchip processors.

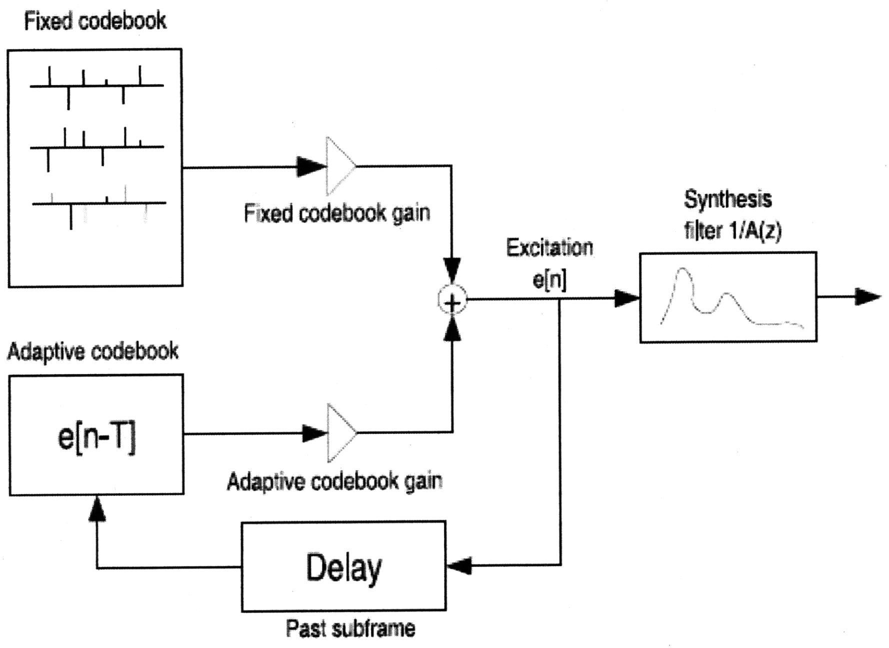


Figure 2-9: Block diagram for Speex's CELP vocoder. The vocoder uses both a fixed and an adaptive codebook to generate the source for the synthesis filter [4].

The Speex vocoder for Microchip operates as follows [4]:

- Allocate memory for encoder and decoder
- Allocate shared scratch memory for encoder and decoder
- Initialize encoder for audio channel
- Initialize decoder for audio channel
- Encode a raw speech data frame
- Decode an encoded speech data frame

The narrowband mode for Speex compresses the speech signal to a rate of *8 kbps*. The input to the Speex encoder is a frame of speech that is *20 ms* long, and consists of *160* samples. The encoder then outputs a vector of *20 bytes*. In order to produce intelligible sound quality both the encoder and decoder must be called every *20 ms*. This library was used because of its high-level applicability. The user needs to only supply input and output buffers for both the encoder and decoder. Thus, the source coding for this project ultimately ended up being a call to the Speex encoder. Please refer to the Speex manual for additional details and specifications [4].

Chapter 3

Channel Coding for Optical Communication

3.1 Introduction

Most infrared data communication systems are designed for high-bandwidth short-distance communication [5]. Several different types of channel coding methods have been proposed and designed to meet the challenges of optical communication. Some have looked at multi-pulse encoding methods [6], while others have explored the pulse modulation techniques [7]. In order to design the optimal communication system it is necessary to first characterize the optical channel [8].

This section describes the channel coding techniques used for this optical communication system. Channel capacity calculations were done with the photodiode system, but are readily applicable to the optoelectronic fibers. Since the Speex encoder is robust against dropped packets, but is very susceptible to corrupted packets, it was necessary to minimize the probability of packet error. Additional overhead was added to minimize errors and the universal asynchronous receiver transmitter (UART) protocol was adopted. Finally, to maximize the range of link, the LED's were pulsed which reduced the signal-to-noise ratio.

3.2 Character Framing

The packetized encoding scheme for this optical communication system borrows from the successes of RF communication and universal asynchronous receiver/transmitter (UART) systems. Asynchronous communication protocols must be designed so that symbols can be detected in noisy conditions without a clock signal. Shown in *Figure 3-1* is the standard UART encoding scheme. Flags are bit sized and signal the beginning and end of the symbol. A logical one (start flag) is used to indicate the start of a symbol, while a logical zero (stop flag) is used to verify that all bits of the symbol have been received and provides a buffer between symbols. While this encoding requires additional overhead (*10 bits* are used to encode *8 bits* which leads to a *25%* increase in the minimum bit rate), it also enables symbol detection without a clock.

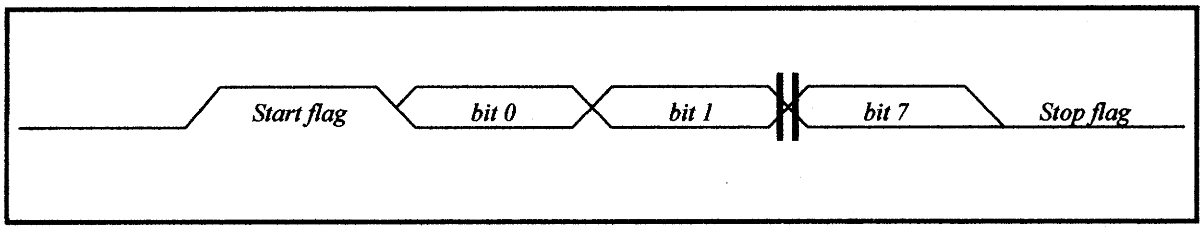


Figure 3-1: UART encoding scheme. A symbol is indicated with both a start and stop flag. Errors occur when the receiver misses a start flag. Since practically every symbol contains a logical one, the receiver mistakes a data bit for a start flag when it misses the original start flag. The stop flag is one method of correcting this. If a stop flag is not correctly received, then the receiver determines that there was a framing error and discards the received symbol.

3.3 Packet Framing

Bit framing is a good method for symbol detection when there is a low probability of a missed start flag and the loss of a single symbol is inconsequential (either the transmitter can retransmit the symbol or the receiver can process the data without that symbol). However, in this communication system, the loss of a single symbol renders the entire packet unusable, and since there is not enough time for retransmission, it is imperative that the channel encoding minimizes the probability or errors. Thus, more overhead is both needed and tolerated for this system. Since only compressed speech data is transmitted across the optical channel, fixed length packet and character based framing can also be used. *Figure 3-2* shows packet composition – two bytes header, twenty bytes speech data, and two byte cyclic redundancy check (CRC). In a noisy environment, such as that for free-space optical communication, it is necessary to add additional redundancy in order to make communication less susceptible to errors.

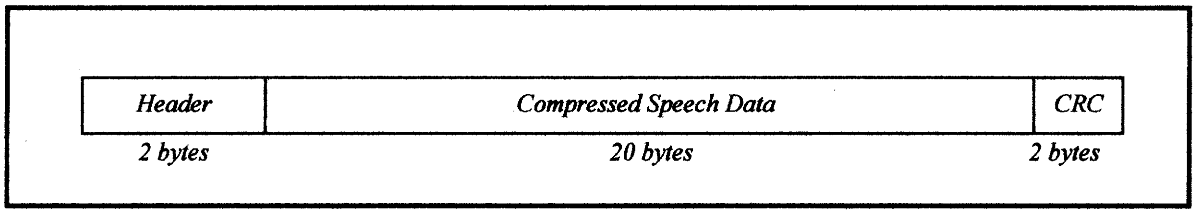


Figure 3-2: Fixed speech packet composition. A two byte header is used as a flag to signal the beginning of the packet. This is followed by twenty bytes of compressed speech data. Finally a 16-bit (two byte) CRC is appended to the packet. The receiver compares the sent CRC to the calculated CRC in order to determine if a valid packet was sent.

3.4 Optical Channel Characterization

Most RF communication systems use a header in order to synchronize the transmitter and the receiver for each packet sent. The header is essentially a long flag, consisting of a set of predefined bytes that must be received before the receiver begins to save data on the optical channel. Careful selection of the packet flag (or header) is essential to system performance. This flag should not be susceptible to the type of errors present on the optical channel. There are three main types of errors on the optical channel – random, sporadic high intensity optical noise, bursts of high intensity optical interference, and finally blocking of the high intensity transmitted signal. This channel can be modeled as a binary (but not symmetric) channel and this is shown in *Figure 3-3*.

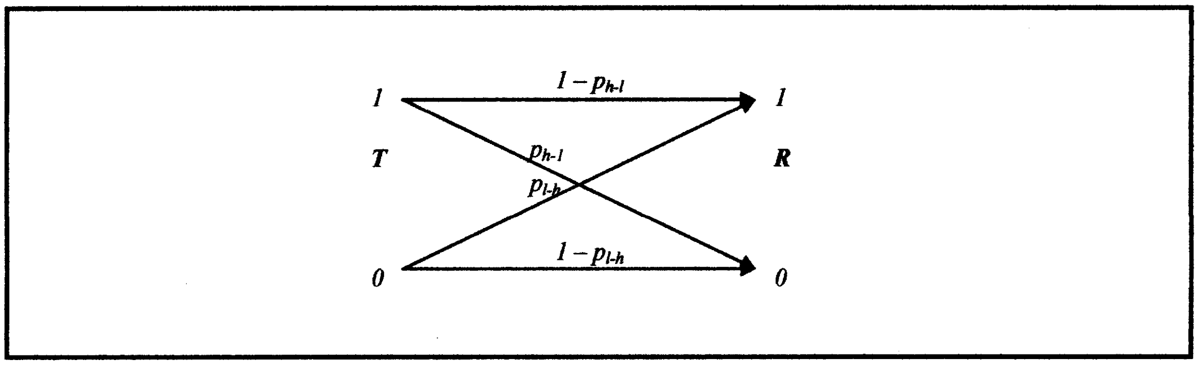


Figure 3-3: Binary channel representation for a discrete, memoryless optical channel. A symmetric representation is inaccurate because low-to-high errors ($p_{l\text{-}h}$) are much more probable than high-to-low errors. This is due to the fact that almost all of the optical noise is additive (sunlight, other infrared sources, etc.). Occasionally the transmitted signal will be blocked resulting in a high-to-low error. In the figure, T represents the bit sent by the transmitted, while, R represents the bit received by the receiver.

The optical receiver designed and implemented by Agustya Mehta blocks almost all DC noise present on the optical channel. This allows the system to be used in infrared-rich operating conditions, such as daylight. More importantly, from an encoding perspective, it practically eliminates burst and blocking errors and from a modeling standpoint the binary channel assumes the form in *Figure 3-4*.

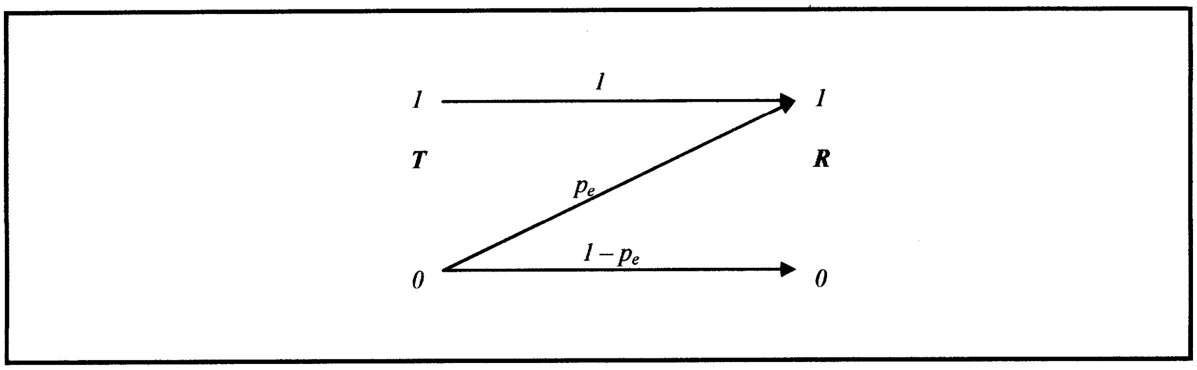


Figure 3-4: Revised model for binary optical channel. This model will be used for entropy and channel capacity calculations. Since burst errors occur in blocks, only random errors at 1 MHz frequency are detected, and p_e expresses the probability of these random errors. Conservative estimates based on measurements of this optical channel show that p_e is on the order of 10^{-3}.

