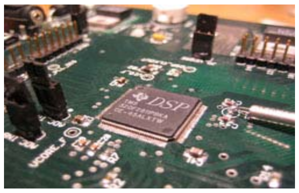
University of Waterloo

THE MARCONI PROJECT

www.sgsolutions.ca/marconi



Transparent Wireless Modem

ECE Fourth Year Design Project Final Report

University of Waterloo

Faculty of Engineering

Department of Electrical and Computer Engineering

200 University Ave. W., Waterloo, ON

Project ID: 2006.038

Jan 15th, 2006

Aaron Cheung
00141344

Stefan Janhunen
00057388

Vincent G. Liu
20069140

Shu Wu
20069140

Weihua Zhuang
Project Consultant

Abstract

Challenges arise when data transmission is required in remote or developing areas where communication infrastructures are nonexistent. This project develops a transparent wireless modem as an economical solution to transmit data wirelessly using UHF/VHF FM voice-band radios. The end result is a flexible and low cost wireless link that behaves like a serial data connection between two locations.

Acknowledgements

We wish to express our appreciation to Paul Hayes for his assistance in the assembly and testing of our prototype hardware. Without his help, it would not have been possible to complete the assembly of the project. We also express our appreciation to David Ma for overseeing the source control server throughout this project. His work made managing all our numerous project files and documents as painless as possible. Finally, we express our appreciation to our project consultant, Professor Weihua Zhuang, for her support and willingness to point us to the material we needed to complete the design.

Glossary

A/D
: Analog to Digital.

AGC
: Auto Gain Control, a process to normalize the signal strength about a fixed operating point.

AWGN
: Additive White Gaussian White Noise, a type of noise that has the same energy across all frequencys.

BER
: Bit Error Rate, a measurement of error rate.

bps
: bits per second, a measure of data transfer rate.

D/A
: Digital to Analog.

DSP
: Digital Signal Processor.

FIFO
: First In First Out.

FM
: Frequency Modulation, an angle modulation scheme used for digital communications.

FRS
: Family Radio Service, a radio service for short-distance communications.

GMRS
: General Mobile Radio Service, a radio service for short-distance communications.

GPIO
: General Purpose Input Output.

IC
: Integrated Circuit.

LED
: Light Emitting Diode, usually used as indicators or display elements.

MAC	Multiply and Accumulate.
PCB	Printed Circuit Board.
PCM	Pulse Code Modulation, a digital pulse modulation scheme for representing and transmitting analog signals.
PSK	Phase Shift Keying, an angle modulation scheme used for digital communications.
QPSK	Quadrature Phase Shift Keying, an angle modulation scheme used for digital communications.
RS-232	Recommended Standard 232. An Industry standard for asynchronous serial communication.
SNR	Signal to Noise Ratio, a measurement of the relative signal strength compared to the noise strength.
UART	Universal Asynchronous Receiver Transmitter.
UHF	Ultra High Frequency, the frequency range from 300MHz to 3GHz.
VHF	Very High Frequency, the frequency range from 30MHz to 300MHz.

Table of Contents

List of Figures

List of Tables

1 Introduction

In this section, the motivation for this project is introduced, the requirements are defined, and the project plan is presented.

1.1 Background

Many applications require a low-speed data link in rural areas for telemetry, data acquisition, or communications and control. The distances spanned may range anywhere from less than a kilometer up to 20 kilometers or more. In regions where no data infrastructure exists, it may be impractical or too costly to install landlines for these applications. Furthermore, many existing wireless data solutions would be unacceptable either due to insufficient range (802.11) or high cost (microwave, satellite, or specialized data links).

Relatively powerful VHF and UHF FM radios exist that can transmit voice-band audio signals in the range of 300 Hz – 3000 Hz over these distances. Transmitter power typically varies anywhere between 1 W to 30 W and above.

By developing a low-cost data modem that can interface these radios to computers or other digital equipment, an economical wireless data link can be established for these applications. Using off-the-shelf radios provides additional flexibility, allowing the modem to be interfaced to different radios depending upon the particular application. Providing a standard asynchronous serial data interface on the modem allows for connection with embedded computers, PCs, smart sensors, or other equipment. By implementing all of the core modem functionality in software rather than hardware, flexibility can be maintained even after the modem is manufactured. Figure 1 illustrates a typical application of such a modem.

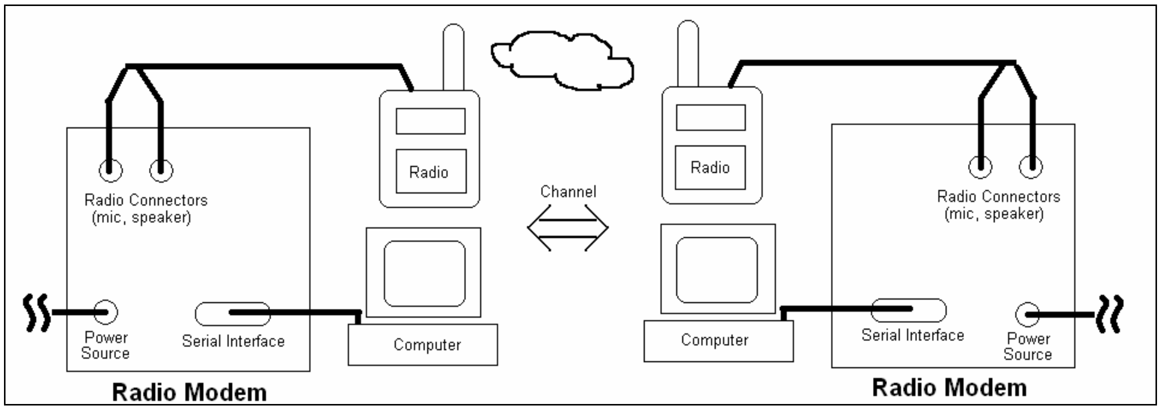


Figure 1 – System diagram of wireless data modem interfaced with radios and computers

1.2 Uses and Applications

Some of the possible applications of this wireless modem are enumerated in Table 1. Primary areas of application are those areas where landlines do not exist or are impractical and where it is not practical to use other off-the-shelf wireless solutions due to the required range.

Table 1 – Possible uses and applications of a transparent wireless modem

User/Application	Needs
Scientific research	Remote data must be collected; control information must be sent
Industrial control/monitoring	Data must be exchanged with process at a distance
Developing regions	Low-rate data links must span distances in areas with little or no infrastructure

1.3 Requirements

The functional requirements of the design are more or less a direct consequence of the device being a modem by nature as well as the existing hardware with which it must operate. Table 2 summarizes these requirements for this project. In short, these requirements ensure the modem can be connected to and function with existing voice band radios and that a pair of such modems connected to radios appears to the user to function as a simple serial link between two locations. The industry-standard RS-232 asynchronous serial protocol must be used for the serial data interface to ensure compatibility with a wide variety of devices. Transparency implies that the modem appears simply as a sort of virtual serial cable to the connected device. That is, no special commands or states are used to manipulate the modem through its data link. The modems should also implement error correction to reduce the Bit Error Rate (BER) over noisy channels and should allow for half-duplex operation over a single radio channel.

Table 2 – Modem functional requirements

Priority	Name	Measure	Description
MUST	Radio Interface	By design	Must work with audio interface of voice-band radios
MUST	Signal bandwidth	300 Hz – 3000 Hz	Modem output signals must fit within audio bandwidth of radios
MUST	Serial interface	By design	Must interface with other equipment over a standard RS-232 serial interface
MUST	Transparent operation	By design	Must provide transparent operation through RS-232 and behave similarly to landline modems or null-modem connections
SHOULD	Forward error correction	By design	Should implement forward error correction to reduce error rate of received data
SHOULD	Half-duplex communication	By design	Modem should be able to operate in a half-duplex mode for two-way operation over a single radio channel
SHOULD	Audio input levels	-20 dBV – 0 dBV	Should tolerate reasonable range of nominal input signal amplitudes
SHOULD	Audio output levels	-20 dBV – 0 dBV into 600 Ω	Should be capable of producing reasonable range of output signal amplitudes

The non-functional requirements in Table 3 list some design aspects that impact overall user experience. The modem must not be excessively heavy or large. Power consumption should be kept within reason so that it can be operated in areas with limited power sources such as generators or perhaps even large batteries. These requirements will be followed for prototype production, but more stringent requirements on size and power are likely required for final production hardware. It will be possible to explore further improvements once an initial prototype is functioning as a reference.

Table 3 – Modem non-functional requirements

Priority	Factor	Requirement
MUST	Size	PCB must be less than 25 cm in either direction
MUST	Weight	Less than 3.0 kg in weight
SHOULD	Power	Should not require over 6 VDC @ 800 mA
SHOULD	Portability	Should be small and light enough to be portable

The data transfer rate and the modem's bit error rate over noisy channels are two metrics by which the final design will be evaluated. Below are the suggested *minimum* acceptable performance metrics for this modem. These are not intended to approach theoretical bounds, but are intended to provide an indication of what is acceptable or unacceptable performance. For the target applications, relatively low data rates are acceptable. It is more important that the overall design is robust and can function with existing voice radios than that it is especially fast. In many applications, additional protocol layers will be added to the transparent data link provided by the modem. Some residual BER is thus acceptable for applications where additional protocols introduce error checking and correction.

Table 4 – Modem minimum performance metrics

Metric	Target Minimum	Description
Net data transfer rate	100 bits/second	Must transfer data at minimum rate over reasonable channels (15 dB SNR per bit)
Bit error rate	10^{-3}	Must not exceed 1 bit error per 1000 bits over reasonable channel conditions (15 dB SNR per bit)

A final aspect of the design that must be considered is the modem unit cost. Table 5 lists the target bill of materials cost for both prototype and production units. Prototypes are generally significantly more expensive to make due to the extremely low part and production volumes.

Table 5 – Modem unit cost

Prototype (1-10)	Mass Production (100-1000)
< $300 CDN each	< $150 CDN each

1.4 Risks

Most of the project risk can be minimized by performing a sufficient amount of up-front research and planning. Table 6 details some of the critical risks that have been identified.

Table 6 - Project risks

Level	**Description**	**Response**
High	Hardware implementation non-functional	Demonstrate modem in software simulation
High	Interfacing to voice radios problematic	Test interfaces early and choose appropriate radios
Medium	Modem has insufficient performance for required goals	Consult with advisor and perform sufficient up-front research
Medium	Insufficient memory and processing resources to implement required features	Choose embedded processor with plenty of processing power and memory to spare
Low	Hardware prototypes cannot be produced in time for demonstration	Begin work on hardware prototypes immediately

1.5 Project Plan

The milestones in Table 7 represent the different phases of the project. Associated with each milestone is a well defined deliverable that should be produced by the target completion date. Each milestone is broken down into different tasks that will be partitioned among the group members. Time estimates are given for each task. While each milestone has a single target completion date, work for several milestones can proceed in parallel since there is a very clean division between the hardware, firmware, and software modem aspects of the design. The list is also somewhat non-linear. For example, a significant amount of high-level analysis must be complete before any work on hardware or software can begin. Thus, even though the Block Verification deliverable is not produced until later in the schedule, the work for that deliverable will unfold sooner as necessary.

In order to successfully complete the many hardware and software aspects of the project, significant work will have been completed on high-level analysis, hardware development, and firmware development by the end of the 3B academic term (as per risk analysis) in order to ensure that the first two milestones can be met successfully.

Table 7 – Major project milestones

Milestone	*Deliverables*	*Target Completion Date*	*Estimated Man Hours*
Hardware development	Functional hardware	9-May-05	
Schematic design			50
PCB layout			50
PCB assembly and testing			50
Firmware development	Functional firmware drivers	16-May-06	
Develop hardware interface drivers			50
Develop debug layer			20
Test drivers and debug layer			10
Research and Simulation	Block Verification	13-May-05	
Test and characterize radios			15
Research modem architecture			20
Simulate modem algorithms			50
Software modem design	Detailed Design	3-Jun-05	
Compare alternatives and choose algorithms			10
Design modem architecture (blocks)			25
Partition block implementation			5
Software modem implementation	Functional Software Modem	18-Jul-05	
Block development			100
Block-level testing (in simulation)			25
System-level testing (in simulation)			50
Hardware-implementation testing (in hardware)			25
Prototype testing checklist	Same	11-Jul-05	5
Prototype demonstration	Same	25-Jul-05	5
Experience report	Same	29-Jul-05	10
Total project man hours			575

2 High-Level Analysis

The high-level partitioning and analysis of the system will now be considered. The goal is to partition the design sufficiently so that the feasibility of the design can be considered and key design decisions can be made that will govern detailed design. This section is based upon the Block Verification deliverable with some modification in organization and detail as seen fit. Given that the project scope includes hardware, software, and signal processing design, significant up-front analysis is required.

While effort is made to make this discussion understandable without significant special knowledge, some basic knowledge of hardware and software design as well as fundamental communications system terminology is assumed.

2.1 Design Overview

The major divisions of the hardware and software portions of the design will be designated first. This will pave the way for further analysis.

2.1.1 Hardware Partitioning

Figure 2 illustrates a simplified block diagram of the modem hardware, which will ultimately be realized on a Printed Circuit Board (PCB). The audio and serial port interfaces are a direct consequence of the application illustrated in Figure 1. An external radio is connected to the audio interface and an external PC or other serial device is connected to the industry-standard RS-232 serial interface.