Channel capacity is defined by the amount of mutual information between the receiving and transmitting ends of the channel. This can easily be calculated using entropy as follows:

$$C = \max I(T;R)$$

$$C = \max\big(H(R) - H(R|T)\big)$$

Where entropy is defined as:

$$H(R) \triangleq \sum_i p_i * \log\frac{1}{p_i}$$

Expressions for the probability of each value (a zero or a one) are shown in the following equations. Notice, as expected that the probability of receiving a one is equal to one minus the probability of receiving a zero.

$$p_{R0} = p_{T0} * (1 - p_e)$$

$$p_{R1} = p_{T0} * p_e + (1 - p_{T0})$$

The resulting expression for the entropy at the receiver is shown in the next equation. Since channel capacity increases as receiver entropy increases, the maximum channel capacity is achieved when receiver entropy is maximized which occurs when both values are equally probable.

$$H(R) = p_{R0} * \log\left(\frac{1}{p_{R0}}\right) + p_{R1} * \log\left(\frac{1}{p_{R1}}\right)$$

$$p_{R0} = p_{R1} = \frac{1}{2}$$

The maximum entropy approaches one, which is its maximum value, when the probability of sending a zero takes the following value.

$$p_{T0} = \frac{1}{2(1-p_e)}$$

$$H(R) = 1$$

Next, an expression for the conditional entropy of the receiver is needed. This can be expressed as the summation of conditional entropy given each transmitted value multiplied by the probability of that transmitted value. Since if a one is transmitted, only a one can be received, the conditional entropy given a transmitted one is zero. Equations for conditional entropy are shown below.

$$H(R|T) = P(T=0) * H(R|T=0) + P(T=1) * H(R|T=1)$$

$$H(R|T) = p_{T0} * \left(p_e * \log_2\left(\frac{1}{p_{T0}p_e}\right) + (1-p_e) * \log_2\left(\frac{1}{p_{T0}(1-p_e)}\right)\right)$$

When the channel is active, $p_{T0} \sim 0.5$, but when the channel is idle $p_{T0} = 1$. During continuous operation assume the channel is active *75%* of the time.

$$E[p_{T0}] = 1 - 0.5 * active\ rate = 0.625$$

And the channel capacity, C, is *0.4743 bits* per transmission. The receiver is bandwidth limited, W, to approximately *2 MHz*. The maximum bit rate is expressed as follows:

$$R_b = 2 * W * C = 1.74\ Mbps$$

This is substantially greater than the maximum achievable bit rate of the optoelectronic fibers developed by the Photonic Bandgap and Devices Group. Because of this, the speech compression algorithm described in the previous section is more than adequate for the current photodiode setup.

3.5 Packet Synchronization Flag

If only a single bit is used to synchronize packets, the probability of a false synchronization is equal to the probability of a random error. Since false synchronization errors are extremely costly (once a synchronization flag is received, the receiver begins to save all twenty speech bytes), it is necessary to minimize the probability of a synchronization error. By increasing the size of the flag (header), and using alternating values, the rate of false synchronizations can be substantially reduced. The length of the header requires a design tradeoff – the longer the header, the less likely there will be a synchronization error; however, the overhead (and therefore required minimum bit rate) also increases. *Figure 3-5* shows part of the header for this communication protocol and the next equations show the relationship between the probability of synchronization error, p_{se}, increase in bit rate, R_b, and header length (in bytes), l.

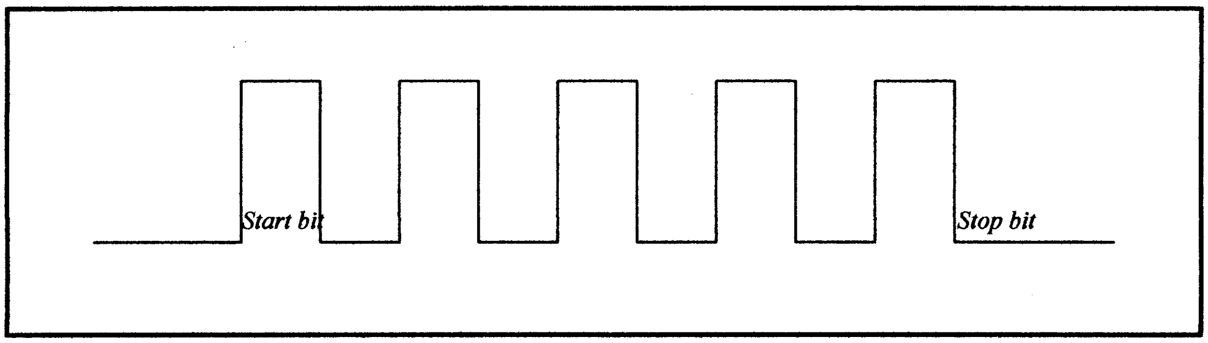


Figure 3-5: Example header composition of length 1. Essentially, the header creates a square wave that the receiver is able to "lock" on to. Traditional flags for wired connections consist of the byte 01111110 whereas this header uses 1010101010. An alternating header was chosen because it is easier for the receiver to detect, and because it minimizes the type of errors typical on the optical channel.

$$p_{se} \sim (p_e)^{5l}$$

$$\%\,increase\ in\ R_b = \frac{1}{20}l$$

Thus, a header of length two reduces the probability of a synchronization error to the order of approximately 10^{-30}, which is essentially zero. This leads to an acceptable ten percent increase in the bit rate.

3.6 Error Detection

The aforementioned encoding provides protection against flag errors for packet and symbol detection. However, with this scheme it is impossible to determine if valid data has been received. Since the speech compression algorithm is susceptible to corrupted packets (but is fairly immune to dropped packets), the receiver must ensure the packet is valid before passing on for speech decompression. One method for error detection and correction is a cyclic redundancy check (CRC). The CRC used for this application has been standardized by the Comité

Consultatif International Téléphonique et Télégraphique (CCITT). The CCITT sets international communication standards and the CCITT-16 generator string is commonly used in many WAN, IrDA, and Bluetooth communication systems. The following equations show the generator string, G. In order to compute the CRC, the data, D, is divided by the generator string, G, and the remainder is appended to the data stream such that the transmitted data, D_T, is a multiple of G.

$$G = x^{16} + x^{12} + x^5 + 1$$

$$R = D \, mod \, G$$

$$D_T = [D \; R]$$

A CRC increases the minimum Hamming distance (HD) and can detect up to (HD – 1) errors in the packet. The standardized CRC described above has a Hamming distance of four and so up to three random errors can be detected (but not corrected). The probability of detecting an error as a function of packet length, Hamming distance, and error rate is expressed as follows. This probability increases as the packet length decreases, error rate decreases, or Hamming distance increases.

$$P_{ed} = \sum_{k=0}^{HD-1} \binom{n}{k} (p_e)^k (1 - p_e)^{n-k}$$

For a *160 byte* packet with channel error probability on the order of 10^{-3}, the selected CRC can detect over *99.99%* of all random errors. Adding the CRC also increases the minimum bit rate by inserting an initial twenty bits into the packet. The code rate of this communication protocol, defined as the ratio of data bits to transmitted bits is shown below.

$$R_C = \frac{D_b}{T_b} = \frac{160}{240} = 0.667$$

In order to understand this code rate, it is necessary to compare it to the code rates for different types of encoding methods. Another common type of encoder is a block encoder. Symbols of length, k, are mapped to a larger block, n. A (7, 4) Hamming Code is an older block code that is able to detect up to two errors (comparable to the CRC code used above), but has a code rate of *0.571*. Thus, the proposed encoding method is not only more capable of detecting errors, but is also more efficient.

3.7 Data Rate

Even though the maximum data rate for this optical system is on the order of *4 MHz*, it would be best to minimize the data rate since this system will eventually need to use bandwidth-limited optical fibers. Each packet contains *20 ms* of speech samples and so the packet must be encoded, transmitted, received, and decoded in less time. The next equation shows how to calculate the minimum acceptable bit rate in order to meet these criteria.

$$t_p = \frac{packet\ size}{minimum\ bit\ rate} \leq 0.02$$

Each packet contains *240 bits* of data and so the minimum bit rate must be greater than *12 kbps*. In order to provide sufficient processing time for encoding and decoding the packet the channel should only be active for about *75 percent* of the time. Therefore, a data rate of *14.5 kbps* was used for this system. Another factor that was considered when selecting the data rate is the visibility of the overall system. Since the head-mounted LED’s are visible with infrared cameras, the LED's should be on as little as possible.

3.8 Energy Content of Encoded Signals

The energy content of a baseband signal is expressed in the following equation. This is a good method for determining the amount of energy in an electromagnetic wave and is used for energy calculations for RF systems.

$$E = \int_0^T (S(t))^2 dt = A^2 T$$

The amount of optical power for any given LED is a function of the input current and is typically expressed in terms of radiant intensity (W/sr). Datasheets for infrared LED's contain a figure illustrating the relationship between input current and radiant intensity. *Figure 3-6* shows the normalized radiant intensity as a function of input current for Osram's SFH 4550 *850 nm* infrared emitter.

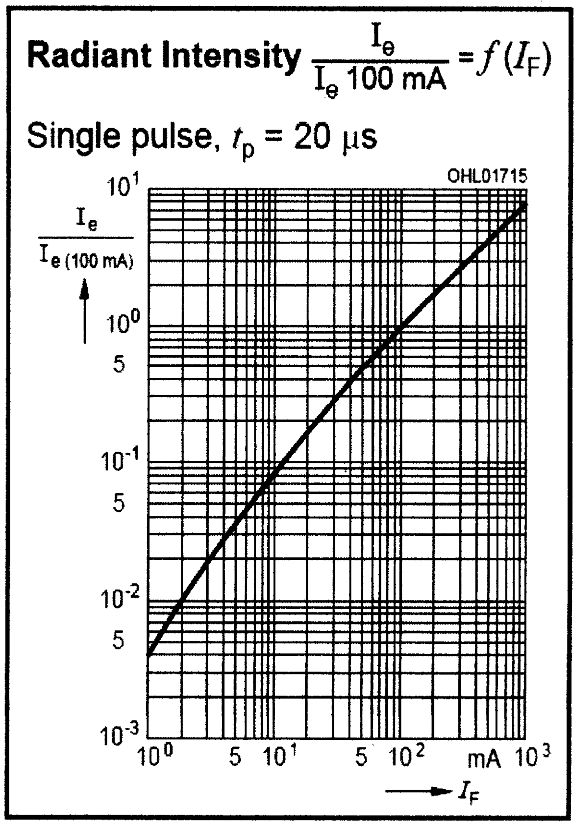


Figure 3-6: Normalized radiant intensity as a function of input current for OSRAM's SFH 4550 850 nm infrared emitter. As input current increases, the radiant intensity also increases. Thus, for maximum range the largest input current possible should be used. The maximum possible input current is approximately one ampere and this value was used for system design [9].

When the LED's are pulsed, they can withstand a current around one ampere which leads to a radiant intensity that is approximately nine times greater than the radiant intensity when the input current is one hundred milliamps. The signal to noise ratio of a data link is described as follows and depends on the power content of the transmitted signal.

$$SNR = 10\log_{10}\frac{P_s}{P_n}$$

Assuming that the optical noise injected on the receiver is additive white Gaussian noise (AWGN), the power of the noise is constant across all frequencies. By using the pulsed transmission scheme the signal-to-noise ratio increases by more than *9 dB* as shown below.

$$\Delta SNR = 10\log_{10}\frac{P_{s-p}}{P_s} = 9.5\ dB$$

Thus, pulsing the signal allows communication at a longer distance because of the increased signal-to-noise ratio. Another benefit of the pulsed signal is that it the overall LED energy consumption decreases. The ratio of pulsed signal's energy consumption per bit to the original signal's energy consumption per bit is expressed in the next equation. The pulsed method consumes significantly less energy while providing a higher signal-to-noise ratio and so it is the optimal communication method.

$$\frac{E_{s-p}}{E_s} = \frac{P_{s-p}f_s}{P_s f_{s-p}} = \frac{9 * 14285}{1000000} = 0.13\%$$

3.9 Modulation Encoding Techniques

Another common encoding technique used in RF communication systems is sinusoidal modulation. This type of system is shown in *Figure 3-7* and is often used because of bandwidth constraints and to avoid interference from competing electromagnetic signals. It also shows the Fourier transform associated with a modulated signal.

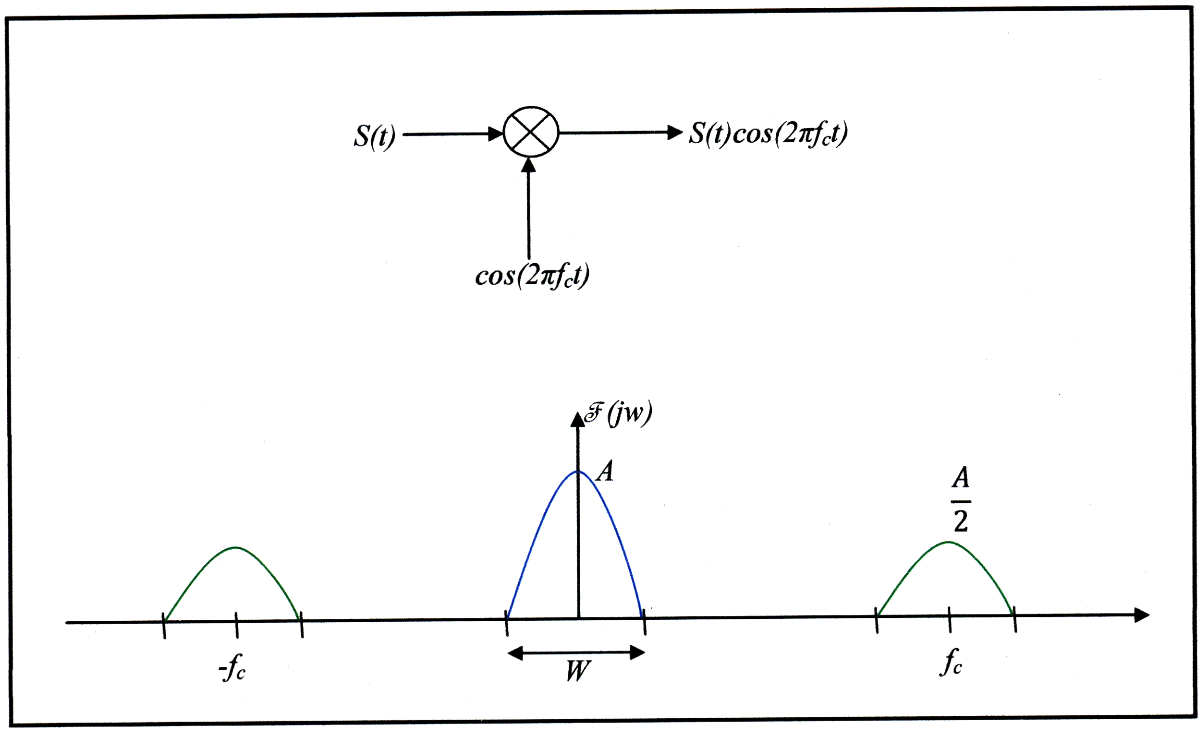


Figure 3-7: Generation of a sinusoidal modulation and its associated Fourier transform. By changing the carrier frequency the baseband signal can be shifted to a occupy a different section of the frequency spectrum and reduce interference. This also reduces the average energy required per bit.

The obvious benefit of this technique is that it prevents interference by shifting the baseband signal. But this optical communication system does not have to compete with other signals (except mostly DC ambient noise) and so modulation provides no additional benefit. From an energy perspective modulation is also beneficial. Looking back at the Fourier transform of the modulated signal, the energy content of the modulated signal is half that of the original baseband signal. Thus, the modulated signal could be used to further reduce the energy requirements of the optical link. However, the carrier frequency of the modulated signal needs to be substantially higher than the signal bandwidth (at least two twice as large). In order to modulate the *1 MHz* pulsed signal it would require a carrier frequency on the order of *10 MHz.* The silicon photodiode used in this system can detect a *10 MHz* signal, but this is drastically outside of the range of the optoelectronic fibers. Note, the *1 MHz* pulsed signal is still acceptable for the fibers because fibers only need to detect the edge, but not the entire pulse. Even though the modulated signal would be more energy efficient, it is less bandwidth efficient. In anticipation of integrating the low bandwidth optoelectronic fibers, modulation techniques were not used.

Chapter 4

System Implementation and Integration

4.1 Introduction

In order to make the optical communication system as discrete as possible it was necessary to select the proper components. This section describes the hardware and software implementation of the source and channel coding theory described in the previous sections. Several versions of the processing unit were designed in order to minimize its obtrusiveness. Also, an additional power unit was added to the system to help miniaturize the processing board. Once the processing unit was designed and thoroughly tested work began on integrating it with the transmitting, receiving and power boards developed by the other members of the electronics team.

4.2 Hardware Implementation

Figure 4-1 shows a block diagram for the hardware for this system. There are two main blocks of circuitry – the audio processing components and the components used for routing the digital signals to and from the transmitters and receivers, respectively. The audio components are responsible for amplifying and filtering the microphone input and headphone output. Central to this is a codec which also handles the conversion between digital and analog signals. The digital

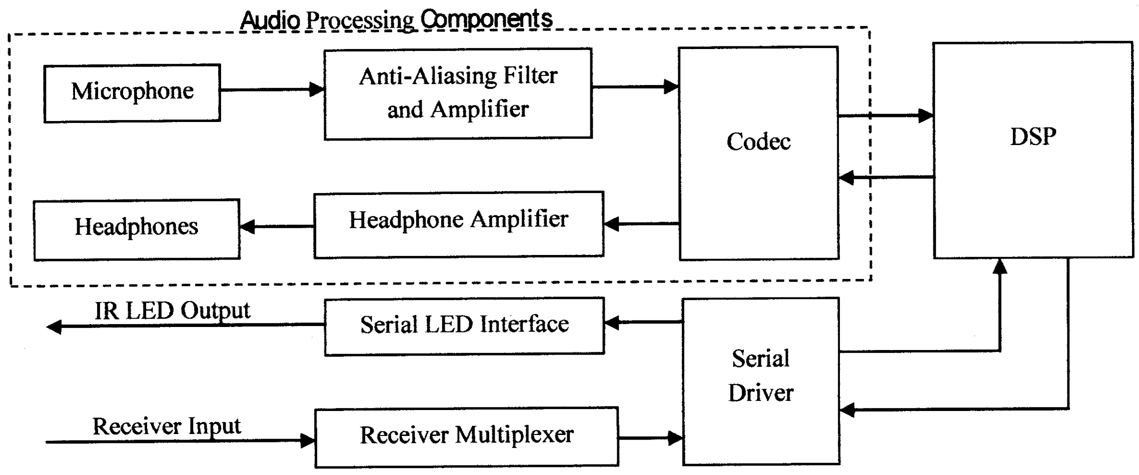


Figure 4-1: System hardware block diagram. There are two main modules – the audio processing components and the interface between the receivers and transmitters. Central o the board is a DSP which is responsible for nearly all signal processing.

inputs and outputs for the receivers and transmitters are all directly connected to the DSP via a serial driver.

4.2.1 Microcontroller

After careful research on current microcontroller technology, a digital signal processor (DSP) was chosen to anchor the processing unit. DSP's are specifically designed and optimized for the intensive mathematical operations associated with signal processing. Microchip (www.microchip.com) offers several DSP evaluation boards that are designed to be used in speech and audio applications. The fast instruction clock, coupled with large on-chip memory made the *dsPIC33FJ256GP710* ideal for this system. One such evaluation board for the dsPIC was purchased and is shown in *Figure 4-2*. The Speex encoder library described in Chapter 2 was loaded on the board in order to assess the compressed speech quality. Once the ISN approved the sound quality a compact processing unit was designed.

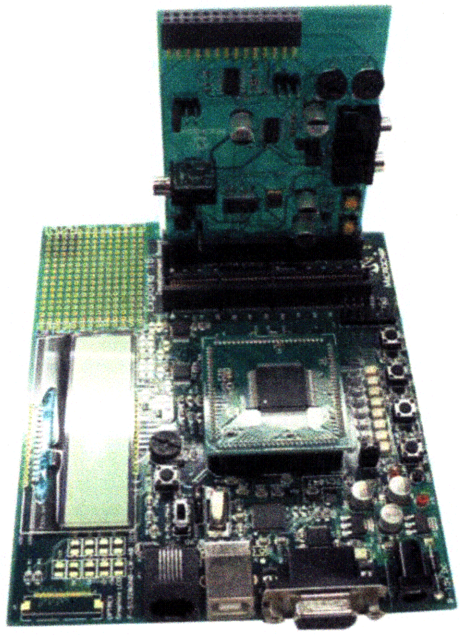


Figure 4-2: dsPIC33FJ256GP710 evaluation board available directly from Microchip. This board was used to assess the speech quality of the Speex library and to determine if optical communication was viable with Microchip's DSP's.

4.2.2 Audio Processing

Human speech is still recognizable when it is band-limited to approximately *4 kHz* and so the signal only needs to be sampled at a rate of *8 kHz*. In order to prevent aliasing of the high frequencies, a low-pass filter is necessary with a cutoff around *4 kHz*. An active RC filter was chosen to perform this task as it allows for simultaneous amplification of the audio signal. *Figure 4-3* shows the filter. The potentiometer is used to control the microphone gain and cutoff frequency of the filter. Early versions of the board had a second stage low-pass filter to ensure that there was no aliasing. This second stage was removed in subsequent versions because it threw away too much gain.

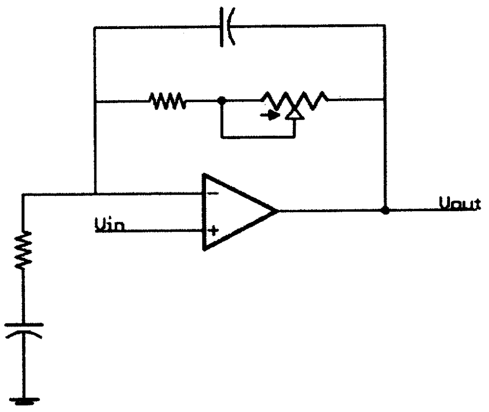


Figure 4-3: Microphone amplifier and low-pass filter. An active RC filter was used to prevent aliasing of high frequencies. The potentiometer in the feedback loop can be used to adjust the gain and cutoff frequency of the amplifier.

Next, the analog audio signal must be converted to a digital signal for the DSP to process. The dsPIC contains several *10-bit* analog-to-digital converters (ADC's) but these were not used for several reasons. First, commercial audio codecs exist that have ADC's with greater resolution (*16-bit*) and additional signal processing capabilities. Second, even though the DSP has a much faster instruction cycle than the PIC used in the previous system, the DSP processing power should be devoted to speech compression and packet assembly rather than ADC conversion.

Wolfson's mono codec, WM8510, was used to convert the analog audio signal into a digital input for the DSP to process. This codec was chosen because it has several pre and post-processing features. All of these features were selected using an I^2C connection between the DSP and the codec. The first feature used was a digital high-pass filter. This filter has a cutoff frequency around *100 Hz* and eliminates any potential *60 Hz* buzz on the microphone input. The codec can also provide up to *6 dB* boost of the microphone input, allowing for greater range of volume control. Even though the codec is capable of driving a *16 ohm* or *32 ohm* headset without any additional buffers or amplifiers, an additional headphone amplifier was added. This amplifier, the LM4811 can provide an additional gain of *12 dB*. It was chosen because the gain settings can be controlled via a two-wire serial interface with the DSP. By setting the gain to its highest setting, the volume can be controlled exclusively by the external potentiometer. This allows the system board to be completely embedded in the vest without any external controls.

4.2.3 Transmitter and Receiver Interface

Each transmitter is connected to the system board via a five-wire interface. This consists of the pulse bit stream, the line to select an LED to use, a clock signal, and finally power and ground lines. Since the transmitter requires purely digital signals the pins of the DSP can be used to directly drive the transmitters. The original design for the transmitters used a parallel interface – each LED was controlled by a different pin on the DSP. This was cumbersome as over fourteen wires were needed to connect the transmitter. This also added additional hardware – a LAN switch was used to switch between each LED signal. The board associated with the original parallel transmitter interface is shown in *Figure 4-4*.

The processing board must also be able to accept signals from several different receivers. Since the built-in UART receiver is used for reception, all receiver inputs must be connected to the same pin on the DSP. In order to prevent interference from other receiver channels, a multiplexor is used to select a receiver channel. Thus, only one receiver is connected to the UART module at same pin on the DSP. In order to prevent interference from other receiver channels, a multiplexor is used to select a receiver channel. Thus, only one receiver is connected to the UART module at

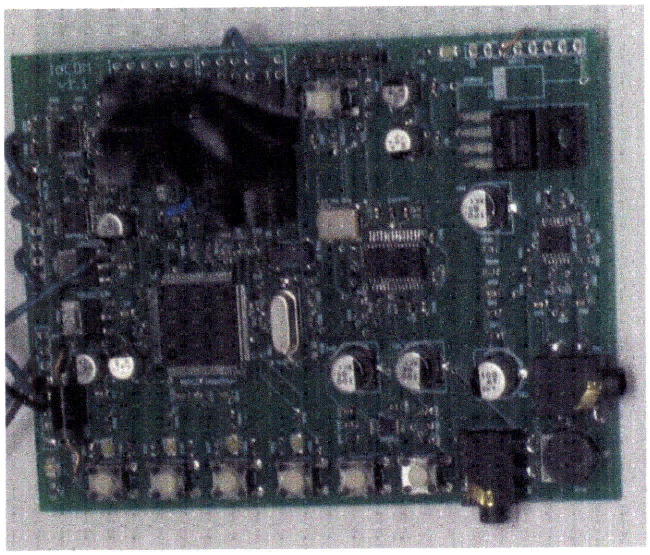


Figure 4-4: Original system board design. The board measured 3" by 4" and contained several test LED's and switches. The parallel interface for the transmitters (covered in electrical tape) was later changed to a serial connection.

any given time. The DSP then must be able to cycle between receiver channels to ensure omni-directional coverage.

4.2.4 Final Hardware Design

The major modification made to the board shown above was converting the parallel interface to the transmitters to a serial interface. This change, coupled with the movement of the voltage regulators to a separate board allowed for the reduction of the existing unit. All of the buttons were also removed as they were only for testing purposes. Once the system functioned properly these switches were no longer necessary. *Figure 4-5* shows the top and bottom of the final processing unit. This board measures *2.5"* by *2.5"* and comfortably fits inside the back of the vest. It was later wrapped with heat shrink for protection, thus all connections and indicator LED's needed to be located on the edge of the board.