A highly integrated embedded Digital Signal Processor (DSP) is used to execute the modem software. These devices typically include all memory necessary for program storage and execution as well as an assortment of hardware to support various peripherals. A DSP is preferable to a general-purpose microcontroller due to the computationally intensive nature of the signal processing software required for a software-based modem. The DSP is connected to the serial interface for data transfer between the modem and external devices or a host PC and to the audio codec to transmit and receive digital audio samples before or after conversion.

The audio codec provides analogue-to-digital (A/D) and digital-to-analogue (D/A) conversion for voice-band audio input and output, respectively. It is connected to the audio interface, which ultimately drives an external radio.

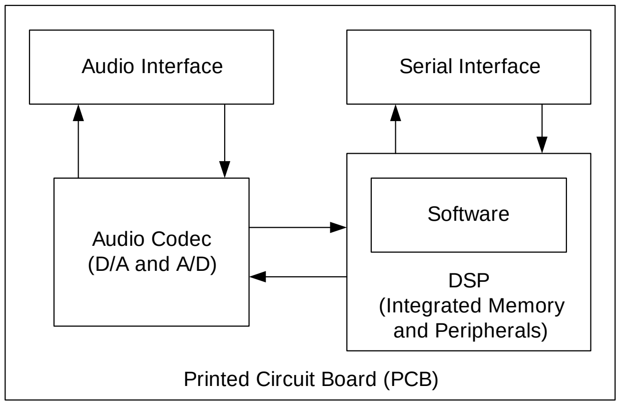


Figure 2 – Hardware block partitioning of modem

2.1.2 *Software Partitioning*

Figure 3 illustrates the rough partitioning of software required for the modem. At the highest level, the software is divided into modem software and firmware. The modem software performs the actual modem tasks such as signal processing and encoding and decoding. The firmware provides an abstracted interface to the hardware features that allows the modem software to be kept independent of the particular hardware platform. This has the added advantage of making it easy to simulate and test all modem software on a PC before loading the code on an embedded DSP, where debugging is more difficult. Debugging support is provided through the serial interface that allows debug messages and commands to be exchanged while the modem software is being executed in hardware on the DSP. This allows for some level of in-system debugging for problems that do not appear in simulation.

Referring to Figure 3, the modem software is partitioned into a transmit pipeline and a receive pipeline. Only one of these can be active at a time, based upon the requirement for half-duplex communication noted in the project requirements. The specific software modem blocks are a direct consequence of the project requirements and are based upon industry-standard communication system design [1].

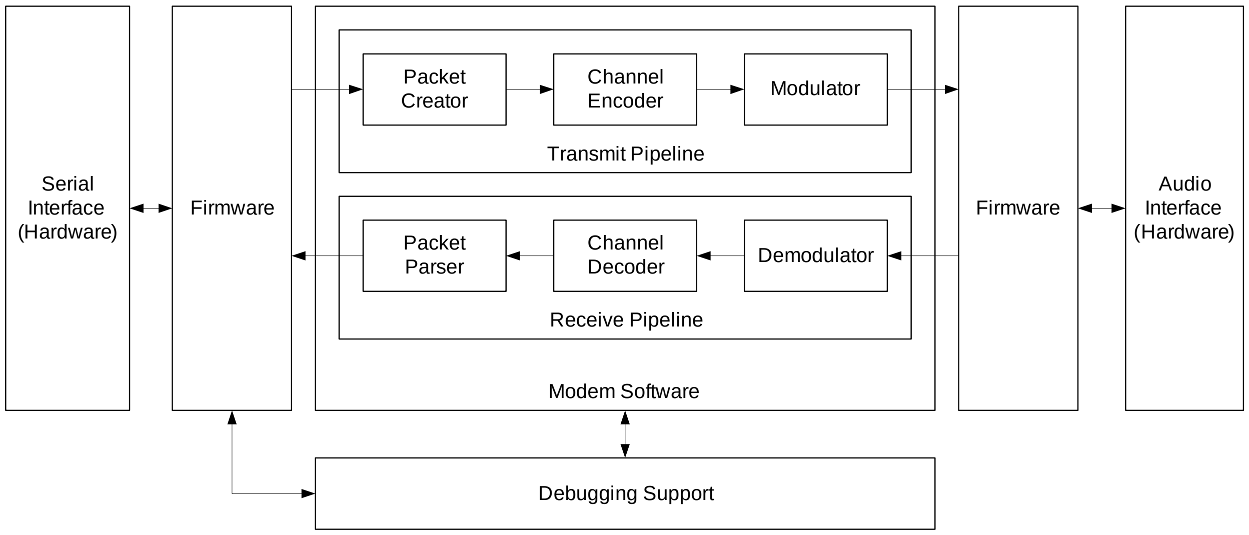


Figure 3 – Software block partitioning of modem

The modulator and demodulator are necessary to convert data bits into modulated tones of some type and vice versa. This is necessary since it is not possible to directly modulate data pulses through radios designed to accept voice bandwidth only.

The channel encoder and decoder are used to provide forward error protection for the transmitted data as recommended. Forward error protection refers to the fact that no re-transmission is required to correct errors. Redundancy is injected into the transmitted bit stream and used to correct errors at the receiving end. This allows for reliable communication in the presence of noise at the cost of bandwidth. Since the primary applications require low data rates, this is a reasonable tradeoff.

Finally the packet creator and parser are used to partition and reassemble the transmitted data and to provide the necessary bit-level synchronization required between the transmitter and receiver. Optionally, the packets can allow for automatic retransmission of corrupted data to provide some level of guaranteed delivery from transmitter to receiver. All of these features allow for the original goal of transparent operation in the sense that none of this behavior is visible through the serial interface. Thus, devices or PCs using the modem at the local end see the modem as a virtual “serial cable” to the remote end of the link.

The C language was chosen to implement the project since it was already familiar to the team members. The language provides a good balance between ease of use and performance and is virtually universal in its acceptance as a development language for embedded software today. For any chosen hardware platform, it is practically guaranteed that a C compiler exists.

2.2 Voice-Band Radio Performance Measurements

The voice-band radios used with the modem play a key role in determining design decisions, particularly for the modulator and demodulator blocks. For this reason, it is necessary to have at least some metrics by which the radios can be characterized.

The Cobra MicroTalk PR-245-2 GMRS radio was chosen for use in this project due to its low cost and simple audio interface that will allow connection to the modem. This 1-watt UHF (Ultra High Frequency) band FM (Frequency Modulation) radio provides a relatively clean voice-grade link spanning several kilometers. Due to the limited power of this radio as well as regulatory limitations on using GMRS (General Mobile Radio Service) radio channels for data transmission, this unit is really only feasible for use in testing the modem in a controlled environment. However, the modem will not be limited to functioning with this particular radio and will function well with higher-power UHF voice-band radios that operate on other radio channels.

Since the particular characteristics of this radio were unknown, several tests, two of which are detailed in the following discussion, were performed to provide performance characterization. It should be noted that all of these test were performed indoors with the radios 10 cm apart. Thus, they represent the best-case conditions, or an upper bound on voice-link performance. The PR-245-2 supports both FRS (Family Radio Service) and GMRS channels. The maximum allowable transmit power for GMRS channels is 1 watt as opposed to 0.5 watts for FRS channels. Additionally, channel spacing is wider over GMRS channels. The tests discussed here were performed only over FRS channels.

2.2.1 Frequency Response

A vital parameter in the design of the modem is the bandwidth available for transmitting data. Figure 4 illustrates the measured frequency response of the end-to-end bandpass voice link. A frequency swept tone was transmitted from one radio while the received amplitude response was measured at the output of the other radio. It should be noted that the vertical offset of the response (or the passband amplitude) is somewhat arbitrary since it depends upon the gain settings of the receiving audio equipment. What does matter is the relative variation of the response. From Figure 4 it should be noted that frequencies below 300 Hz and above 2000 Hz are significantly attenuated. Thus, as long as the modulated spectrum is constrained within the passband sufficiently the frequency response should not present an obstacle to data transmission at sufficiently low data rates.

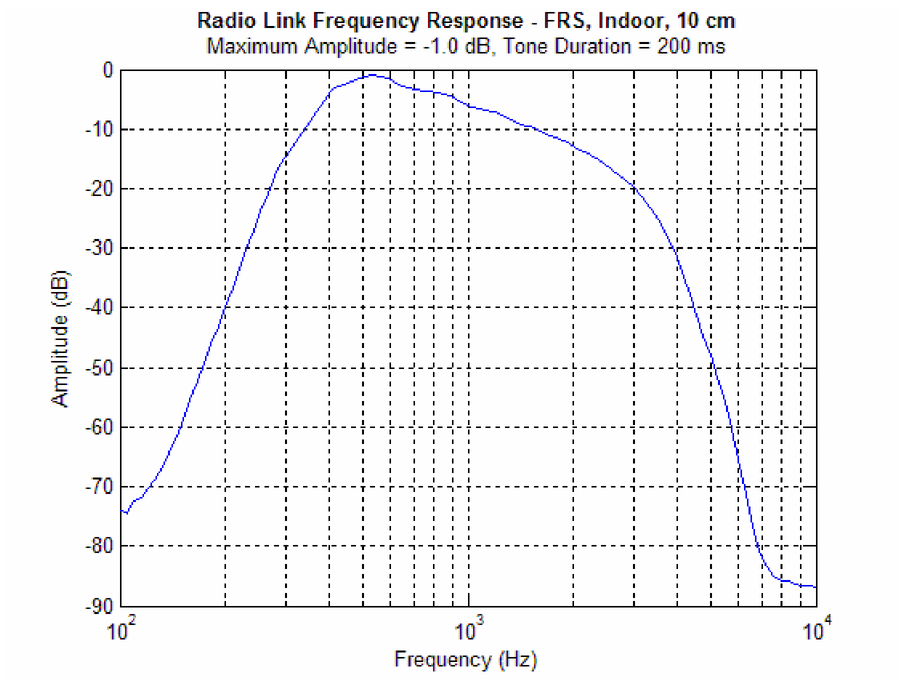


Figure 4 – Frequency response of voice-band channel

2.2.2 Noise Floor

Another useful metric is the noise floor of the radios at close range. Figure 5 illustrates the received power spectrum of a 1000 Hz test tone transmitted between the two radios. The amplitude of the transmitted tone was maximized while attempting to keep the power of the odd harmonics from increasing due to audio circuitry distortion in the radios.

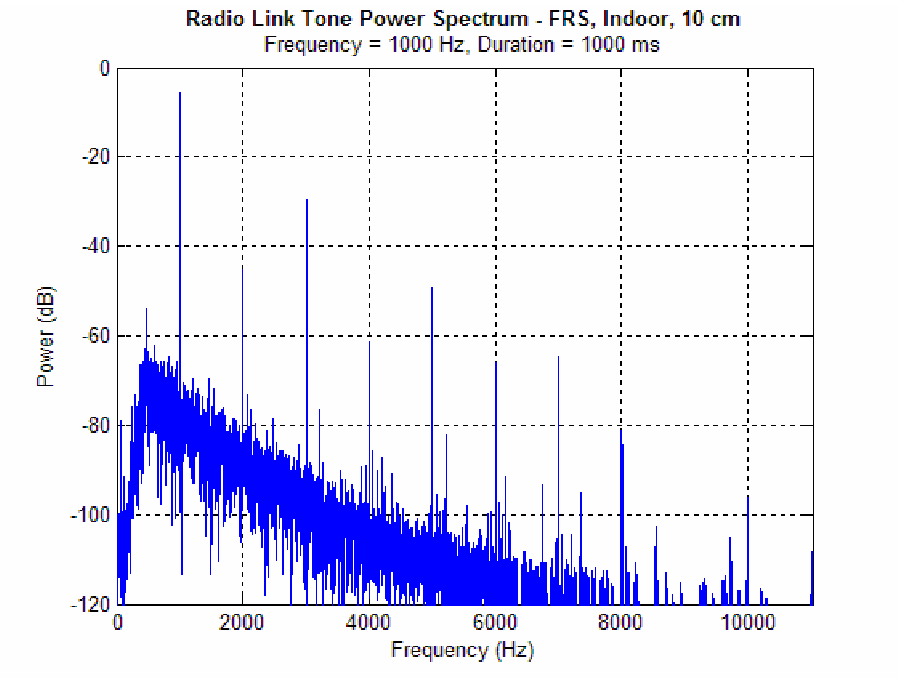


Figure 5 – Power spectrum of test tone

Several observations are now possible. This measurement represents a best-case channel condition: maximum transmitted signal power with minimum path loss. As path length is increased, the tone will eventually disappear into the noise floor. Also, interference from other sources will serve to increase the noise floor. Noise conditions over longer path lengths will not be explored here, but will be simulated implicitly as part of the BER performance measurements.

The harmonic distortion introduced by the radios is evident in Figure 5, particularly at the odd harmonics. Non-linear amplification and FM modulation imperfections contribute to this distortion.

The SNR (Signal-to-Noise Ratio) for the test tone under these conditions was measured by bandpass filtering the receiver output both in the presence and absence of the tone and comparing the relative measured power. The equivalent noise bandwidth (BW_N) of the measurement filter and the SNR (C/N) of the tone were calculated to be:

$$BW_N = 1006.95\ \text{Hz}$$

$$\frac{C}{N} = 10\log_{10}\left(\frac{P_{TONE}}{P_{NOISE}}\right) = 35.48\ \text{dB}\,.$$

From these figures, and with knowledge of the desired baud rate, it is possible to estimate the ratio of bit energy to noise power density for a particular data rate under ideal conditions and get a bound on the best-case BER performance. This analysis will be considered in a later section.