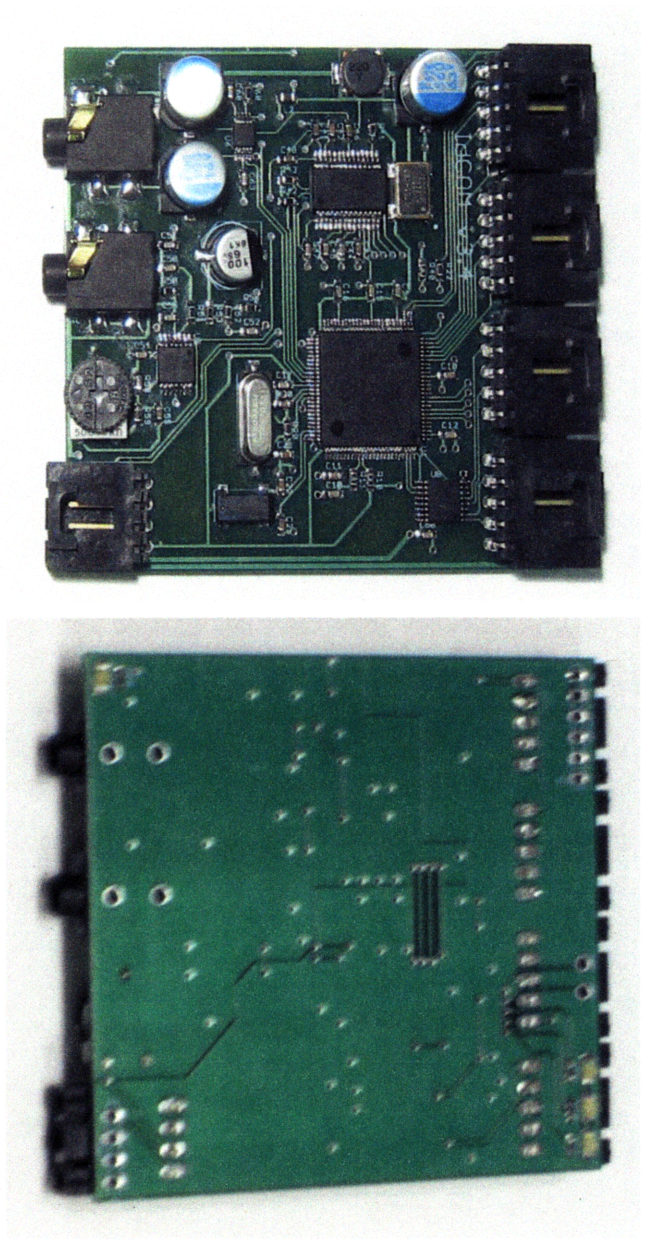


Figure 4-5: Top and bottom of final system board. The board is now a compact 2.5" by 2.5" with standardized Molex connectors for the transmitters and receivers. On the bottom of the board an analog and digital ground plane was added to prevent parasitic inductance and capacitance.

4.3 Software Implementation

All of the software was coded in C using Microchip's MPLAB IDE. Please see Appendix B for select code segments. A block diagram and overview of the software implementation is shown in *Figure 4-6*. The main program continuously cycles through this loop, first encoding and transmitting a packet before trying to decode a valid packet. The aforementioned Speex library is used to encode and decode the speech signals. It takes in *20 ms* frames of speech (which is *160* samples), and compresses it to *160 bytes*, which is a compression rate of *16*.

When transmitting a packet, a timer is initialized to interrupt at the bit rate (not character rate). This interrupt service routine (ISR) is responsible for pulsing the LED's. It can be in several states, both within a byte and within the packet. As the communication protocol in chapter three describes, first a synchronization byte is sent before the data packet and finally CRC are sent. Once the ISR sends the entire packet it disables the timer until the next packet is ready to be sent. The start and stop bits for UART transmission are appended to each byte within this ISR.

The UART module on the dsPIC is setup to interrupt after a character has been received. The receiver processes this character depending on its current state. Once a valid header has been received, the receiver begins storing the data into a temporary buffer. When all of the data has been received a CRC is computed and is compared to the received CRC. If there is a match then the packet is sent to the decoder. Since the decoder is susceptible to corrupted packets, if there is a CRC error then the packet is discarded and null data is sent to the decoder.

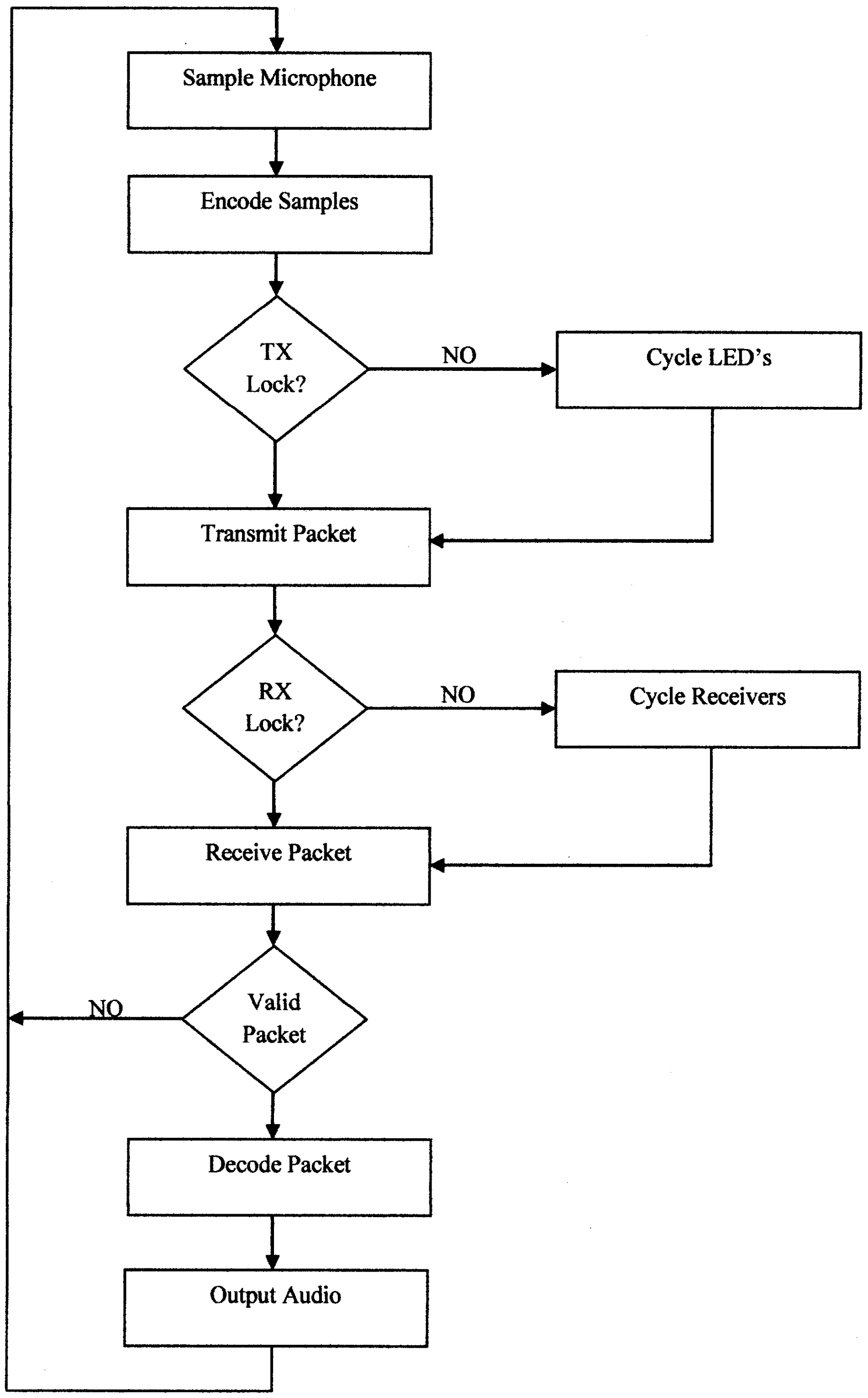


Figure 4-6: Software block diagram. The system repeatedly executes the same loop. Because of integration problems this loop was modified so that the system only transmitted when voice was present on the microphone.

If the receiver loses its connection with the transmitter in the middle of a packet it has the potential to hang. This would stall the program until the remainder of the packet is received. In order to prevent this, several checkpoint were inserted in the code. If reception was not completed in *300 ms*, which is more than adequate, then receiver was reset and the execution of the remainder of the loop continued.

Since multiple receive and transmit channels are used it is necessary for the processing unit to switch between them in an appropriate manner. On the receiving side, the system searches for a valid header on one of the receivers. If a header is not detected within a predetermined time frame then the next channel is checked. The scanning frequency of channels was approximately *5 Hz* to ensure enough time to detect a packet on each channel. Ultimately LED switching for the transmitters was not implemented. Synchronization between the boards was lost too easily and the time that it took to resynchronize them was too costly. Instead, all of the LED's on both the front and back transmitters were pulsed when a packet was transmitted.

4.4. System Integration

Once work was complete on the processing unit it was integrated with the transmitters and receivers as shown in *Figure 4-7*. Subsequently several problems ensued. The first problem was that the large pulses on the transmitter magnetically coupled with the photodiodes on the receivers. Even though the receivers had an electrostatic shield, they were still susceptible to magnetic interference. Because magnetic shielding is difficult and the proper materials were not readily available, this problem was solved with a software modification of the processing unit. When a unit was transmitting it essentially rendered the receiver useless because of the magnetic coupling. Therefore, the unit should only transmit or receive, but never both simultaneously. A simple voice detection algorithm was implemented to determine if voice was present on the microphone input. If voice was present, then the unit transmitted data. If not, then it attempted to receive a valid packet.

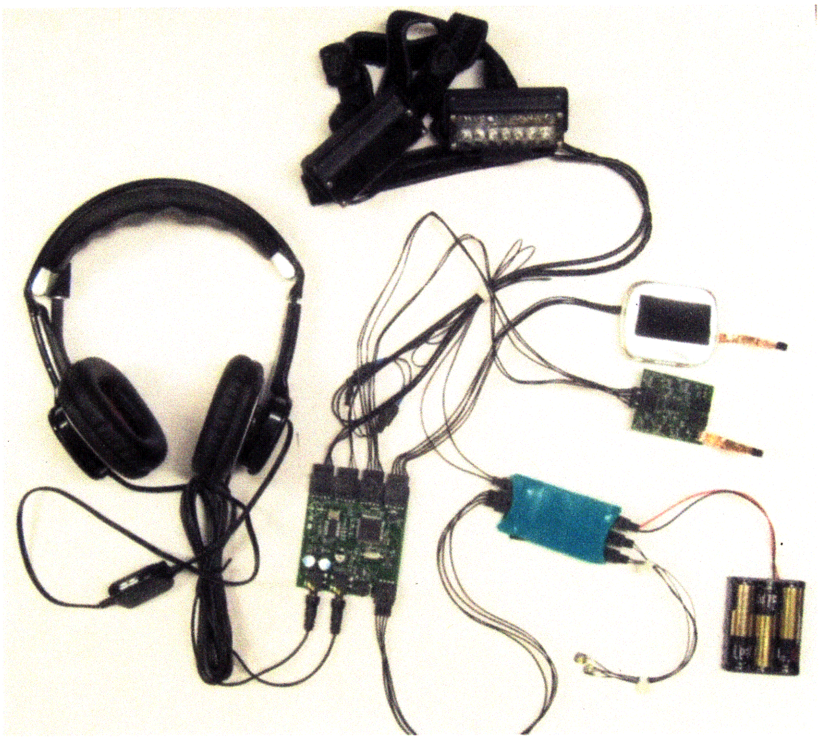


Figure 4-7: Complete system. This consisted of a system board, two transmitters, two receivers, power unit, headset and battery pack. All power and signals were routed through the system board.

Because of some noise on the microphone, the voice detector used the power of the signal to determine if voice was present. In order to prevent overflowing and to speed up computation only the twelve most significant bits were used for voice detection. If the power content of the signal exceeded the power threshold for the frame, then data was transmitted for the next half second (*500 ms*).

Once the system was tested work began on embedding it into the vest. This proved to be a more difficult task than originally anticipated because most of the epoxies would not adhere to the water resistant fabric on the vest. The boards were wrapped in heat shrink to protect the components and Velcro was glued to the back of the boards. This ensured that the boards would not be permanently attached to the vest. Matching strips of Velcro were glued to the vest and the corners were sewn to make sure that the strips would not move. An image of the complete vest on Agustya Mehta is shown in *Figure 4-8*. All of the components except for the transmitter and headset are discretely concealed within the vest.