2.3 Hardware Design Analysis

Specific high-level design decisions for the modem hardware will now be considered based upon the previous analysis. Referring back to Figure 2, there are four blocks that fall into the hardware section of the radio modem, namely the serial interface, DSP, the audio codec, and the audio interface. Each block serves a specific function in the modem. The following sections will briefly describe the functions of each block, followed by an overview to evaluate whether the specifications listed in the project requirements can be satisfied.

2.3.1 Serial Interface

The main function of the serial interface is to allow data transfer between the radio modem and the device attached, a portable computer, for example. Since the radios in consideration will support half duplex data transfers, one of the two modems will be transmitting data while the other will be receiving data at any given time. Since the serial interface is specified as industry standard RS-232 in the project requirements, the major remaining concerns are the speed of data transfer and the number of serial ports required.

The serial interface must be able to transmit and receive data above a minimum speed. From the project specification, it is required that the modem should be able to transmit or receive data greater than a rate of 100 bits per second (bps) over a reasonable channel. A preliminary investigation shows that current RS-232 technology supports data transfer speeds up to or greater than 250kbps, hence this requirement can easily be satisfied.

Regarding the number of serial ports, the radio modem requires only one serial port under normal operating conditions. However, during the development phase, it is much more convenient for the designer to have an additional serial port for hardware debugging. Consequently, it is desirable to have a serial interface that has two serial ports. Preliminary investigation shows that most DSPs support at least two asynchronous serial interfaces, hence this requirement is not a concern.

2.3.2 Audio Codec

The audio codec serves as a digital to analog (D/A) converter on the transmitting side and an analog to digital (A/D) converter on the receiving side of the radio modem, connecting the audio interface and the DSP. On the transmit side, the audio code receives a pulse-coded modulation (PCM) bit stream from the DSP and outputs an analog signal to the audio interface. On the receive side, the audio codec samples the analog signal and outputs a PCM bit stream to the DSP for processing. The two key concerns for the audio codec are resolution and sampling speed.

The number of required bits is bounded by the required dynamic range of the voice-band radio link. Since the SNR of the voice link was previously measured as about 35 dB for a full scale tone, it would be reasonable to allow for at least 8 bits of resolution per sample, which allows for roughly 48 dB of dynamic range. Greater resolution will not hinder performance, but will probably not improve it significantly either.

Referring back to Figure 4, the usable frequency response of the measured voice link does not extend past several kilohertz. A common sampling rate for voice systems is 8 kHz, which allows signals up to 4 kHz in frequency to be represented. Considering that the voice link amplitude response is significantly attenuated at 4 kHz, it is not likely that higher sampling rates will be necessary.

Single-chip audio converters that fit the above requirements are widely available in the industry. The particular chip under consideration for this project is the Texas Instruments TLV320AIC10

[2], which provides 16-bit A/D and D/A conversion at numerous possible sampling rates determined by an external clock.

2.3.3 Audio Interface

The main function of the audio interface is to amplify or attenuate signals as required depending on the choice of radio modules, so as to ensure signals can be transmitted or received clearly. On the transmitter side, operational amplifiers along with potentiometers can be used to amplify or attenuate signals from the audio codec to the radio modules as desired to ensure clear transmission. On the receiver side, op amps and potentiometers can also be used to ensure clear reception from the radio modules to the audio codec.

To interface with a wide variety of equipment, the audio interface should allow the input and output levels to be adjustable between 0 dBV and –20 dBV. Since most op amps can deliver such performance, the audio interface block pertains no major concern for the project deign.

2.3.4 Digital Signal Processor (DSP)

The DSP is the most important block within the hardware section. Its main function is to carry out all computation required by the modem software in Figure 3. On the transmission end of the transmit-receive pair, the DSP receives binary information from the serial interface, performs numerous operations on the raw data in preparation for transmission, and finally sends the transformed bit stream to the audio codec. On the receiving side, the DSP performs the inverse operations in a reversed order to produce data to be sent off to the serial interface. Of all the operations performed on both the transmitter and receiver side, the demodulation operation is one of the most computationally intensive. As such, it is necessary to investigate the computational speed of the DSP and its internal peripheral features.

During demodulation on the receiver’s end, the DSP block receives a PCM bit stream from the audio codec and transforms the modulated data back into bits. Assuming a simple modulation-demodulation scheme like Frequency Shift Keying (FSK) is used, the DSP will need to perform convolution on the input bit stream in real time to identify which of the possible tones was transmitted. Discrete-time convolution can be broken down into multiplication and addition operations, known as multiply and accumulate (MAC) operations.

With a preliminary investigation of current technologies, most embedded DSPs are sufficiently powerful to meet this requirement. The DSP under investigation for this project is the Texas Instruments TMS320F2811 [3], which is capable of performing single-cycle MAC operations at its operating speed of 135MHz. Assuming a sampling rate of 8 kHz, this allows for roughly 16,000 MACs per sample. If a symbol rate of 100 symbols per second were used, the symbol length would be 80 samples. If matched filters were used in the demodulator, it would be possible to process 200 matched filters at the full sampling rate of 8 kHz. While still a rough estimate, this is plenty of processing power for a simple multi-tone receiver, leaving plenty of power left for the other modem blocks. Since the other blocks operate on the symbol or bit level, as opposed to the audio sample level, processing requirements are not as much of a concern.

In addition to the DSP's computational power, it is also necessary to consider the peripherals included in the package, so as to simplify circuit integration and avoid unforeseen hardware problems in the future. For the DSP to perform computations on the data for transmission and reception, there must be sufficient (non-volatile) FLASH memory to store the software and sufficient (volatile) RAM to store data for processing. The TMS320F2811 includes 36K of RAM and 256K of FLASH, which should be sufficient for the purposes of this software modem.

2.4 Modem Software Design Analysis

Key design aspects of the software modem itself will now be considered. High-level analysis for the transmit and receive pipelines of Figure 3 will be conducted. High-level design decisions will be made and discussed to prepare the way for detailed-design.

2.4.1 Modulation Scheme

Several aspects that affect the modulation scheme design will now be considered. Modulation schemes that have constant or relatively constant envelopes are less sensitive to fluctuations in amplitude and non-linearity in amplification. Thus in applications where these effects are factors, angle modulation of some form is generally preferred. Two forms of angle modulation will be considered here: FSK (Frequency-Shift Keying) and PSK (Phase-Shift Keying).

Referring back to Figure 3, the channel encoder and decoder reduce the net data rate of the system. This reduction can be offset by choosing a multilevel modulation scheme that compensates for the loss in data rate by increasing the number of bits transmitted per symbol. Allowing for a one-half rate convolutional code, the net data rate will remain unchanged if two bits are encoded in each transmitted symbol for 4 unique symbols in total. For the cases of FSK and PSK, this implies using 4-FSK or 4-PSK (also known as QPSK), respectively.

Figure 6 illustrates the uncoded BER curves for these two modulation schemes. Instead of using SNR as the horizontal variable, a normalized metric of bit energy per noise power density (E_b/N_o) is used. This is the ratio of the energy contained in one transmitted bit divided by the noise power per Hertz of bandwidth. By converting SNR figures to E_b/N_o, implementations with different symbol rates (baud rates) and different receiver filter bandwidths can be compared.

The upper curve represents the error rate for non-coherent, orthogonal 4-FSK. In this modulation scheme, four tones are used to encode all four possible values of a symbol of two bits. Non-coherent receivers do not rely on phase synchronization for the demodulation of the modulated signal. The advantage of this is simplicity of implementation. The cost is reduced performance over coherent demodulation schemes. Orthogonality implies that the various tone frequencies are spaced such that they do not interfere with one another when filtered with a matched filter.

The lower curve represents the error rate for (coherent) differentially encoded 4-PSK. In this scheme, the four possible values of a two-bit symbol are encoded into four equally-spaced carrier phases (each 90° apart). Coherent demodulation is used to recover these shifts in carrier phase. Additionally, the bits are usually differentially encoded so that the exact starting phase is not required to recover the data, only the relative phase shifts.

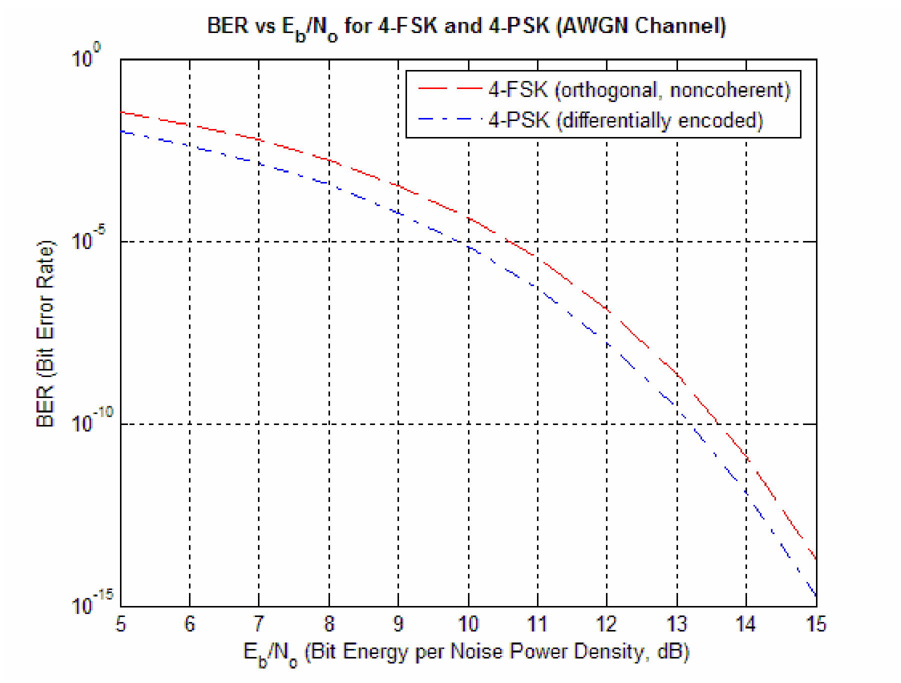


Figure 6 – Error rate curves for 4-FSK and 4-PSK

To evaluate the best-case theoretical performance of these two modulation schemes with various SNRs, it is necessary to convert SNR figures to E_b/N_o. This can be done using

$$\frac{E_b}{N_o} = \frac{1}{n_b} \cdot \frac{C}{N} \cdot \frac{BW_n}{f_b},$$

where C is the carrier power, N is the noise power, BW_n is the equivalent noise bandwidth of the noise-measuring filter, f_b is the symbol rate (or baud rate), and n_b is the number of bits per symbol. Here, the ratio of carrier to noise power is the same as SNR, except that it is expressed linearly instead of in decibels.

With our previously measured noise floor for the radios used in this project, the following results are obtained:

$$\frac{C}{N} = 10^{\frac{SNR}{10}} = 10^{\frac{35.48}{10}} = 3531.8 \text{ and } BW_n = 1006.95\,\text{Hz}.$$

This yields

$$\frac{E_b}{N_o} = \frac{1}{n_b} \cdot \frac{C}{N} \cdot \frac{BW_n}{f_b} = \frac{3531.8(1006.95)}{2(100)} = 17781.89 = 42.5 \text{ dB}$$

for a symbol rate of f_b = 100, corresponding to the minimum data rate set in the project specification (100 bits per second) while allowing for a one-half rate channel code. From Figure 6, it can be seen that this best-case condition of E_b/N_o will result in an extremely low BER for both 4-FSK and 4-PSK without considering any coding improvement. The error rate is so low, it is not shown on the graph.

The analysis can also proceed in the reverse direction. In the project specification, the maximum acceptable BER for usability was set at 10^{-3}, or 1 error in 1000 bits. The E_b/N_o corresponding to this condition (neglecting any improvements due to coding) can be estimated from Figure 6 to be near E_b/N_o = 8.3 dB for the case of 4-FSK. Working backwards, it can be seen that this requires a C/N ratio of

$$\frac{C}{N} = n_b \cdot \frac{E_b}{N_o} \cdot \frac{f_b}{BW_n} = 2 \cdot 10^{\frac{8.3}{10}} \cdot \frac{100}{1006.95} = 1.342 = 1.3 \, \text{dB} .$$

The C/N radio for the same error rate using 4-PSK will be even lower. All of these numbers assume AWGN and an ideal demodulator. In reality, neither of these conditions will be achievable. Thus the numbers above serve only as bounds on the theoretically achievable modem performance. Actual error rates for the implemented modem will be considered later.

One additional aspect that must be explored is the power spectrum produced by each modulation scheme. This is important since the voice-band radios provide a band-limited channel for transmission. Figure 7 illustrates a typical power spectrum of a modulated square wave for a single carrier frequency (PSK or on-off keying). The main lobe is shown centered around a normalized frequency of zero. In reality, this would correspond to the carrier frequency of the modulated data, but the analysis still holds. While the spectrum theoretically extends indefinitely, it is convenient to approximate the minimum bandwidth required for transmission as the width of the central lobe, as it contains most of the signal power. For simplicity, no pulse shaping will be considered.