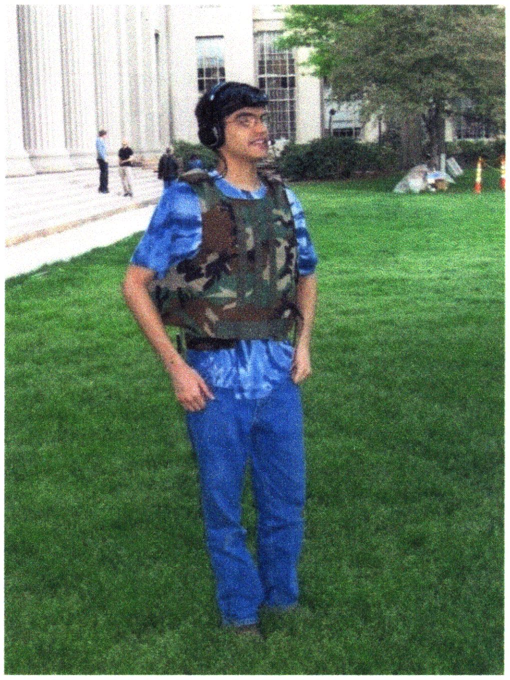


Figure 4-8: Agustya Mehta wearing the vest. The transmitter rests on the forehead allowing the user to effectively aim his speech. All other components are embedded inside the vest.

The system was tested in daylight to ensure that it could function even with ambient light flooding the receiver. *Figure 4-9* shows Professor Fink and Agustya using the system at a range of approximately *20 m*. The system was also tested indoors to make sure that it did not suffer

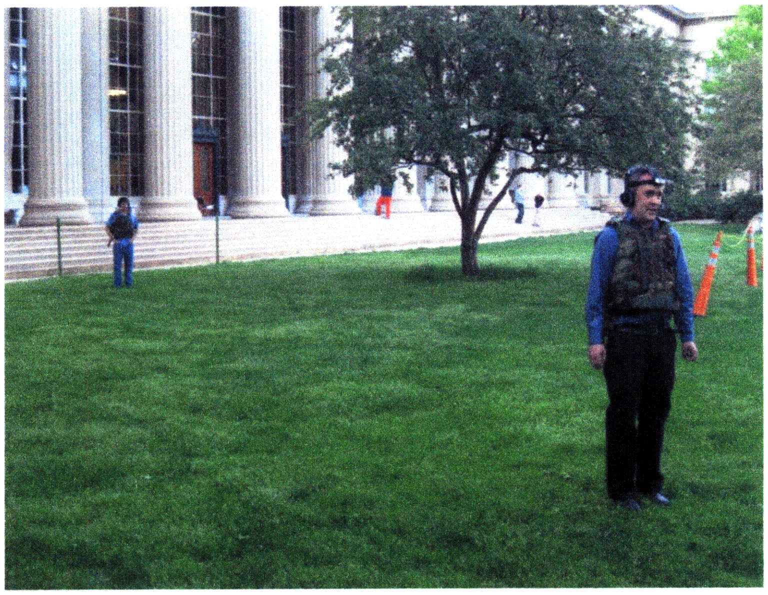


Figure 4-9: Professor Fink and Agustya performing a test of the system. This shows that front-to-back and back-to-front communication is possible at a range of approximately 20 meters.

from multipath interference as RF communication does. Range was actually further indoors, approximately *25 m*, mostly likely because there was not ambient light interference. Finally, the system was tested under night conditions to make sure that the LED's were not visible and to determine if the aiming capabilities of the system were still as intuitive when it is difficult to see.

Chapter 5

Conclusion and Future Work

This thesis designed and implemented a communication protocol designed specifically for a band-limited optical channel. First, two types of speech vocoders were examined, a channel vocoder and a linear prediction vocoder. The channel vocoder, though easy to implement did not provide the speech compression necessary to be used with the optoelectronic fibers and so a linear predictive vocoder was simulated using Matlab. Implementing a linear predictive vocoder on a DSP proved to be a non-trivial task and so the Speech CELP library was used. This allowed speech compression to a rate of *8 kbps*.

Next, a channel coding algorithm was developed to maximize the signal-to-noise ratio on the optical channel. This algorithm was modeled after the UART coding scheme with start and stop bits used to frame characters. A *16 bit* CRC was appended to the end of the packet for error detection and correction. Finally, the bit stream was pulsed in order to maximize the signal-to-noise ratio and range of the system.

Once the system board was completed it was integrated with the transmitters and receivers developed by the other members of the group. The entire system was placed in a standard military vest and tested under several operating conditions to ensure that the system would work in a hostile environment.

This system did not, however, use any optoelectronic fibers. The author of this thesis strongly recommends that the next step be to replace the photodiodes with fibers as they become available. After that there are several directions that research can take.

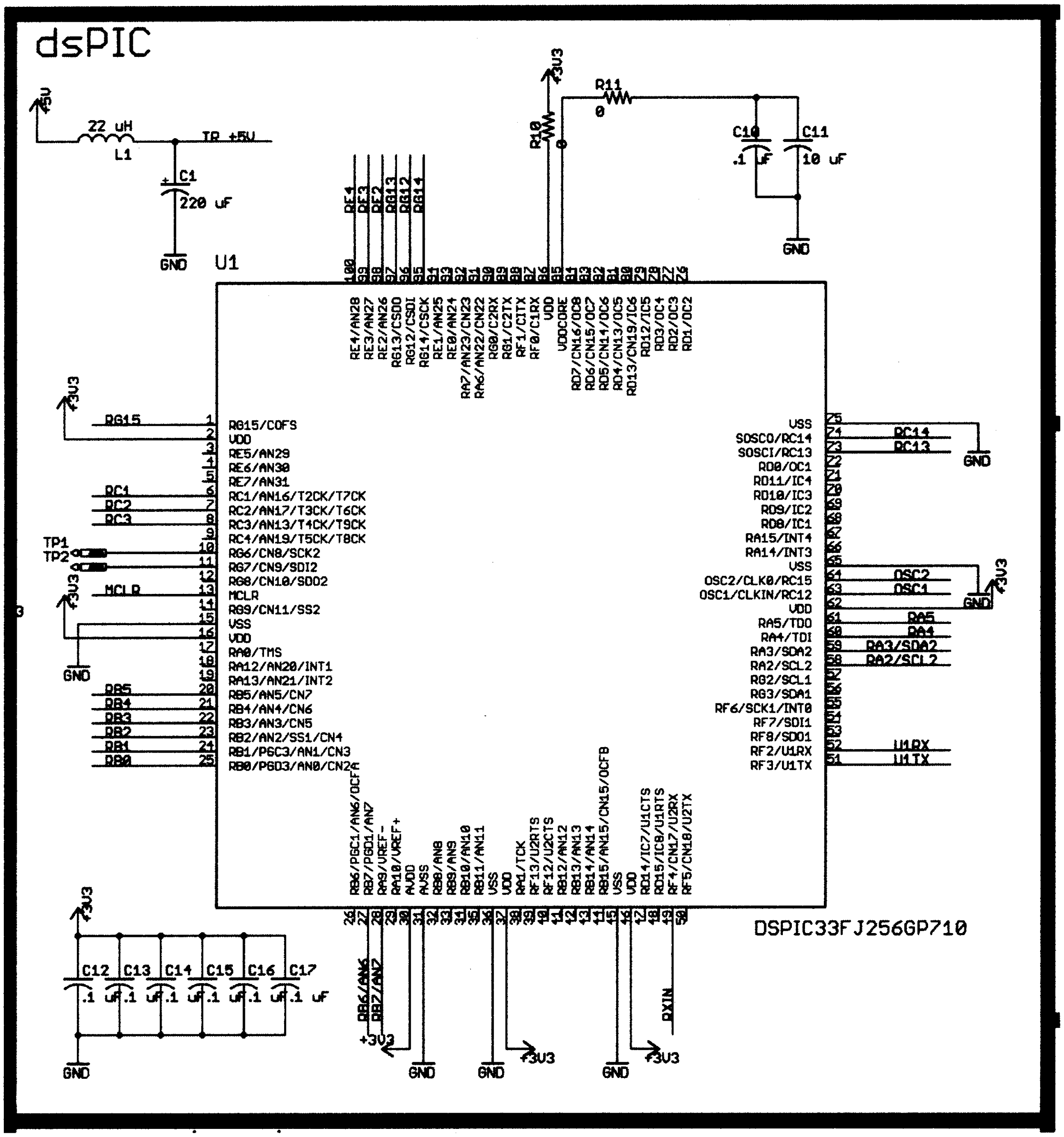
First, the range of the system needs to be improved. Ideally soldiers should be able to communicate at a range ten times farther (*200 meters*). In order to accomplish this the channel coding techniques need to be reexamined. Other options would be to purchase higher power LED's and more sensitive operational amplifiers.

Next, the transmitter must be able to switch between LED's. When all LED's are on the speaker is very visible with an infrared viewer. This is not acceptable if the SOF wants to maintain a discrete profile. A synchronizing function should be added to the system code. This function should synchronize the boards approximately every second (or *5* packets). This is a non-trivial task, as an efficient object tracking algorithm will need to be designed to meet this requirement.

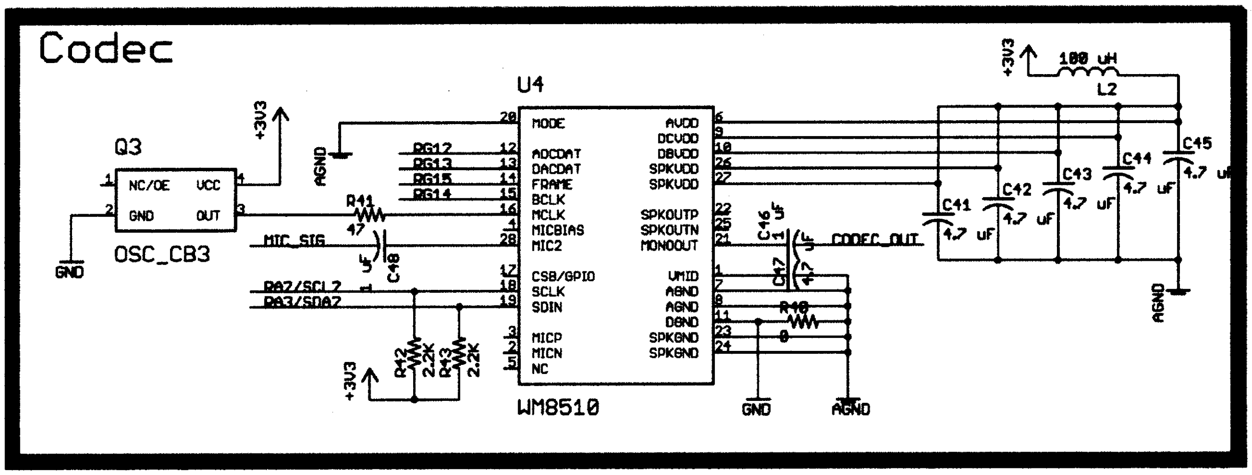
Finally, the system does not have the friendly force identification circuitry that previous versions did. Since the original proposed system had both a communication and identification feature this should also be incorporated into the digital communication system.