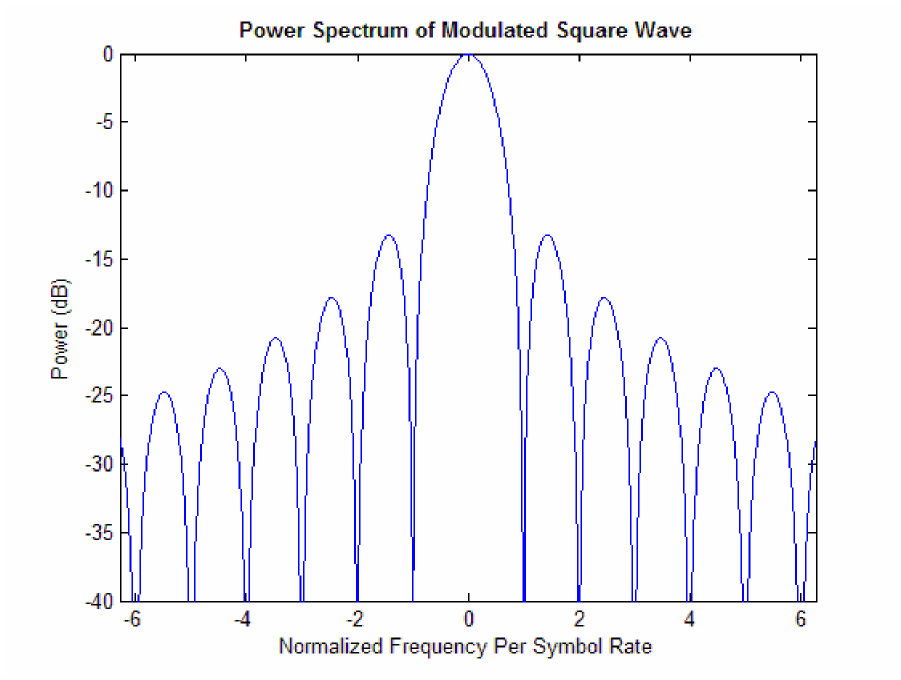


Figure 7 – Typical power spectrum of a modulated square wave

The width of the main lobe is

$$W = 2 f_b ,$$

where f_b is the baud rate or symbol rate of the baseband digital signal. This is seen in Figure 7 by the fact the main lobe extends from –1 to 1, representing a (normalized) frequency width of twice the symbol rate.

A 4-PSK modulated signal contains a single carrier with varying phase. Thus, the bandwidth required for a symbol rate of 100 symbols per second is estimated as the width of the main lobe:

$$BW = W = 2 f_b = 2(100) = 200 \text{ Hz} .$$

4-FSK modulation can be thought of as four independent carriers each modulated with on-off keying such that only one is on at any time instant (at least for the case of unfiltered baseband pulse data). For orthogonality in non-coherent demodulation it is required that a given carrier be located at the spectral nulls of all other carriers. From Figure 7 it is evident that the shortest possible spacing between carriers is thus unity, or f_b, the symbol rate. In this case, the required bandwidth of 4-FSK for a symbol rate of 100 symbols per second can be estimated as the total width of all the main lobes when spaced appropriately:

$$BW = (M+1)W = (M+1)2f_b = (4+1)(100) = 500\,\text{Hz}\,.$$

From the frequency response in Figure 4, it can be seen that frequencies below 300 Hz and above 2000 Hz are significantly attenuated, leaving a usable bandwidth no larger than about 1700 Hz. Moreover, the response is flattest between 400-800 Hz. From the above calculations, it should be possible to work with either 4-FSK or 4-PSK within the bandwidth of the specified radios while still meeting the project requirements.

A decision was ultimately made in favor of 4-FSK due to the simplicity of robust implementation. Since the design is non-coherent, it is not necessary to track phase and frequency variations in the received carrier. This simplifies the receiver design significantly at the expense of some BER performance. This was determined to be a reasonable tradeoff given the group's level of experience and the time frame of the project.

To choose the appropriate signal set for 4-FSK modulation, it is necessary to consider the available bandwidth and the project requirements. According to project specifications, a net data rate of 100 bits per second is required. Allowing for a one-half rate channel-coding scheme, this necessitates a minimum gross bit rate of 200 bits per second or a symbol rate of 100 symbols per second.

The goal is to meet or exceed this requirement with the choice of FSK tone set and associated symbol rate. Tests were performed using Matlab and the voice-band radios to evaluate the inter-symbol interference of various tone sets when transmitted through the actual radios. After some experimentation, the following set of tones was chosen: 600 Hz, 800 Hz, 1000 Hz, and 1200 Hz. These tones allow for orthogonal FSK signaling at a symbol rate of 200 symbols per second, meeting the project requirements with some extra margin. As illustrated in Figure 8, the main spectral lobes of the four tones fit within a bandwidth of 1000 Hz, stretching from 400 Hz to 1400 Hz. Consulting the measured frequency response in Figure 4, this still allows for some additional spectral roll-off in the octave above and below the main tone lobes.

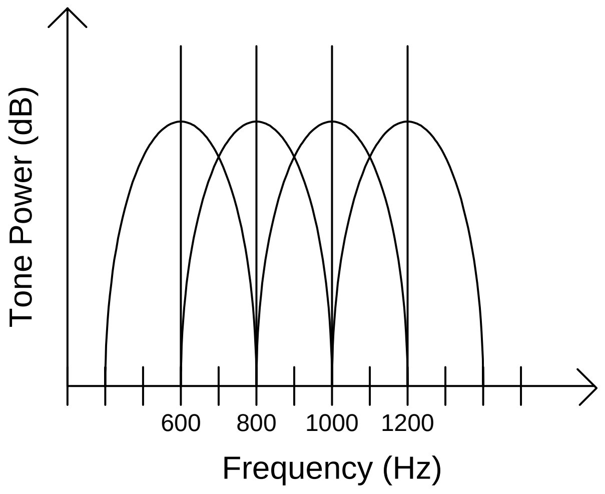


Figure 8 – Frequency spectrum of FSK tone main lobes

2.4.2 Channel Coding

The function of channel coding is to reduce the BER of the radio link in noisy conditions at the expense of data bandwidth. Two error correction channel coding methods were considered for this project: block coding and convolutional coding. A combination of the two implemented in sequence or recursively (Turbo Codes) was also considered.

Block codes are effective at correcting low uncorrelated bit errors while convolutional codes can be scaled to correct higher rates of correlated bit errors. When implemented in sequence, data is often block coded and then convolutionally coded to obtain optimal error reduction for radio transmission. Turbo codes are an optimal technique approaching Shannon's limit, however it is complex to implement and requires a high amount of processing.

It was realized early in the design process that Turbo codes would be beyond the scope of this project. Choosing to use block codes only would likely be insufficient for the high noise conditions expected when approaching the radio range limits. And using a combination of both would also be an inefficient allocation of development time.

The decision was made to use a convolutional encoder and a soft-decision Viterbi decoder [1]. This combination is effective in reducing errors in high noise conditions. The Viterbi algorithm is well-defined and its performance in an AWGN channel is known. The algorithm is affected by several parameters: code rate, code length, polynomial coefficients, and the length of its trellis. In the modulation scheme analysis, an allowance for half-rate codes was made in the choice of multi-level modulation. Thus, the remaining parameters of the code can be varied freely to provide the desired performance tradeoffs.

The correct functionality of the Viterbi decoder will be verified by comparing it to recorded results of industrial Viterbi decoders using the same parameters. A priority in the design will be to optimize for memory utilization and performance.

2.4.3 Packets

The purpose of the packeting block is to establish transmission synchronization and data flow control by adding header information into the data streams. The main motivation of this block is to improve the reliability of the modem data transmission. Without this process there is no guarantee that all the packets sent to the receiver will be received. This is a problem for our application of the modem because every packet is crucial in our transmission session. Adding this packet encapsulation block will improve data error rate and accommodate interruptions in transmission by establishing "hand-shaking" of the sender and the receiver.

The two commonly used networking protocols researched are User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). The design of the packeting scheme is loosely based on TCP. The essence of the packeting block is to allow the sender and receiver to keep track of the status of the transmission session when there are errors or interruptions. This is done through the use of the "hand-shaking" mechanism.

The two major reasons for having packets are to achieve transmission synchronization, and allow "hand-shaking" between the sender and the receiver. On the sending end, the packet creator will encapsulate blocks of signal data and encapsulate them with meaningful information. The receiver will look for a synchronization sequence from the input stream to look for the beginning of the packet. Once found, the parser will parse the packet and confirm with the received check sum to see if the transmission was error free. If so, the receiver will pass this information to its packet creator and send back an acknowledgement to the sender. If the transmission contains errors, the receiver will wait for the sender to resend the information after an application specified time out period.

3 Detailed Design

The design details of the various hardware and software blocks of the project will now be presented. This section is based off of the Detailed Design deliverable, but covers some additional implementation details and has been updated to include changes made since the deliverable was written.

While effort is made to make the descriptions of the design understandable without significant special knowledge, some basic knowledge of hardware and software design as well as fundamental communications system terminology is assumed.

3.1 Hardware Design

In addition to the four blocks that fall under the hardware section of the modem, an implicit block – the power supply – must be included. Hence, in total, there are five blocks within the hardware section, namely the power supply, serial interface, Digital Signal Processor (DSP), audio codec, and the audio interface. This section will examine the critical design decisions made in each of these sub blocks, as well as presenting some of the board layout considerations. The final circuit schematic diagrams and board layout can be viewed in Appendix B and Appendix C, respectively. The complete hardware bill of materials can be found in Appendix E. A high-level view of the hardware is shown in Figure 9.

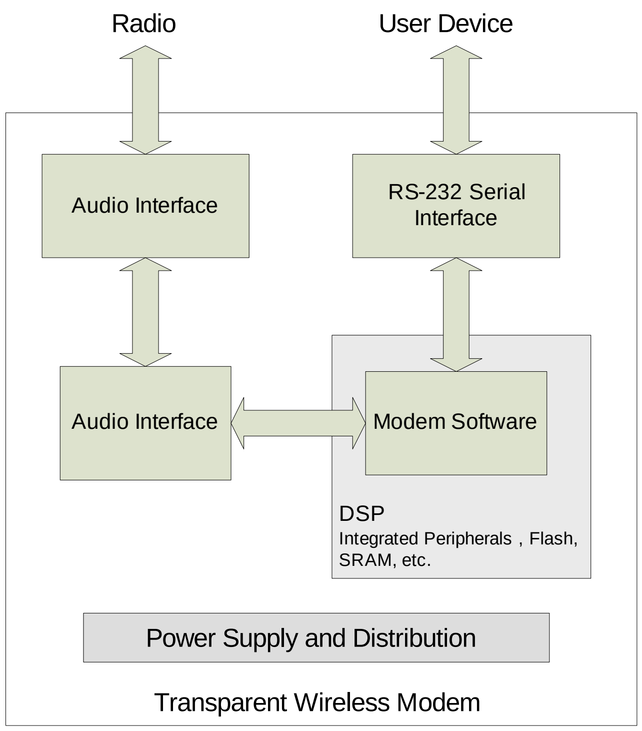

Radio
User Device
Audio Interface
RS-232 Serial Interface
Audio Interface
Modem Software
DSP
Integrated Peripherals , Flash, SRAM, etc.
Power Supply and Distribution
Transparent Wireless Modem

Figure 9 – High-level hardware design

3.1.1 Power Supply Circuitry

At the time of design, the top priority of the power supply circuit is reliability. Since the radio modem is not to be designed primarily for mobile applications, it was decided to sacrifice board space, part count, and power efficiency for reliability and redundancy. The general approach taken to design the power supply circuit was to first decide on the components to be used on the radio modem, then based on each component’s voltage and current requirements, the proper voltage supply components were chosen with ample headroom.

After all components were selected, it was decided that one 1.8V power supply and two 3.3V power supply ICs were required. Since each 3.3V power supply is capable of supplying 1.5A of current, it was predicted that only one 3.3V power supply would be sufficient. However, since there are digital and analog components in the complete design, it is better to separate the digital power supply from the analog power supply in order to prevent noise from the digital components affecting the analog components. Hence it was decided to use one 3.3V power supply for the analog components and one 3.3V supply for the digital components, along with a 1.8V supply for the DSP's core power supply.

There are a few special design considerations for the power supply for the DSP. The power up sequencing and reset signal control are important. Two voltages are used in the DSP – 3.3V and 1.8V – for the input/output buffers and for the DSP core logic, respectively. At power up, the DSP requires the 3.3V power supply to be established first before the 1.8V supply takes effect. As a result, a special IC was required to ensure that there is a delay for the 1.8V power supply after the 3.3V power supply is detected. Aside from power-up sequencing, the reset signal for the DSP and the audio codec needs to be properly controlled. In order to implement the above functions, a special reset control IC was selected in conjunction with the 1.8V power supply IC to provide the power-up sequencing function as well as the reset signal control for the DSP and the audio codec.

3.1.2 Serial Interface

From the block verification document, it was decided that the hardware should have two serial ports, so as to facilitate software debugging during the software development phase. In order to allow two serial ports to function simultaneously, a dual input/output serial driver suitable for the RS-232 protocol was selected.

3.1.3 Interconnections with the Digital Signal Processor

Almost all connections made with the DSP were based on the manufacturer's specifications. The only major design decision made regarding the DSP was that an array of LEDs connected to the DSP's general purpose input/outs (GPIO) pins will be very useful in debugging any software issues. In total, the DSP provides 32 GPIO pins. For debugging reasons, it would be excessive if all 32 bits GPIO pins were used for driving LEDs. Since one ASCII character can be represented by one byte, it was decided that it would be sufficient to use eight out of the all 32 GPIO pins for debugging purposes. These GPIO pins cannot supply large currents. In order to drive eight LEDs, it was necessary to use a high-current octal buffer to supply the necessary current.

3.1.4 Audio Codec and Audio Interface

Similar to the case for the DSP, most interconnections made for the audio codec were based on the specifications given by the manufacturer. The only relevant design decision was to use external opamps in the audio interface rather than the internal opamps provided by the audio codec. The main reason for this is because the output signal from the audio codec is in differential mode, whereas the input to the transmitting radio module needs to be single-ended. As such, it is necessary to transform differential signals to/from single ended signals externally. Coincidentally, differential to single-ended conversion can be implemented with signal amplification. As such, the opamps were used in such a way that they can achieve both operations at the same time.