Appendix A

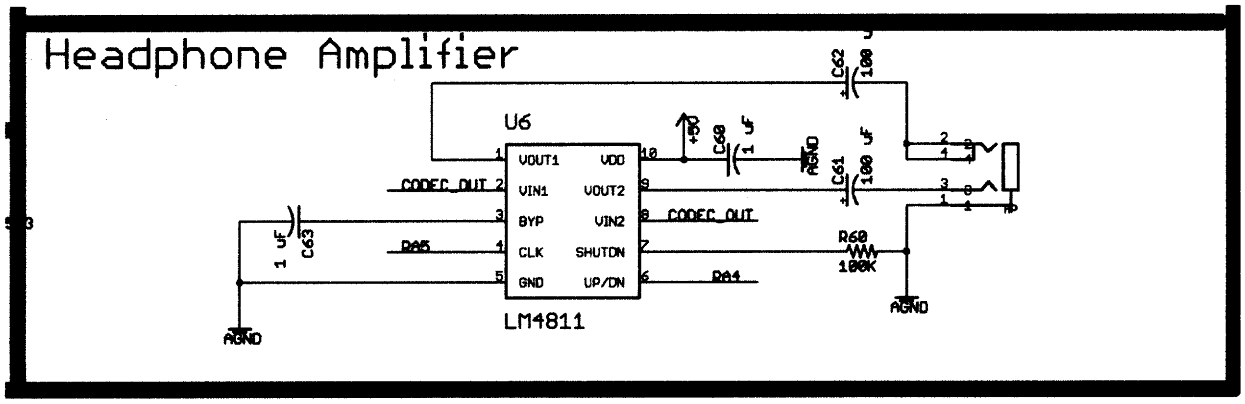
Schematics



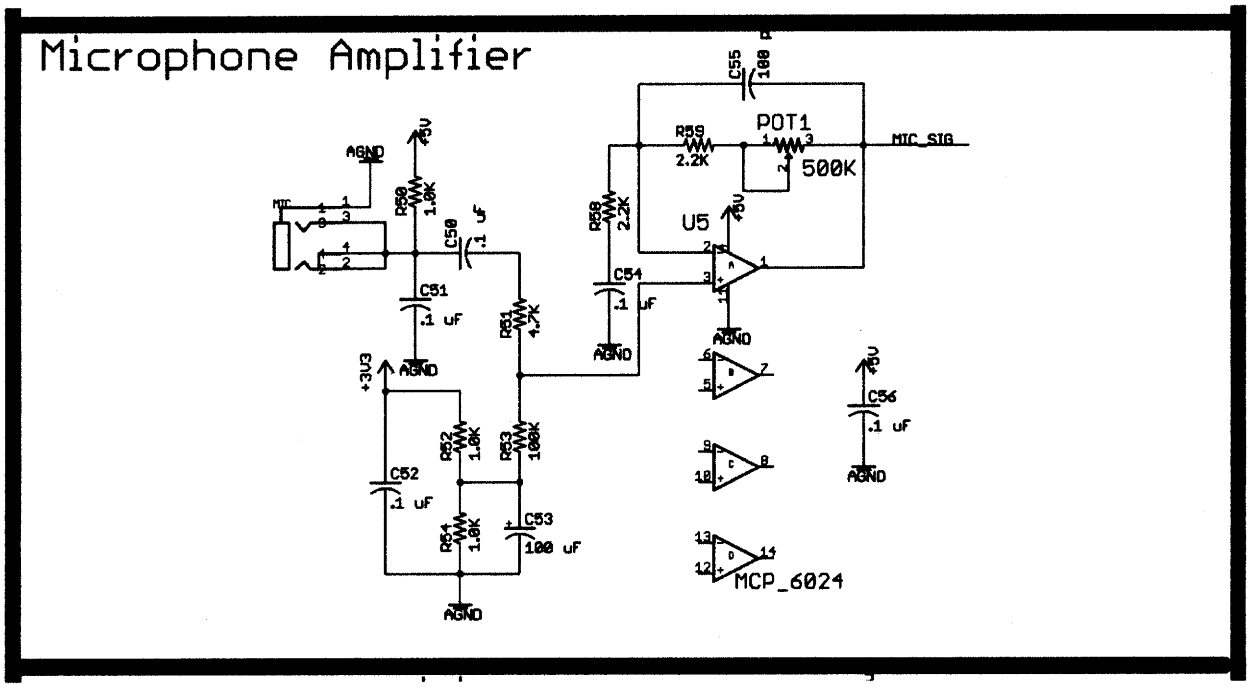
A-1: Schematic for dsPIC33FJ256GP710 connections



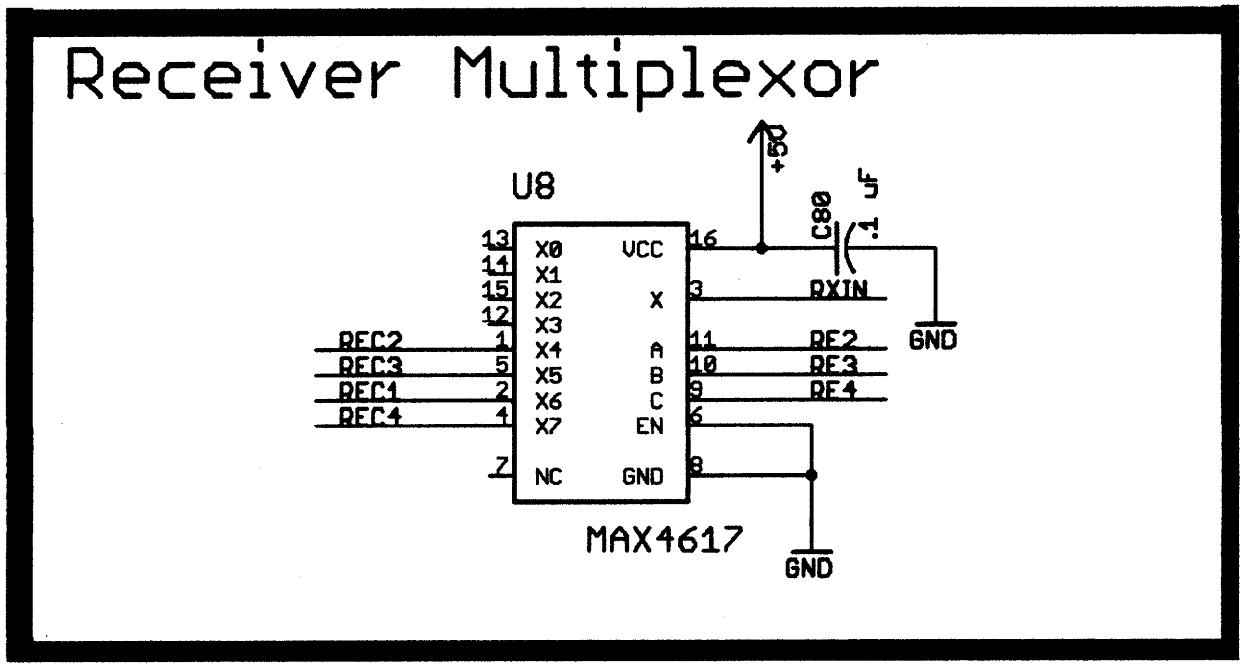
A-2: Schematic for WM8510 codec connections.



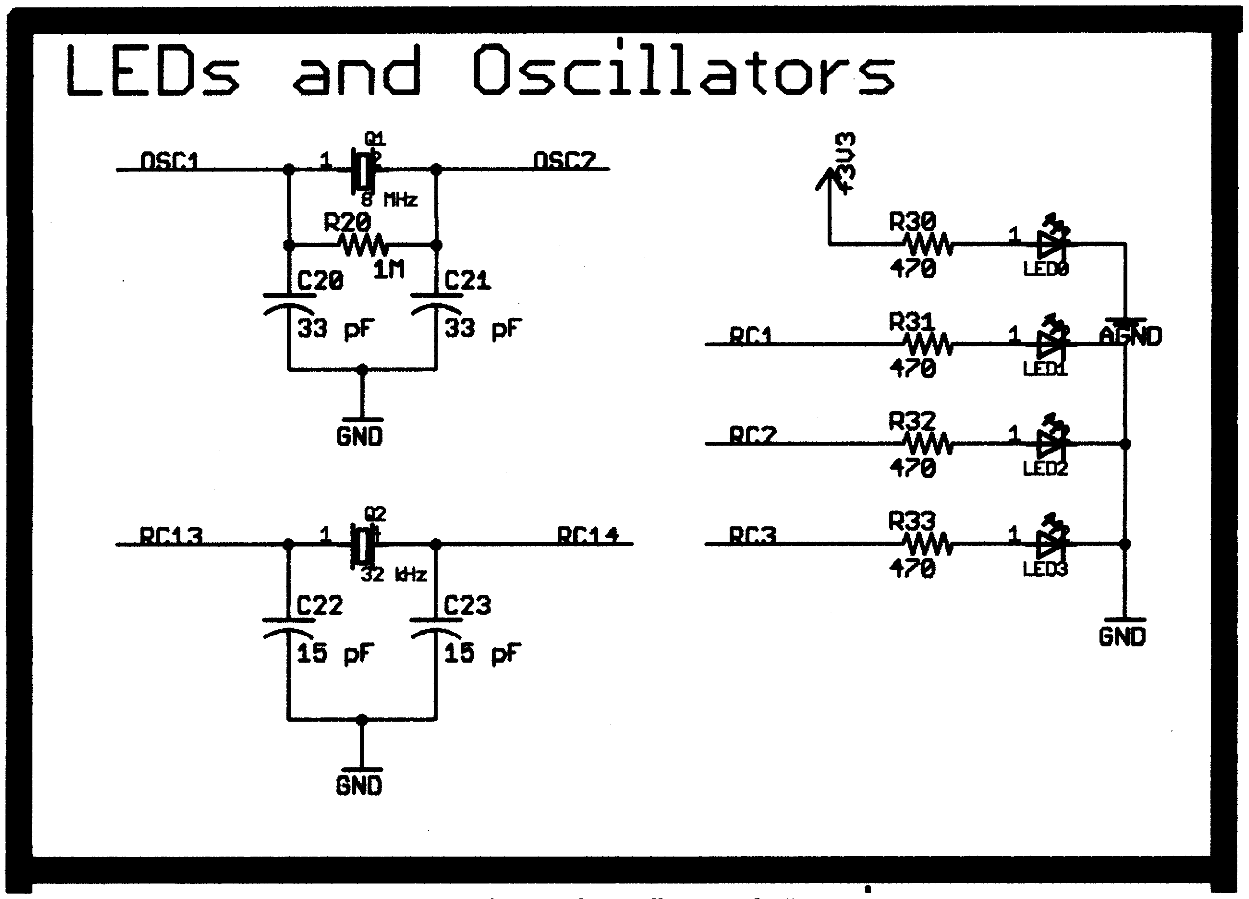
A-3: Schematic for LM4811 headphone amplifier connections.



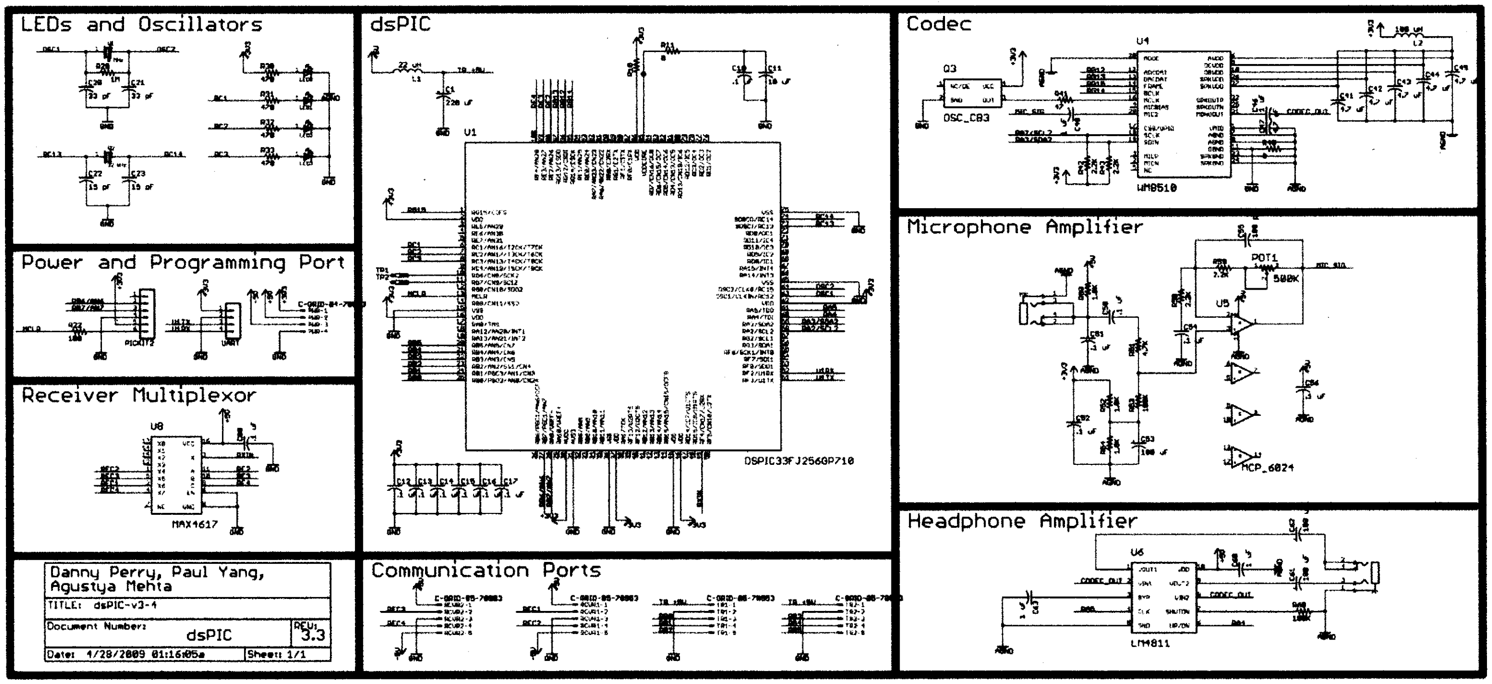
A-4: Schematic for MCP6024 connections (microphone amplifier and filter).