A notable design decision made on the audio interface was that each input and output pair was duplicated exactly, such that there are two input and two output connections available. The motivation was to minimize reconfiguration time during testing. Since each input and output opamp circuit has a potentiometer for adjusting the gain to and from the radio modules, if only one input-output pair was present, then repetitive adjustments will need to be made each time a testing configuration is changed. For instance, if at one setting a group member needs to connect to a computer to generate or record audio signals for analysis, then the opamp gain settings will need to be changed when he/she needs to test the radio modem on actual radios. By having two separate input-output opamp circuits, a developer may switch between a specific opamp gain setting quickly and easily.

3.1.5 Design Considerations for PCB Layout

After the necessary circuits were designed and the schematic diagram was drawn, the actual circuit board needed to be laid out manually. Three specific designs goals are worth mentioning here.

First, an attempt was made to separate the digital components from the analog components, so as to minimize noise coupling between the two. When digital signal switch, the voltage or current changes very rapidly. If analog components are placed near the digital components, the digital components are capable of inducing a high-frequency noise component on the analog signal. In order to minimize this effect, the digital components should be placed together and far away from the analog components. Additionally, analog signal traces should not be placed in parallel with digital traces, so as to avoid crosstalk and signal coupling.

Second, a common and large ground fill should be used in between traces to minimize crosstalk and signal coupling. In addition to the separation of digital and analog devices, a common ground in between traces should minimize any crosstalk that may exist between traces, ensuring the best signal integrity on the design.

Third, decoupling capacitors should be placed close to the device in question. Decoupling capacitors should be used throughout the digital section, and to some degree in the analog section, to ensure that the voltage supply will be well regulated during rapid transitions or transients.

If the printed circuit board is designed with these guidelines, the final board should have good signal integrity and minimum noise levels.

3.2 Software Architecture Design

The DSP software project is clearly divided into two parts: the lower-level firmware and the modem software. A high-level overview of the architecture is shown in Figure 10. The firmware is responsible for performing input/output operations and controlling hardware peripherals. The modem software performs all modem functions. The debug interface produces a command-line interface that allows the developer to control the modem through the serial port.

The modem software is platform independent; it can be executed on the PC, on the current hardware platform or another platform in the future. The ability to run the modem on the PC is important because it allows the modem to be debugged in a familiar PC environment. Because there is minimal need to debug the modem in hardware, there is no need to purchase in-circuit debugging tools, which are relatively expensive.

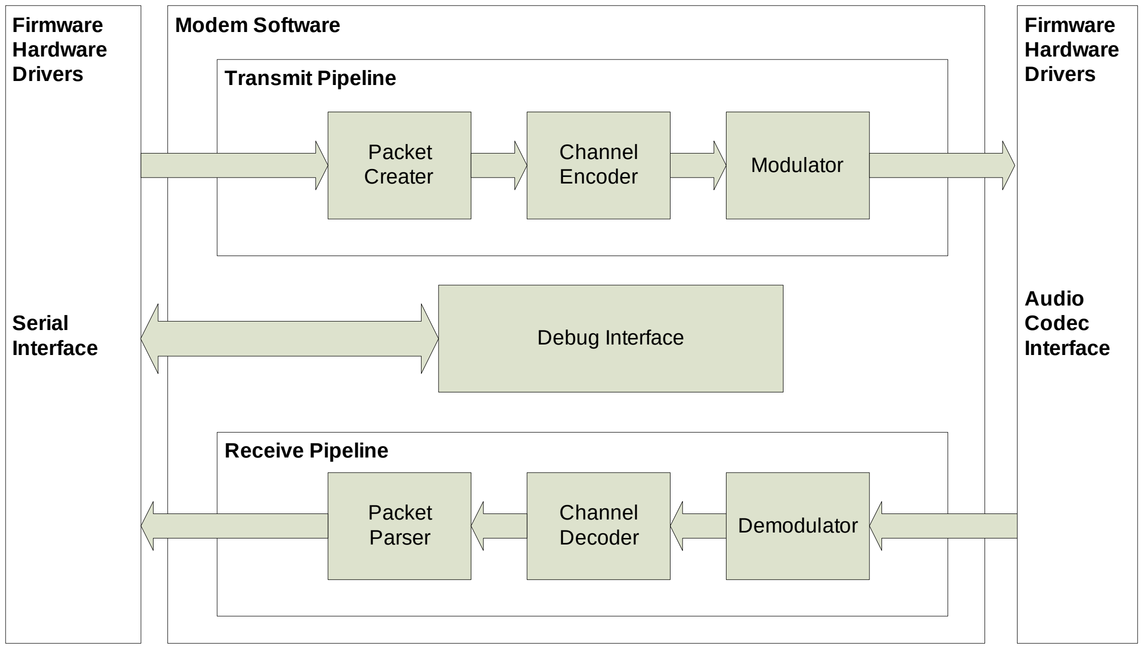


Figure 10 – Software Architecture Overview

The modem software is divided into many pieces through an explicit object framework specially designed for this project. This object framework allows the development of each modem block independently while structuring each block in a way that they can fit together perfectly.

Another advantage of the object-oriented design is the ability to instantiate multiple modems in simulation. This makes it possible to perform simulations of point-to-point modem connections in the PC environment.

The top-level objects in the modem software are pipelines, which control the flow of data from one modem block to the next and arbitrate any conflicts between them. The interface and function of modem blocks within the pipelines are discussed in a later section.

3.2.1 Firmware Interface

In order to make development of the firmware simple and efficient, a well-defined interface between the modem software and the individual DSP hardware blocks must be defined. This is the purpose of the firmware. While numerous on-chip peripherals are present, only a few are actually required for this design. Drivers must be written for the UART serial ports used for RS-232, the synchronous serial port used to interface to the audio codec, and the GPIO ports used to drive external hardware such as LEDs. Additionally, drivers are required for the hardware timers as well as to configure the DSP for proper operation after reset. Clear definition of this interface is important so that team members can simultaneously work on both sides of this dividing line without problems. Table 8 details the defined functional interface.

Table 8 – Hardware interface functions organized by peripheral

Peripheral	*Interface Functions*
System Initialization	`HwInit(…)`
Asynchronous Serial (UART)	`HwUartOpen(…)`
	`HwUartClose(…)`
	`HwUartRd(…)`
	`HwUartRdAvail(…)`
	`HwUartWr(…)`
	`HwUartWrAvail(…)`
Audio Codec	`HwCodecOpen(…)`
	`HwCodecClose(…)`
	`HwCodecSetRxPGAGain(…)`
	`HwCodecSetTxPGAGain(…)`
	`HwCodecSetRxChannel(…)`
	`HwCodecSetLoopback(…)`
	`HwCodecRdAvail(…)`
	`HwCodecRd(…)`
	`HwCodecWrAvail(…)`
	`HwCodecWr(…)`
GPIO	`HwGpioSet(…)`
	`HwGpioClear(…)`
	`HwGpioSetOutputEnable(…)`
	`HwGpioClearOutputEnable(…)`
	`HwGpioRead(…)`
Hardware Timer	`HwTimerConfigure(…)`
	`HwTimerEnable(…)`
	`HwTimerDisable(…)`
	`HwTimerGetCurrentTicks(…)`
	`HwTimerGetTicksPerMS(…)`

Rather than expose the details of interrupts and hardware buffering, a clean and consistent interface is maintained across both the UART and the audio codec interface. Read and write functions are available that provide a non-blocking polling interface to the transmit and receive FIFO (First-In-First-Out) buffers of each port. The drivers perform all interrupt servicing completely transparently. This allows the main application code to reside in an infinite loop while periodically polling the ports for data, processing and sending data as necessary.

Hardware initialization upon reset is accomplished with a single function call. Complete UART functionality is encapsulated in a file-like interface with open, close, read, and write functions. The audio codec is represented with a similar file-based interface. Additional functions allow control of the programmable gain amplifiers and other special features. The GPIO pins are supported through set, clear, and read operations. Finally, the hardware timer can be configured, enabled, disabled, and polled for the current clock tick count.

While this overview presents the functional aspects of the firmware interface, please refer to Appendix D for the complete specification of this interface in the form of a C header file.

3.2.2 Debug Interface.

There are two serial ports available on the hardware platform, and one is dedicated to the debug interface. The debug command parser, shown in Figure 11, provides a command-line interface to access modem objects in the software framework. The command structure used by the parser is created using a structured array and several command structures that can be layered on top of each other. After a command match is found, multiple parameters in hexadecimal or decimal form can be parsed and passed into the function associated with that command.

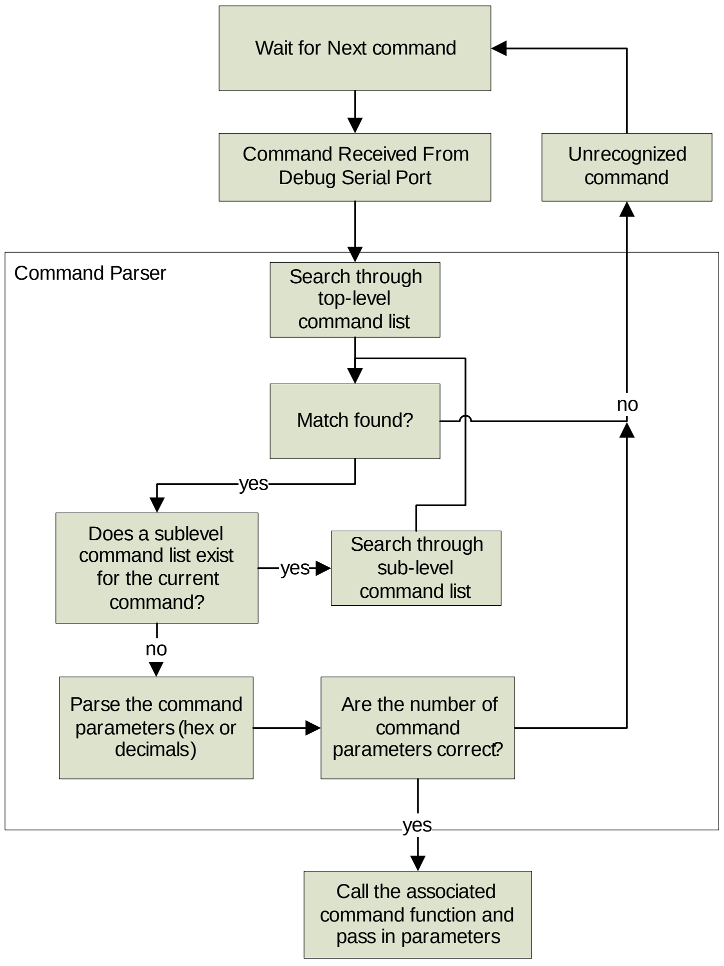

Wait for Next command
Command Received From Debug Serial Port
Unrecognized command
Command Parser
Search through top-level command list
Match found?
no
yes
Does a sublevel command list exist for the current command?
yes
Search through sub-level command list
no
Parse the command parameters (hex or decimals)
Are the number of command parameters correct?
yes
Call the associated command function and pass in parameters

Figure 11 – Command parser flow chart

3.2.3 Pipeline Framework

As shown in Figure 12, all modem blocks are wrapped within a controller object. The execution of the block and the flow of data from one block to the next are arbitrated by the interaction between the containing pipeline and the controller object. Note that the pipeline and controller objects in Figure 12 correspond to the pipeline and modem blocks illustrated in Figure 10. That is, the transmit and receive pipelines each contain controller objects for each modem block.

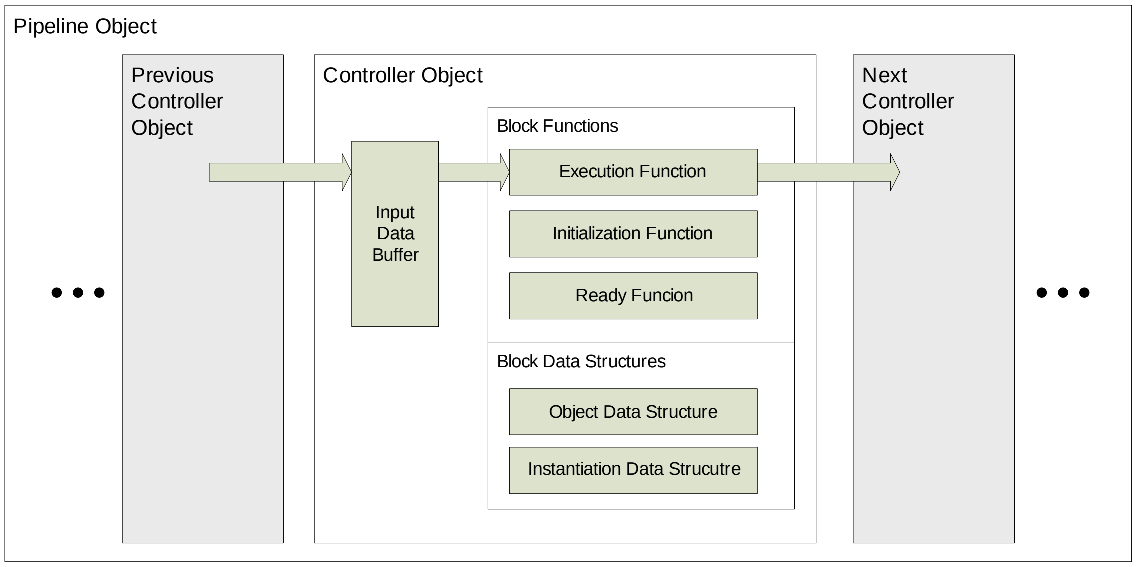


Figure 12 – Pipeline and controller objects

Each block in a pipeline implements three interface functions through its controller object: an initialize function, a ready function, and an execute function. The initialize function is called once upon startup to set all internal block state. A block is able to interact with the pipeline via the ready function, which tells the pipeline whether it is ready to be executed. Because there is an input buffer for each block, the blocks need not be executed in a linear sequence. They can execute whenever there is data available in the input buffer and when the next block's input buffer is not full.