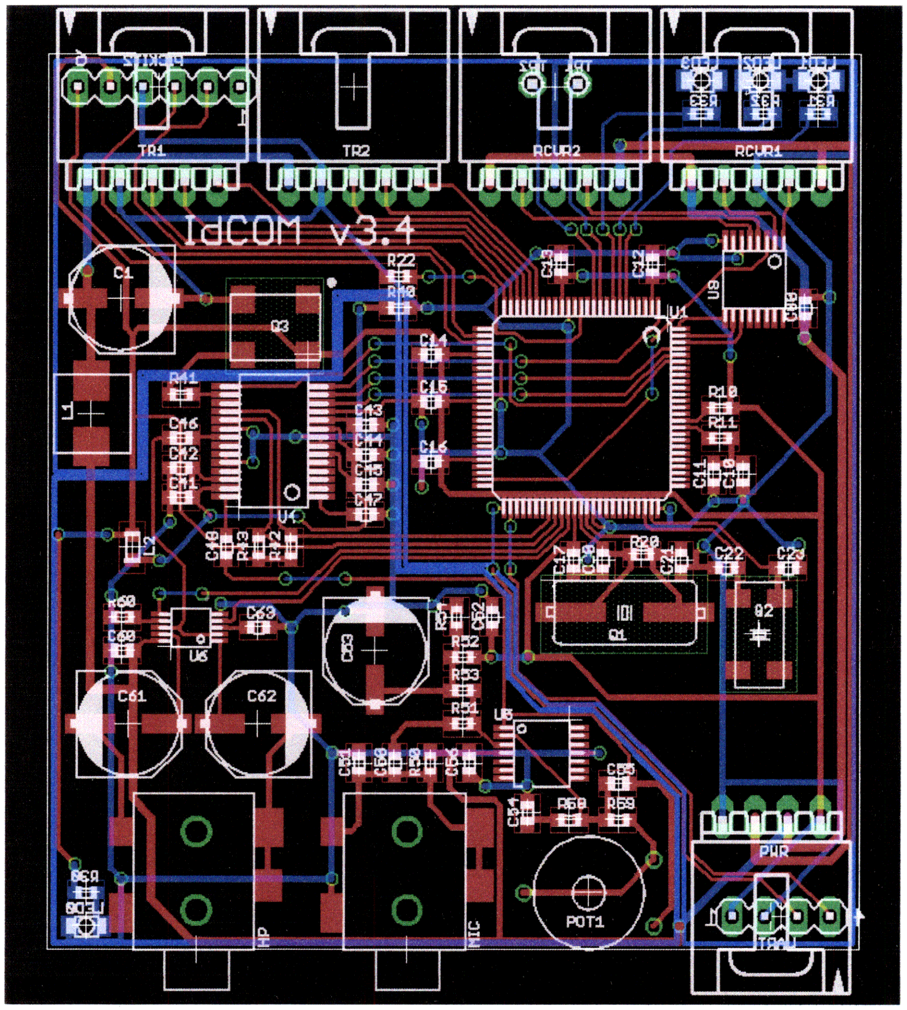
A-5: Schematic for MAX4617 multiplexor connections



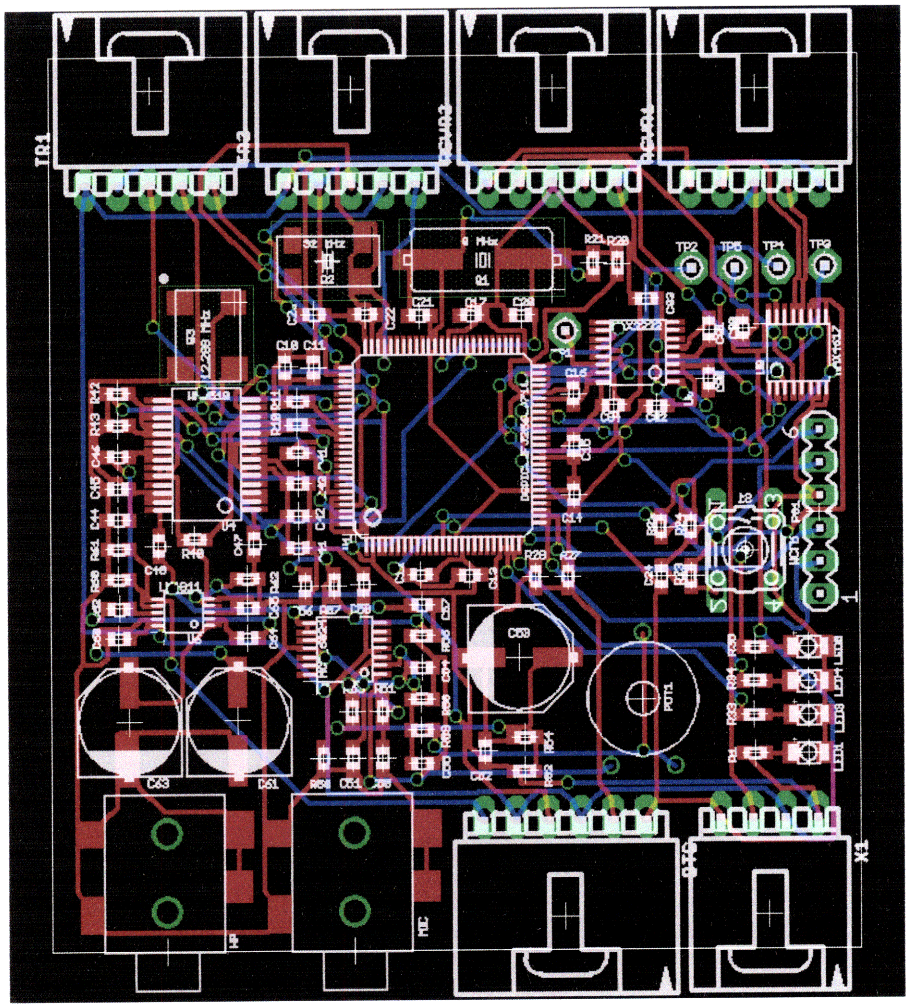
A-6: Schematic for oscillators and LED's



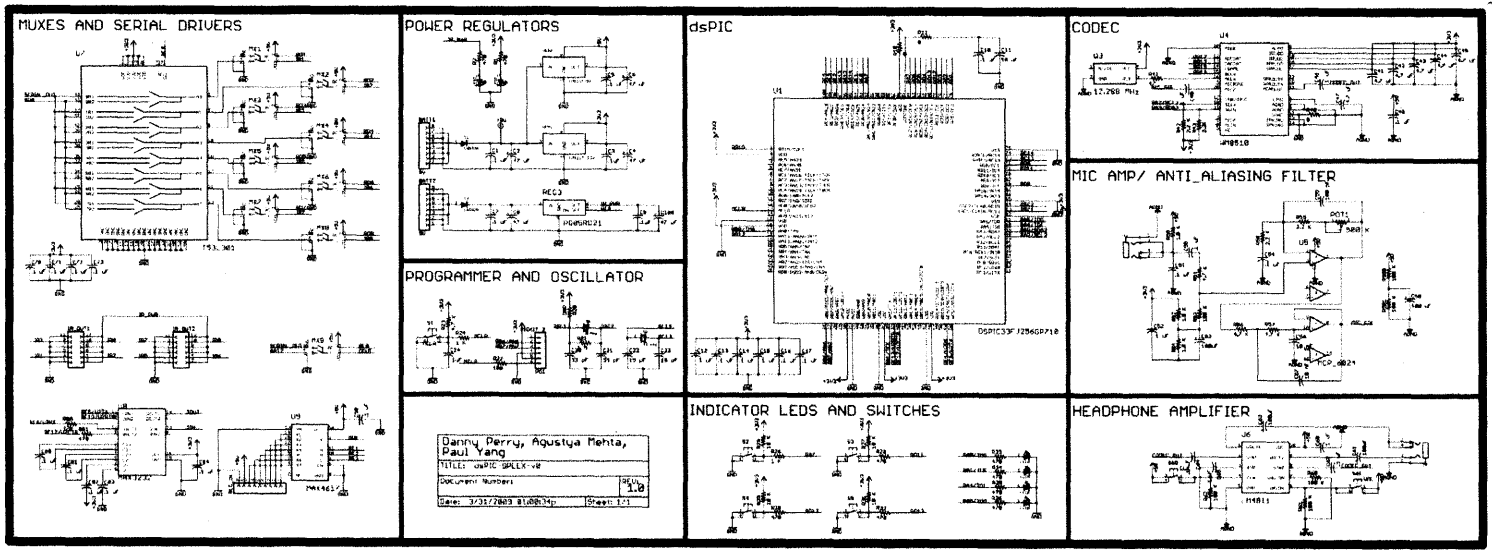
A-7: Complete schematic for final system board



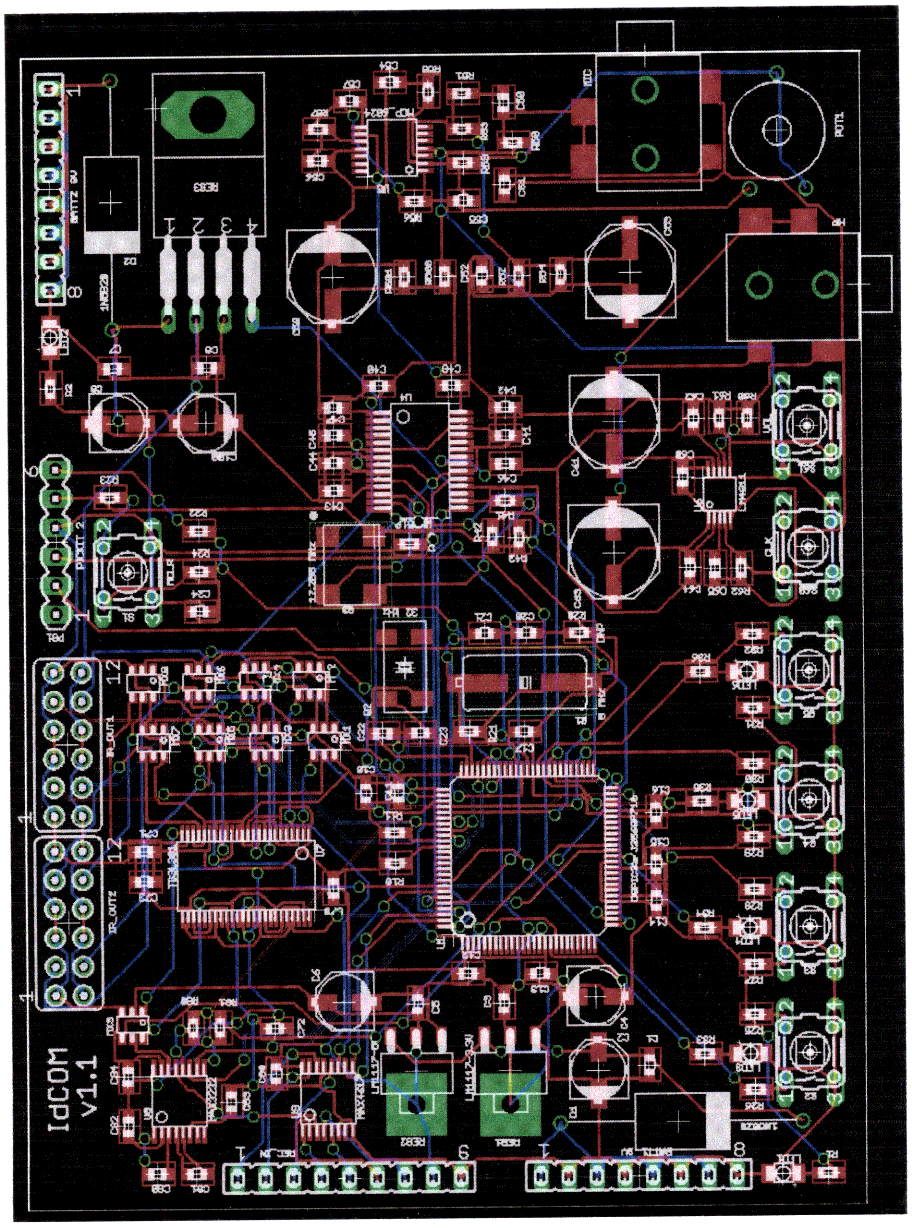
A-8: Final system board. Additional ground plane was added to improve performance.



A-9: System board without ground plane and programming port on outer layer. This board was used for testing only.



A-10: Complete schematic for initial system board.


A-11: Initial system board.