By enforcing this interface on all software modem blocks, development and testing can proceed independently for each block. When each block is complete and tested, there is a high degree of confidence that the overall collection of blocks in a pipeline will work well together, since the coupling between blocks is kept to an absolute minimum.

3.3 Software Modem Design

This section discusses the design of the specific software modem blocks in more detail. The design decisions made as part of high-level analysis are used to guide the design and implementation decisions. Each of the blocks described here has been implemented to conform to the controller object interface discussed previously. Design and implementation techniques for the modem blocks are based off of those discussed in [1] and [4]

3.3.1 FSK Modulator

The purpose of the FSK modulator is to translate bits into tones that can be transmitted over voice-band channels. Since the decision was made to use 4-tone FSK, each pair of transmitted bits must be mapped into one of the 4 tones. Figure 13 illustrates the modulator design conceptually. Note that all signals are in discrete time since all processing is performed digitally. The sample rate *T* is the chosen sampling rate of 8 kHz and *k* is the index of the sample to be generated.

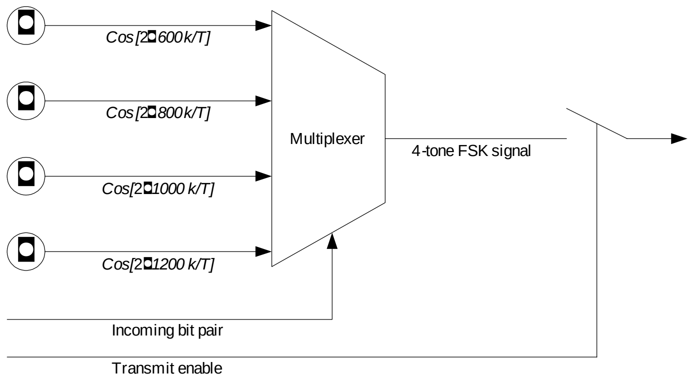


Figure 13 – FSK modulator design

Four oscillators are used to generate the four orthogonal tones (600, 800, 1000, and 1200 Hz). These tones are generated directly at the sampling rate of 8 kHz and are fed into a 4-to-1 multiplexer that selects samples from only one of the oscillators as output at any given instant. The next available bit pair from the incoming bit stream controls the multiplexer, choosing the correct tone based upon the mapping given in Table 9. This specific mapping was chosen so that the bit pairs associated with adjacent tones differ by only one bit.

Table 9 – Mapping between bit pairs and FSK tones

Bit Pair	Tone
0 0	600 Hz
0 1	800 Hz
1 0	1200 Hz
1 1	1000 Hz

The duration of each tone is one symbol period, which is 40 samples is this case since

$$T_s = \frac{F_s}{R_b} = \frac{8000}{200} = 40\,,$$

where T_s is the symbol duration in samples, F_s is the sampling rate in Hz, and R_b is the baud rate (symbol rate). The output of the modulator is only enabled when incoming bit pairs are available, thus, the possible outputs of the modulator are a complete symbol's worth of tone samples or no samples at all.

To ensure phase continuity between adjacent symbols, symbol transitions are made only when the phases of the Cos[…] terms in Figure 13 are equal zero or a multiple of 2π. This corresponds to *k* values that are multiples of the symbol length, 40 samples, and guarantees continuous, smooth output waveforms. The resulting output of the modulator when presented with input data in the pipeline is a phase-continuous 4-tone FSK signal. No pulse shaping is performed before modulation.

Software implementation of the modulator is straightforward since the 40 samples for each tone can be pre-computed and stored in memory. Modulating a given bit pair is then just a matter of reading and outputting the sample values for one symbol from memory. This is repeated for all available bit pairs for transmission. If no bit pairs are available, no samples are output from the modulator.

3.3.2 FSK Demodulator

The purpose of the FSK demodulator is to receive voice-band FSK tones and translate them back into the original transmitted bit pairs. This problem is made more difficult by the fact that symbol timing synchronization must be recovered at the receiving end before the signal can be decoded correctly. Figure 14 illustrates the high-level structure of the FSK demodulator. All signals are in discrete time, where indices of *k* indicate a signal updated at the sampling rate of 8 kHz, while indices of *n* indicate a signal updated at the symbol rate of 200 Hz.

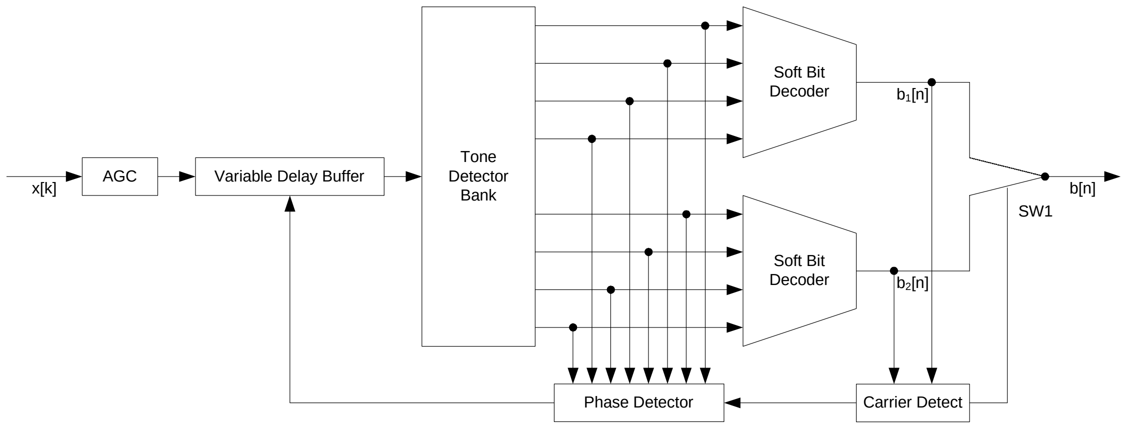


Figure 14 – Structure of FSK demodulator

Incoming audio samples are first passed through a simple auto gain control (AGC) block that is used to normalize the sample levels about a set operating point. This is important so that subsequent processing can make assumptions about signal amplitude that will be valid for a number of conditions. The samples are then passed through a variable delay buffer that aids timing recovery by providing the necessary sample shifts in symbol timing as directed by the phase detector. The time-shifted input is passed through a bank of tone detectors to produce 8 tone envelope signals. Each set of 4 signals corresponds to the output of 4 tone detectors. Two sets are provided corresponding to two different sampling instants, which are required to estimate the received symbol timing.

The output of the tone detectors is fed into two soft-bit decoders that map the four tone levels into two soft bits. Each soft-bit is an integer between –7 and 7 indicating how confident the decoder is that the bit was either definitely a one (7) or a definitely a zero (-7). Values between these two extremes indicate relative confidence. Using a soft bit instead of the usual hard bit (either a 0 or a 1) improves the performance of the channel decoder.

The output of the demodulator is selected by a switch (SW1) that is controlled by the carrier detect logic. The carrier detect monitors the filtered average soft-bit quality of the upper and lower soft-bit decoders, moving the output switch to select the better of the two. Once the demodulator has achieved symbol synchronization, the output switch position remains fixed.

The tone detector bank is shown in more detail in Figure 15. A classic in-phase and quadrature demodulator is used to downshift each tone to 0 Hz [1]. A matched filter $h_m[k]$ is then applied to pass only the desired tone. Since the FSK modulator uses square data pulses and no pulse shaping is applied, the matched filter is simply the rectangle described by

$$h_m[k] = \begin{cases} 1 & 0 \le k < T_s \\ 0 & \text{otherwise} \end{cases},$$

where T_s is the symbol length in samples, or 40 in this case. After filtering, the sampling rate is reduced by a factor of 20 to leave two samples per symbol, differing in time by half a symbol. The magnitude of these complex (in-phase and quadrature) samples is taken to produce the tone envelope for each of the four tones. The 4 outputs at two samples per symbol each are demultiplexed to produce 8 outputs at one sample per symbol each, where the upper and lower set of outputs differ in time by one half symbol.

Each of the soft-bit detectors applies a simple mapping to produce the output softbits. This mapping essentially reverses that given in Table 9:

$$\begin{aligned} bit_0 &= \pm \log_2(|y_{800} + y_{1000} - y_{600} - y_{1200}|) \\ bit_1 &= \pm \log_2(|y_{1200} + y_{1000} - y_{600} - y_{800}|), \end{aligned}$$

where y_{600}, y_{800}, y_{1000}, and y_{1200} are the four tone envelope signals. Base-two logarithms are used to compress the possible range of output into values that can be mapped into a 4-bit quantization between –7 and 7. Note that the sign of the soft bit is re-applied after the logarithm of the absolute value is taken.

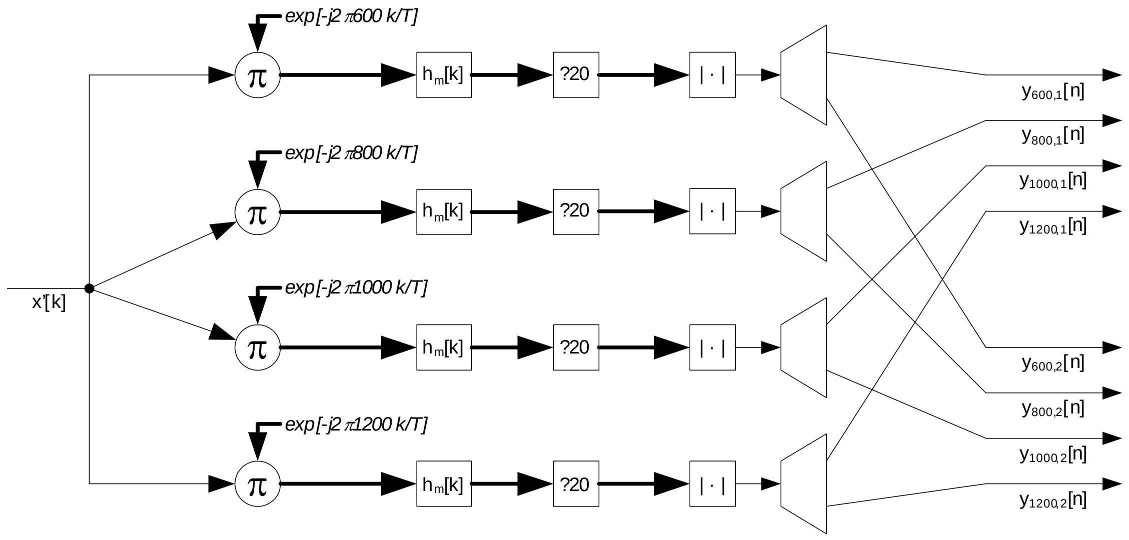


Figure 15 – Demodulator tone detector bank

Symbol timing recovery is achieved by using a simple early-late gate phase detector [1]. This technique requires two samples per symbol and is illustrated in Figure 16. The triangular shape is the envelope of a demodulated symbol, with the peak indicating the ideal sampling instant. The early and late samples are one half-symbol away from this point on either side. The amplitudes of the these two samples is used to derive a phase estimate:

$$phase = K(y_{early} - y_{late}),$$

where K is gain used to tune the detector performance, and y_{early} and y_{late} are the early and late sample amplitudes. This phase difference is low-pass filtered and then fed back to the variable delay buffer in Figure 14, closing the control loop. The K of the phase detector is adjusted to ensure stability and a reasonable closed-loop settling time.

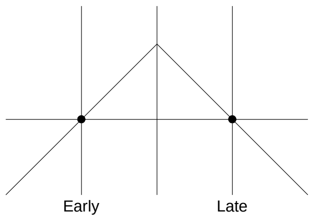


Figure 16 – Illustration of early-late gate phase detector

3.3.3 Channel Encoder

The Viterbi channel encoder will process input data and produce a convoluted bit stream as bit output. The purpose of the block is to add forward error correction to the bit stream. The receiver must use a matching Viterbi decoder to correct errors and return the original data stream.

The output rate of the encoder can be modified at runtime to produce output streams of varying bandwidth to adapt to the signal noise level of the environment. The codec parameters are configurable at run-time, which allows for easy testing. The results are shown below. The BER (Bit Error Rate) vs. SNR graph is normalized for different data rates (which some books usually omit). The constraint length parameter controls the length of the trellis. Increasing the length reduces the error rate, but increases the computational requirements for decoding exponentially. In our final implementation, the default bit rate is set to 1 / 2 and the constraint length to 3. Using this setting offers good error correction without sacrificing too much transmission speed.

During beginning of transmission and end of transmission, the encoder is designed to add bit pads. Without this precaution, the head and tail of transmissions would be especially susceptible to errors because they are not protected fully by the convolutional code.

The encoder is relatively easy to implement compared to the decoder. At each input bit, the encoder convolutes the bit history with a defined bit pattern which minimizes correlation to produce 2 output bits (because we chose to use 1 / 2 bit rate). The algorithm is very simple, and is O(n) complexity. The memory and computational resources required are minimal.

3.3.4 Channel Decoder

The Viterbi channel decoder implemented uses the Viterbi algorithm (hence the name) [1]. It is almost universally used for small constraint lengths because it provides maximum likelihood performance and is highly parallelizable (although this characteristic does not benefit this project). Other sequential algorithms could be used for higher constraint lengths; however, its results are not maximum likelihood. For this project, we developed a unique implementation of the Viterbi algorithm that is fast and memory efficient.

The encoder can be though of a state machine whose state is defined by the bit history. At each new bit, the state can branch 2 ways. A ‘1’ branch and a ‘0’ branch will produce different output bits (via convolution of bit history). All the possible states over a sequence of bits can be represented by a trellis. The Viterbi algorithm is a method of solving from these output bits, the most likely path through the trellis. The resultant path which contains the branches can be used to reconstruct the original bit stream. The Viterbi algorithm finds the maximum-likelihood path by keeping most-likely paths in memory, and discarding duplicate paths as new bits arrive.

To minimize memory usage, the paths are bit packed into memory. Our implementation requires the storage of two paths in memory (current and previous) to optimize for performance. As data is received, the maximum likelihood path through the current trellis is constructed. That information is used to pick the most likely path from the previous trellis.

The decoding process is complicated by the fact that at the beginning of transmission and the end of transmission, the data generated by the FSK demodulator may be garbage. Because convolutional output bits are based on a bit history, these garbage bits will increase the BER of bits close to the beginning and end of transmission. The solution is to have pad bits are encoded at the beginning and end of transmission.

Because the hardware platform was not completed during the design phase, a PC simulation environment was created. The environment simulates an AWGN channel to measure the codec performance.

The results are shown below in Figure 17. The BER (Bit Error Rate) vs. SNR graph is normalized for to the data rate. In the chart, the n and k parameters represent the output data rate and constraint length parameters respectively. To represent the performance more practically, the BER rate is normalized by the data rate parameter.

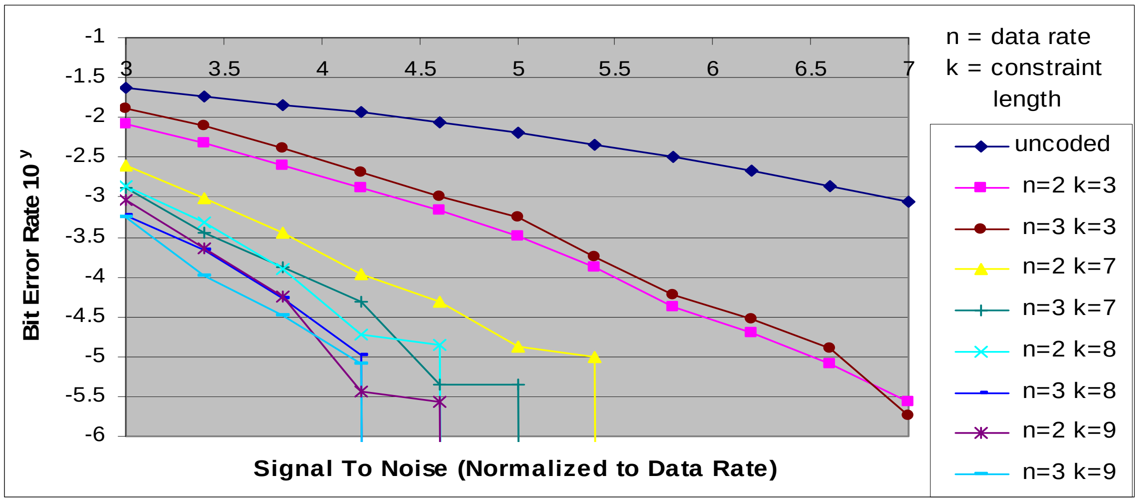


Figure 17 – BER vs. SNR for Soft-Decision Viterbi Convolutional Codec

BER performance increases for larger constraint lengths. This parameter costs nothing in bandwidth but it's computational and memory requirements increase exponentially. The simulation time was used as a measure of computational intensity, and it is shown below in Figure 18.

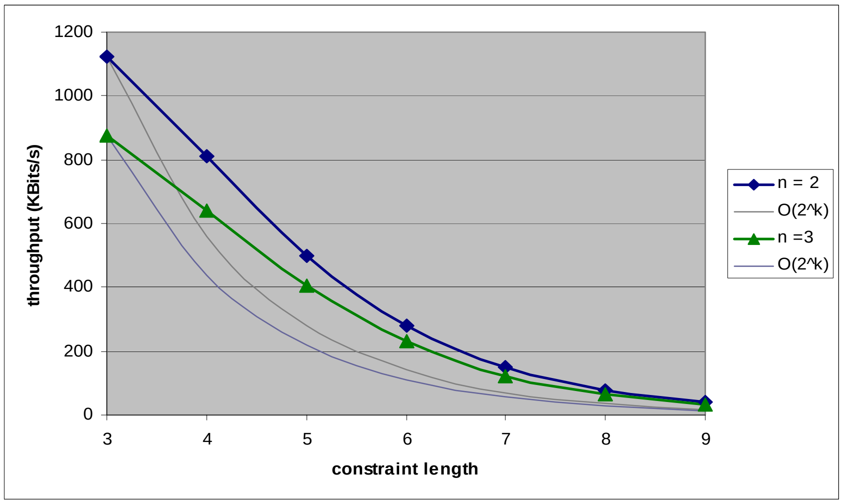


Figure 18 – Codec performance vs. Constraint Length

The Viterbi algorithm implemented by our code was tested in simulation first and then in actual hardware. It is proven to decrease bit error rate and uses minimal computational resources.

3.3.5 Packet Creator

The purpose of the packeting block is to create flow control and data reliability by encoding header information into each transmitted packet by data encapsulation. The block was implemented because the advantages gained from the packeting block, such as flow control and data reliability significantly out weighs the disadvantages from the added complexity since data integrity is a must for our application. The challenge of the implementation comes mostly from transmission synchronization and "hand-shaking" between the sender and the receiver.

This specific design of the packet encapsulation block is used loosely based on the TCP transmission control protocol. This type of encapsulation scheme is proven to be effective in other modems. The protocol was chosen for its simplicity and ease of implementation to ensure a working product for our primary design objectives. The sender and receiver will each have a packet encapsulator (creator) and a decapsulator (parser).

The three major functions of the packeting block is to achieve transmission synchronization, create the packet by encapsulating the data stream with header information on the sending end, and parse the received bits to retrieve the original data stream. Based on the state of the Packet Parser, the Packet Creator decides what type of header information to use and its behavior. The Packet Creator is responsible for creating data packets as well as creating acknowledgement packets to let the sender know it has received the packets correctly. Table 10 shows the contents of a packet and Table 11 details the header information.

Table 10 – Packet format

Header Information	Data	Checksum
1 or 2 words	up to 16 words	1 word

Table 11 – Header format

Sync Bits	SEQ. #	Ack Flag	Ack ID (optional)
11 bits	4 bits	1 bit	4 bits

The functions of each part of the packet content are listed below.

Synchronization bits (Sync. Bits) are a bit pattern added to the beginning of every packet by the Packet Creator. This allows Packet Parser to use bit matching to find the beginning of packets in an input stream. The parser will continue searching the input stream until the end of the input stream.

Sequence number (SEQ. #) and acknowledgement number (ACK. #) are used to implement "hand-shaking" between the sender and the receiver. Each octet of data will be given a unique sequence number. The receiver will send an acknowledgement number of the received sequence number plus one to the sender in order to notify the sender that it has successfully received the block of data and is anticipating the next block of data. When the sender receives this acknowledgment, it will then send the next packet. This cycle will happen continuously until the end of the transmission. This is also called "handshaking" and keeps the transmission synchronized. In the case of a packet dropped, the handshaking mechanism is to aid the recovery algorithm.

The Acknowledgement bit is set to one when the packet is an acknowledgement. This bit enables a second header word which contains the acknowledgement number and information regarding the size of the acknowledgement packet.

The Check Sum field is used to ensure data integrity during transmission. If the check sum and the sum of all the bits received are not equal, the packet is discarded. Because we cannot distinguish between a corrupted packet and a falsely detected sync, the receiver will not initiate a resend.

3.3.6 Packet Parser

The two functions Packet Parser is responsible for is to achieve transmission synchronization and parse the received packet for meaningful header information.

The parser will search for the synchronization bits by shifting the input stream one bit at a time until a match. Once synchronized, the block will begin parsing the header elements. The Packet Parser uses a state machine to remember the position in the packet based on received data. A correct packet is verified by calculating and matching the check sum. The Packet Parser will notify the Packet Creator and send back an acknowledgement to the sender.

We have also picked a simple transmission scheme for our primary design objectives by sending and receiving one packet at a time. In order to minimize the effect of the large, physical delay caused by the switch pushed to change the radio from sender to receiver, resend time out and the delay caused by “hand-shaking” between the sender and the receiver, the packet size is made as large as possible to maximize throughput and minimize resends.

The Packeting block assumes a low bit error rate, supported by forward error correction. With high error rates the transmission would become saturated with resend requests.

Duties and Responsibilities

The various responsibilities for each team member are listed in Table 12. It is important to note that the team discusses all design objectives together and the responsibilities are not exclusive. In addition, this listing does not give credit for all the many smaller contributions that each member has made. Overall, it is believed that a fair division of labor was reached based upon each member’s skills and level of experience.

Table 12 – Responsibilities of team members

	Stefan Janhunen	Shu Wu	Vincent Liu	Aaron Cheung
Hardware Design	●			●
PCB Layout				●
Hardware Assembly				●
Radio Characterization	●			
Firmware	●			
Debug Interface		●		
Modem Architecture		●		
Channel Encoder/Decoder		●		
Packet Creater/Parser		●	●	
Modulator/Demodulator	●			
Business and Administration			●	

4 Experimental Results

The working results of the prototype will now be considered. Two aspects will be presented and discussed: the prototype demonstration and bit error rate performance measurements. The prototype demonstration was presented to our consultant in the previous academic term. The bit error rate measurements are unrelated to the prototype demonstration, but were made as an additional verification of the software modem performance.

4.1 Prototype Demonstration

The prototype checklist is based heavily upon the original project requirements. It itemizes the key requirements that the prototype must meet to be considered successful. Table 13 lists the function requirements from the prototype checklist.

Table 13 – Prototype checklist functional requirements

Priority	Name	Description
MUST	Radio Interfacing	Modem has 3.5mm audio connectors for signal input and output for interfacing with audio interface of voice-band radios
MUST	Modem Output Signal Bandwidth	Modem output signals must fit within audio bandwidth of radios (300Hz to 3kHz)
MUST	RS-232 Interfacing	Must interface with other equipment over a standard RS-232 serial interface
MUST	Modem Operation Transparency	Must provide transparent operation through RS-232 and behave similarly to landline modems or null-modem connections
SHOULD	Error Correction Scheme	Modem has a forward error correction scheme to reduce error rate of the received data
SHOULD	Data Transmission Mode	Modem operates in a half-duplex mode for two-way operation over a single radio channel
SHOULD	Input Signal Amplitude Tolerance	Can tolerate reasonable range of input signal amplitudes, from -20dBV to -3dBV (NOTE: This has been revised slightly from initial specification)
SHOULD	Output Signal Amplitude Range	Can produce reasonable range of output signal amplitudes, from -20dBV to -3dBV into a 600Ω load (NOTE: This has been revised slightly from initial specification)

These requirements are a direct reflection of the original functional requirements set out in the project's specification. All of these requirements were met by design and verified in the prototype demonstration. The interfacing and signal bandwidth requirements ensure that the modem can be electrically connected to the chosen voice-band radios and that its output tone frequencies are compatible with the voice bandwidth of the radio. The RS-232 interface and transparency requirements ensure that a pair of modems connecting two devices can be used as a virtual serial cable between the devices. That is, no special modem commands are sent to the modems in addition to the serial data. The modem's channel encoder and decoder fulfill the forward error correction requirement. Half-duplex communication is supported in hardware by allowing the modem to trigger the transmit function of the voice-band radio. This way, either end of a link can transmit or receive, but only in a turn-by-turn fashion.

The last two items of Table 13 were modified slightly from the initial project requirements: the input and output signal amplitude levels were considered a non-critical requirement and were changed to reflect actual radio and hardware performance. The initial project requirements had listed the amplitude range as 0 dBV to –20 dBV. After constructing the hardware it was determined that meeting the 0 dBV requirement was barely achievable with the 3.3 V power supply in use for the audio circuitry. It was also found that signals of that amplitude were unnecessary and even too powerful for the chosen radios. Thus, the specification was loosened to –3 dBV to –20 dBV for the checklist and successfully verified in the demonstration.

Table 14 itemizes the non-functional requirements listed in the prototype checklist. These requirements specify user convenience in this case rather than directly whether the modem functions or not. All requirements here were met and demonstrated with margin.

Table 14 – Prototype checklist non-functional requirements

Priority	Name	Requirement
MUST	Size	PCB must be less than 25 cm in either direction
MUST	Weight	The modem must be less than 3.0 kg in weight
SHOULD	Power	Should not require over 6 VDC @ 800 mA
SHOULD	Portability	Should be small and light enough to be portable

Table 15 summarizes the performance requirements from the prototype checklist. These are once again based upon project requirements. These requirements specify the minimum performance metrics the modem must meet to be considered acceptable for its general areas of application. In actual fact both of these requirements were exceeded by the prototype. The average net data transfer rate achieved is 200 bits/second, including the overhead consumed by the channel coding. The bit error rate performance exceeds the minimum requirement and is considered in more detail in the next section.