Appendix B

Select C Code

B.1 Transmit Packet

This function is responsible for initializing packet transmission. It copies data from the encoder buffer to a transmit buffer, calculates a cyclic redundancy check (CRC) and appends it to the end of the packet. Because the system is designed to support multiple transmitters this function also selects a transmitter channel. Timer 3, which pulses each bit in the packet, is turned on and transmission begins.

```
void TXPacket(TXhandle * pTXhandle, char * data) {

        int write_index;
        char * dest = pTXhandle->TX_data;

        for (write_index = 0; write_index < FRAME_SIZE; write_index++) {

                dest[write_index] = data[write_index];
        }

        SelectLED(pTXhandle->channel);
        CRCTX.CRC = crc16_ccitt((void *)dest, FRAME_SIZE);

        pTXhandle->status_flag = TRANSMIT_HEADER;
        pTXhandle->TX_count = 0;
        bit_count = 0;
        bit_start = 1;

        T3CONbits.TON = TIMER_ON;
}
```

B.2 Receive Packet

This function is responsible for copying valid data packets from the receiver buffer to the speech decoder buffer. The inputs are pointers to the receiver buffer and speech decoder buffer. First, the receiver lock status is checked. If the receiver is locked on a valid signal, then the data is copied to the appropriate buffer. If a valid packet has not been received then zeros are copied to the decoder (to prevent hanging on old packets). Since the system must support multiple channels, if a valid packet has not been received on the current channel within the specified amount of time the next channel is checked. At the end of the function, the lock status of the receiver is cleared.

```
void RXPacket(RXhandle * pRXhandle, char * dest) {

        int read_index;
        char * src = pRXhandle->RX_data;

        if (pRXhandle->lock_status != REC_NO_LOCK) {

                for (read_index = 0; read_index < FRAME_SIZE; read_index++) {

                        dest[read_index] = src[read_index];
                }

        } else {

                for (read_index = 0; read_index < FRAME_SIZE; read_index++) {

                        dest[read_index] = NULL_OUTPUT;
                }

                if (pRXhandle->status_flag == RECEIVE_HEADER) {


                        if (pRXhandle->no_lock_count < IDLE_COUNT) {

                                pRXhandle->no_lock_count++;

                        } else {

                                pRXhandle->no_lock_count = 0;

                                if (pRXhandle->channel < (NUM_CHANNELS - 1)) {

                                        pRXhandle->channel++;

                                } else {

                                        pRXhandle->channel = 0;
                                }
                                SelectRec(pRXhandle->channel);
                        }
                }
        }
        REC_LOCK_LED = pRXhandle->lock_status;
        pRXhandle->lock_status = REC_NO_LOCK;
}
```

B.3 Transmit Interrupt Service Routine

This interrupt service routine (ISR) is used to pulse bits on the transmitter ports. The ISR can be in several different states, both within a byte, and within a packet. First the ISR checks to see if a new byte must be loaded into the pulse register. This byte can come from several sources – a constant synchronization byte, the transmitter data buffer, or the CRC buffer. Once the byte is loaded, the start bit (logical one) is transmitted. Then, the ISR steps through the byte each time it is called until it finally transmits a stop bit and loads a the next byte into the register. Once all bytes of the packet have been sent, the ISR turns off timer 3 and awaits the next packet.

```
void __attribute__((__interrupt__,no_auto_psv)) _T3Interrupt (void) {

        int bit_mask = 0x01;
        int i;

        if (bit_start != 0) {

                if (thisTX->status_flag == TRANSMIT_IDLE) {

                        /* Do Nothing */

                } else if (thisTX->status_flag == TRANSMIT_HEADER) {

                        pulse_reg = SYNC_BYTE;
                        thisTX->packet_size = HEADER_SIZE;

                } else if (thisTX->status_flag == TRANSMIT_DATA) {

                        pulse_reg = thisTX->TX_data[thisTX->TX_count];
                        thisTX->packet_size = FRAME_SIZE;

                } else if (thisTX->status_flag == TRANSMIT_CRC) {

                        pulse_reg = CRCTX.crc_bytes[thisTX->TX_count];
                        thisTX->packet_size = CRC_SIZE;
                }

                thisTX->TX_count++;

                if (thisTX->TX_count >= thisTX->packet_size) {

                        thisTX->TX_count = 0;
                        thisTX->status_flag++;
                }

                F_PULSE_OUT = PULSE_HIGH;
                B_PULSE_OUT = PULSE_HIGH;

                bit_start = 0;

        } else if (bit_count >= 8) {

                F_PULSE_OUT = PULSE_LOW;
                B_PULSE_OUT = PULSE_LOW;

                bit_start = 1;
                bit_count = 0;

                if (thisTX->status_flag >= TRANSMIT_END) {

                        thisTX->status_flag = TRANSMIT_IDLE;
                        T3CONbits.TON = TIMER_OFF;
                }

        } else if ((pulse_reg & (bit_mask << bit_count)) == 0) {

                F_PULSE_OUT = PULSE_LOW;
                B_PULSE_OUT = PULSE_HIGH;

                bit_count++;

        } else {
                F_PULSE_OUT = PULSE_LOW;
                B_PULSE_OUT = PULSE_LOW;

                bit_count++;
        }

        for (i = 0; i < PULSE_WIDTH; i++) {

                Nop();
        }

        F_PULSE_OUT = PULSE_LOW;
        B_PULSE_OUT = PULSE_LOW;

        IFS0bits.T3IF = CLEAR_FLAG;

}
```

B.4 UART Interrupt Service Routine

The built in UART transceiver on the dsPIC33FJ256GP710 interrupts after a character has been received. This interrupt service routine is responsible for handling this character. First, the ISR waits for a valid header, which consists of two synchronization bytes. Once the header has been received, then the data is copied to the receiver buffer until it has been filled. Then the receiver calculates the cyclic redundancy check (CRC) for the received data packet and compares it to the sent CRC. If they match, then the receiver lock status is set indicating that a valid packet is ready to decode. The next time the ISR occurs, this function looks for the header for the next packet. If an invalid packet is received its contents are discarded and no retransmission is requested.

```
void __attribute__((__interrupt__,no_auto_psv)) _U2RXInterrupt (void) {

        if (thisRX->status_flag == RECEIVE_HEADER) {

                if (U2RXREG == SYNC_BYTE) {

                        thisRX->RX_count++;

                } else {

                        thisRX->RX_count = 0;
                }

                if (thisRX->RX_count >= HEADER_SIZE) {

                        thisRX->RX_count = 0;
                        thisRX->status_flag++;
                }

        } else {

                if (thisRX->status_flag == RECEIVE_DATA) {

                        thisRX->RX_data[thisRX->RX_count] = U2RXREG;
                        thisRX->packet_size = FRAME_SIZE;

                } else if (thisRX->status_flag == RECEIVE_CRC) {

                        CRCRX.crc_bytes[thisRX->RX_count] = U2RXREG;
                        thisRX->packet_size = CRC_SIZE;
                }

                thisRX->RX_count++;

                if (thisRX->RX_count >= thisRX->packet_size) {

                        thisRX->RX_count = 0;
                        thisRX->status_flag++;
                }

                if (thisRX->status_flag >= RECEIVE_END) {

                        thisRX->status_flag = RECEIVE_HEADER;

                        __asm__ volatile ("PUSH CORCON");
```

```
                    CORCONbits.PSV = 1;
                    CRCcalc.CRC = crc16_ccitt((void *)thisRX->RX_data, FRAME_SIZE);

                    __asm__ volatile ("POP CORCON");

                    if (CRCcalc.CRC == CRCRX.CRC) {

                            __asm__ volatile ("disi #0x04");

                            thisRX->lock_status = REC_LOCK;

                            __asm__ volatile ("disi #0x00");

                    } else {

                            __asm__ volatile ("disi #0x04");

                            thisRX->lock_status = REC_NO_LOCK;

                            __asm__ volatile ("disi #0x00");
                    }

                    REC_LOCK_LED = thisRX->lock_status;
            }
    }

    IFS1bits.U2RXIF = CLEAR_FLAG;
}
```

B.5 Select Transmitter Channel

This function selects a LED to use for each transmitter. The input is the channel to use, and the necessary clock data signal is sent to the transmitter. Since the clock for the bit shift register can be as fast as 60 MHz, no delays are added and the clock is outputted at the instruction clock frequency (40 MHz) to enable rapid switching. Since the LED synchronization was never implemented, the shift register was fixed to the value 0xFF for demonstrations.

```
void SelectLED(int channel) {

        int i; int delay;
        int shift_reg = (0x01 << (channel - 1));
        int shift_mask = 0x80;

        // Clock in LED value
        for (i = 0; i < 8; i++) {

                if((shift_reg & (shift_mask >> i)) == 0) {

                        F_TR_D = BIT_HIGH;
                        B_TR_D = BIT_HIGH;

                } else {

                        F_TR_D = BIT_LOW;
                        B_TR_D = BIT_LOW;
                }

                F_TR_CLK = CLK_HIGH;
                B_TR_CLK = CLK_HIGH;
                F_TR_CLK = CLK_LOW;
                B_TR_CLK = CLK_LOW;
        }

        F_TR_D = BIT_LOW;
        B_TR_D = BIT_LOW;
}
```

B.6 Select Receiver Channel

This function simply selects the receiver to use for this iteration of the loop. Since a multiplexer is used to select up to eight inputs, a three-wire control interface is needed. This function takes in a channel value and toggles each select line for the multiplexer. Bit zero is associated with select line one, bit one is associated with select line two, and bit two is associated with select line three. Thus the byte, 0x07, would set all of the select lines high and uses receiver channel seven.

```
void SelectRec(int channel) {

        REC_SEL1 = (channel & 0x01);
        REC_SEL2 = ((channel & 0x02) >> 1);
        REC_SEL3 = ((channel & 0x04) >> 2);
}
```

B.8 Voice Detection

This function determines if there is voice or noise on the microphone input line. It performs an approximate calculation of the power of the signal. To avoid overflow, the four least significant bits are discarded. The power of the signal is calculated by squaring each value and summing them. If this exceeds a threshold then the transmit flag is activated. The voice detector remains on for several frames after voice was detected because the power content of speech varies and the system cannot cutout for soft sounds. If speech is detected on the line, then the signal is encoded. If not speech is present then zeros are loaded into the encoded buffer to flush the previously encoded values. Note: the autocorrelation function would be the ideal method to detect voice, however, it was too slow and required too much memory.

```
int DetectVoice(void) {

        int sum = 0;
        int transmit_flag = CLEAR_FLAG

        for (i = 0; i <(NB_SPEEX_ENCODER_INPUT_SIZE - 1); i++) {

                int temp = (rawSamples[i] >> 4);
                sum += (temp * temp);
        }

        if ((sum > MIC_THRESH) | (consec_frames > 0)) {

                if (sum > MIC_THRESH) {

                        consec_frames = TAIL_LENGTH;

                } else {

                        consec_frames--;
                }

                VOICE_LED = LED_ON;
                bytes = Speex8KHzEncode(encoder, rawSamples, encodedSamples);
                transmit_flag = SET_FLAG;

        } else {

                VOICE_LED = LED_OFF;

                for (i = 0; i < FRAME_SIZE; i++) {

                        encodedSamples[i] = 0;
                }
        }

        return (transmit_flag);
}
```

B.9 Main Loop

This is the main loop for the processing unit. First, it waits until a frame is available from the codec. Then it determines if speech is present on the line. If there is speech, then the packet is encoded and transmitted. On the receiving end, the received packet is discarded and the encoded packet is decoded. Then it is sent to the codec, thus providing the loopback feature of the system. If no speech is present on the line then the receiver is checked to see if a valid packet is received. Once a valid packet has been received, its contents are decoded and sent to the codec. This loop ensures that the system is either transmitting or receiving, but never both simultaneously. There are also several 300 ms timeouts in the main loop to ensure that the system does not hang transmitting or receiving packets.

```
while(1) {
        while(WM8510IsReadBusy(codecHandle));

        WM8510Read(codecHandle,rawSamples, NB_SPEEX_ENCODER_INPUT_SIZE);

        transmit_flag = DetectVoice(void);

        if (transmit_flag == SET_FLAG) {

                transmit_flag = CLEAR_FLAG;
                EnableTimeout(pTimeouthandle, 30);

                while (pTXhandle->status_flag != TRANMSIT_IDLE) {

                        if (pTimeouthandle->flag != CLEAR_FLAG) {

                                break;
                        }
                }

                DisableTimeout(pTimeouthandle);
                TXPacket(pTXhandle, encodedSamples);

                for (i = 0; i < FRAME_SIZE; i++) {

                        recSamples[i] = encodedSamples[i];
                }

        } else {

                EnableTimeout(pTimeouthandle, 30);

                while (pRXhandle->status_flag != RECEIVE_HEADER) {

                        if (pTimeouthandle->flag != CLEAR_FLAG) {

                                pRXhandle->lock_status = REC_NO_LOCK;
                                pRXhandle->status_flag = RECEIVE_HEADER;
                        }
                }

                DisableTimeout(pTimeouthandle);
                RXPacket(pRXhandle, recSamples);
        }

        decoderRetVal = Speex8KHzDecode(decoder, recSamples, decodedSamples);

        if (decoderRetVal == SPEEX_INVALID_MODE_ID) {

                ERROR_LED = LED_ON;

        } else {

                ERROR_LED = LED_OFF;
        }

        while(WM8510IsWriteBusy(codecHandle));

        WM8510Write(codecHandle, decodedSamples, NB_SPEEX_DECODER_OUTPUT_SIZE);
}
```

Bibliography

[1] M. Spencer. Optoelectronic Fiber Interface Design, 2008.

[2] Y. Fink. Identification and Communication Multifunction Helmet, 2007

[3] J. Greenberg. Course Materials for HST582J/6.555J/16.456J, Biomedical Signal and Image Processing, MIT Spring 2008.

[4] J.M. Valin. The Speex Codec Manual Version 1.2 Beta 3, 2007.

[5] R. Kawashima and D Cai. A Design of a New Infrared Data Broadcasting Protocol and Educational Application. *Proceedings of the 2005 IEEE International Workshop on Wireless and Mobile Technologies Education*, 2005.

[6] G. Atkin K.S. Fung. Coded Multipulse Modulation in Optical Communication Systems. *IEEE Transactions and Communications, VOL. 42: 574-582*, 1994.

[7] T. Luftner, C. Kropl, M. Huemer, R. Weigel and J. Hausner. Wireless Infrared Communications with Edge Pulse Modulation for Mobile Devices. *IEEE Wireless Communication: 15-21*, 2003.

[8] J. Li and M. Uysal. Optical Wireless Communications: System Model, Capacity and Coding. *IEEE Vehicular Technology Conference VOL. 1: 168-172*, 2003.

[9] OSRAM Opto Semiconductors. High Power Infrared Emitter (850 nm) SFH 4550, 2009