Table 15 – Prototype checklist performance metrics

Priority	**Name**	**Description**
SHOULD	Net Data Transfer Rate	Must transfer data at minimum rate of 100 bits/s over voice-band radio link
SHOULD	Bit Error Rate (BER)	Must not exceed 1 bit error per 1000 bits over reasonable channel conditions (15 dB Eb/No)

The final aspect covered by the prototype checklist is modem cost. The requirement was that each modem should cost less that $300. Referring to the updated project budget in Appendix A, this requirement is satisfied, even in the low-quantity prototype units. This cost could be reduced further with a design optimized for production and the savings possible on large volume orders.

4.2 Bit Error Rate Measurements

Bit error rate is one standard metric by which modem performance is evaluated. By measuring the bit error rate (BER) for various levels of channel noise and plotting the results, it is possible to compare a given modem’s performance to the theoretically achievable performance for the modulation scheme used. Figure 19 illustrates the measured and theoretical error rate performance for this modem over an additive white Gaussian noise (AWGN) channel. Simulation was performed on a data length of 10^6 bits.

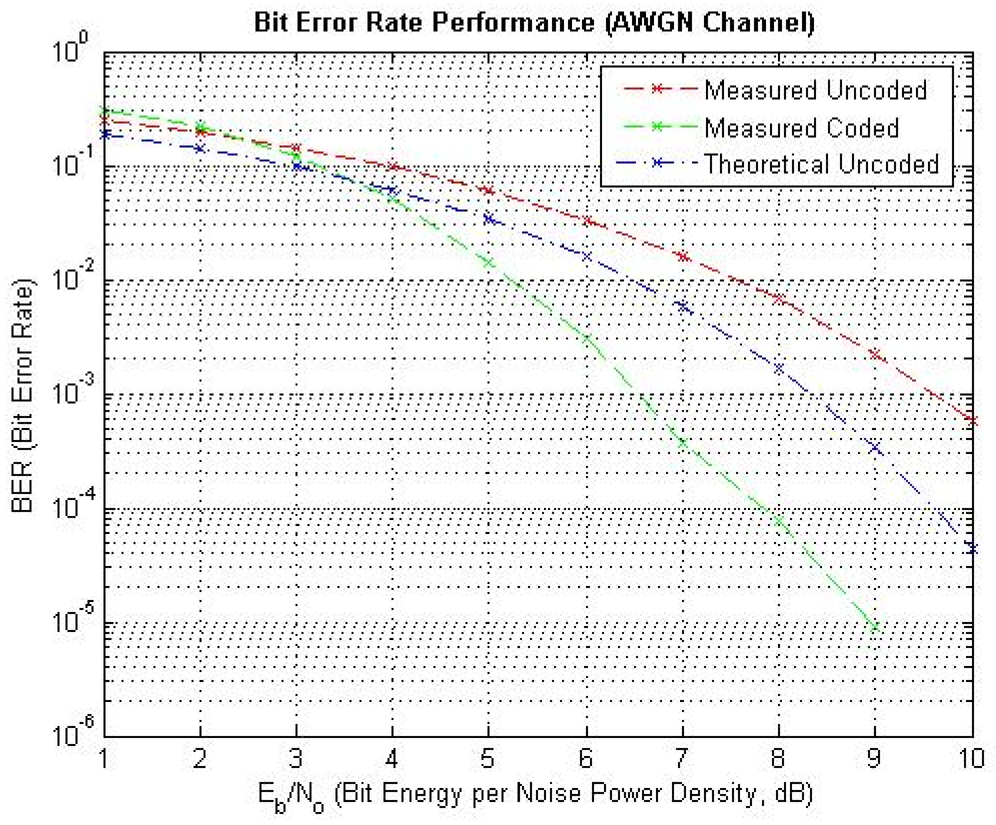


Figure 19 – Measured and theoretical modem bit error rate performance

Since the modem software is fully portable, these measurements were made running the final modem software on a PC, using Matlab to generate the noise. Data packets were not enabled for these tests, as retransmission would mask bit errors made by the receiver. Otherwise, the software modem code used for PC execution was identical to that used for execution on the prototype hardware. This was found to be more convenient than performing multiple tests on actual hardware, requiring a hardware-based noise generator and manual adjustment for each measurement. However, it should be noted that for the prototype demonstration, a simple hardware-based noise source was used to verify the performance of the modem software running on the actual prototype hardware.

Three curves are present in Figure 19. The top curve is the measured BER for the modem with the channel encoder and decoder completely disabled. That is, it directly represents the performance of the 4-tone FSK demodulator in noise. The middle curve is the theoretical performance bound for orthogonal, non-coherent, 4-tone FSK as calculated using the Matlab `berawgn(…)`command. The implemented demodulator performance is roughly 1 dB worse than the theoretical bound. No further investigation was made to understand exactly why this difference exists. However, the implemented demodulator does not have perfect symbol timing, since timing adjustments are made via closed-loop control, which is susceptible to noise. Additionally, the implemented demodulator uses fixed-point arithmetic and thus suffers from limited dynamic range. Additional research could lead to a full justification for this difference. For the present purposes, the results will be considered sufficiently good.

The bottom represents the measured BER of the modem with channel coding enabled. As expected, this performance exceeds the theoretical uncoded performance bound for 4-tone FSK. With channel coding enabled, the error rate performance of the modem improves by roughly 3 dB. This is consistent with the performance gains for the chosen convolutional code previously discussed. The last data point of this curve is not shown since no bit errors were made for the entire test run. To improve the results at low error rates, the tests could be run on a data size several orders of magnitude larger.

The modem performs well under noise. The BER limit for usability was set in the project requirements to be 1 in 1000 bits, or 10^{-3} in Figure 19. With channel coding enabled, the modem meets this requirement with E_b/N_o values as low as 6.5 dB, well below the required maximum value of 15 dB.

4.3 Experience Summary

In addition to producing a working prototype, one of the key objectives of the project was to improve teamwork skills of each member of the design team when working in conjunction with others on a complex problem. Through out the project, the team encountered various situations where members had differences in opinions. Each time, the team was able to resolve the difference in a logical and professional manner. Criticism from each member was also taken constructively.

The team was able to increase its abilities to work with other parties involved in the design projects. These parties include groups that are interested in implementing the technology for future production, possible industry sponsors, department representatives and professors.

Through out the project, the team had to constantly update its knowledge base in the area of wireless communications. Apart from seeking information from courses and the project consultant, the team also did extensive, addition research. Everyone's ability to analyze a problem logically and seek out relevant resources was improved. This is an important skill because solutions to real-world problems are not always clearly presented.

Due to the size and the length of the project, the team had to implement a strict project time line to ensure the progress of each stages of the project. Overall, the major milestones of the project were met on schedule. Members of the team also constantly motivated each other to increase the efficiency of the team.

One key for successful collaboration was the use of a software version control tool to store and back-up all hardware and software work on the project. This tool ensured the smooth operation of the team even when different members were working on the same part of the project while located in different geographical locations.

A major component of the design experience gained in this project relates to signal processing, specifically the design of the FSK modulator and demodulator. While fairly straightforward techniques were used for modulation, demodulation, and synchronization, actually implementing these algorithms in an efficient way for execution on a DSP is not as straightforward as simulating them in Matlab. The DSP supports only 16- and 32-bit fixed-point arithmetic instead of the double-precision floating-point arithmetic used by Matlab. Numerical issues such as dynamic range and saturation or overflow conditions become important. Thus all signal processing routines were first developed in Matlab to create a working reference. The C version was then implemented, optimized, and tested against this working reference. Proceeding in this method greatly reduced bugs during system integration.

A second key aspect of the project was the partition of the design into independent layers with well-defined interfaces. The hardware components, device driver firmware, and modem software were all developed independently but in a chronologically overlapping fashion. By partitioning team member responsibilities along strict functional boundaries, a high degree of independence was possible. That is, blocks worked on by different team members were minimally coupled by design. This saved time and worked out well for everyone. This technique is scalable for use in both larger projects and larger teams.

There was only one minor change made in the original design specification that was used for the prototype demonstration, namely the audio input/output signal maximum amplitude level. This change was made since it was found that the original specification was unnecessarily high and because, using the chosen hardware voltage levels, it was not quite possible to meet the original requirement. All other project requirements were met or exceeded without modification.

5 Discussion and Conclusions

While the use of voice-band modems over radio links is definitely not new, the use of a fully self-contained software-defined modem for this application is less common. Voice-band modems have traditionally played a large role in the amateur radio community, but design has usually been somewhat ad-hoc, limited to the simple, tried-and-true techniques. In this project, a fully software-based modem is designed that requires relatively few hardware components to implement.

Currently, the major limitation of this design is speed of data transfer. For this initial prototype, a very traditional modulation scheme was used (4-tone FSK). Much additional design work could be done to improve this, choosing a modulation and coding scheme more optimal in a performance sense but also more complex in an implementation sense.

To ready this design for production, it is necessary to reduce the size and cost of the hardware. All hardware debugging features would be kept to a minimum and part count would be minimized. Also, PCB board space would be optimally used to allow for the smallest possible footprint. With this in mind, it should be possible to produce small and relatively cheap hardware that runs the same or modified modem software. The modem software could be left relatively untouched for production, or parts of the software could be rewritten for improved modem performance.

In summary, this project has produced a functioning transparent wireless modem that uses voice-band radios to transmit data. By supporting all modem functions in software, the same hardware platform can be reused to run better and more efficient modem software designs in the future.

References

1. S. Haykin, *Digital Communications*, John Wiley & Sons, Inc., 1988.

2. *TLV320AIC10 Data Manual*, Texas Instruments, Dallas, Texas, 2001.

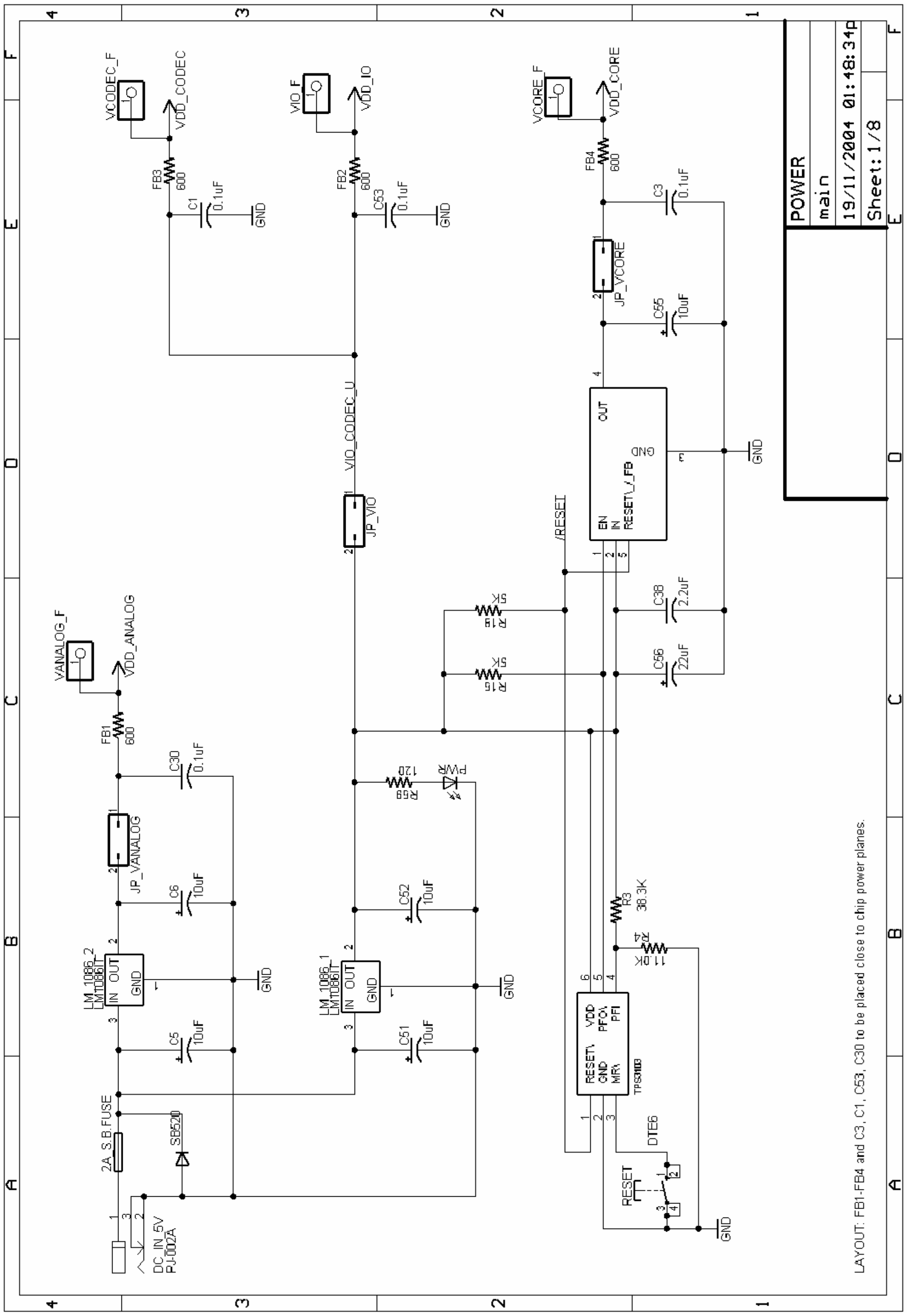
3. *TMS320F2810, TMS320F2811, TMS320F2812, TMS320C2810, TMS320C2811, TMS320C2812 Digital Signal Processors Data Manual*, Texas Instruments, Dallas Texas, 2005.

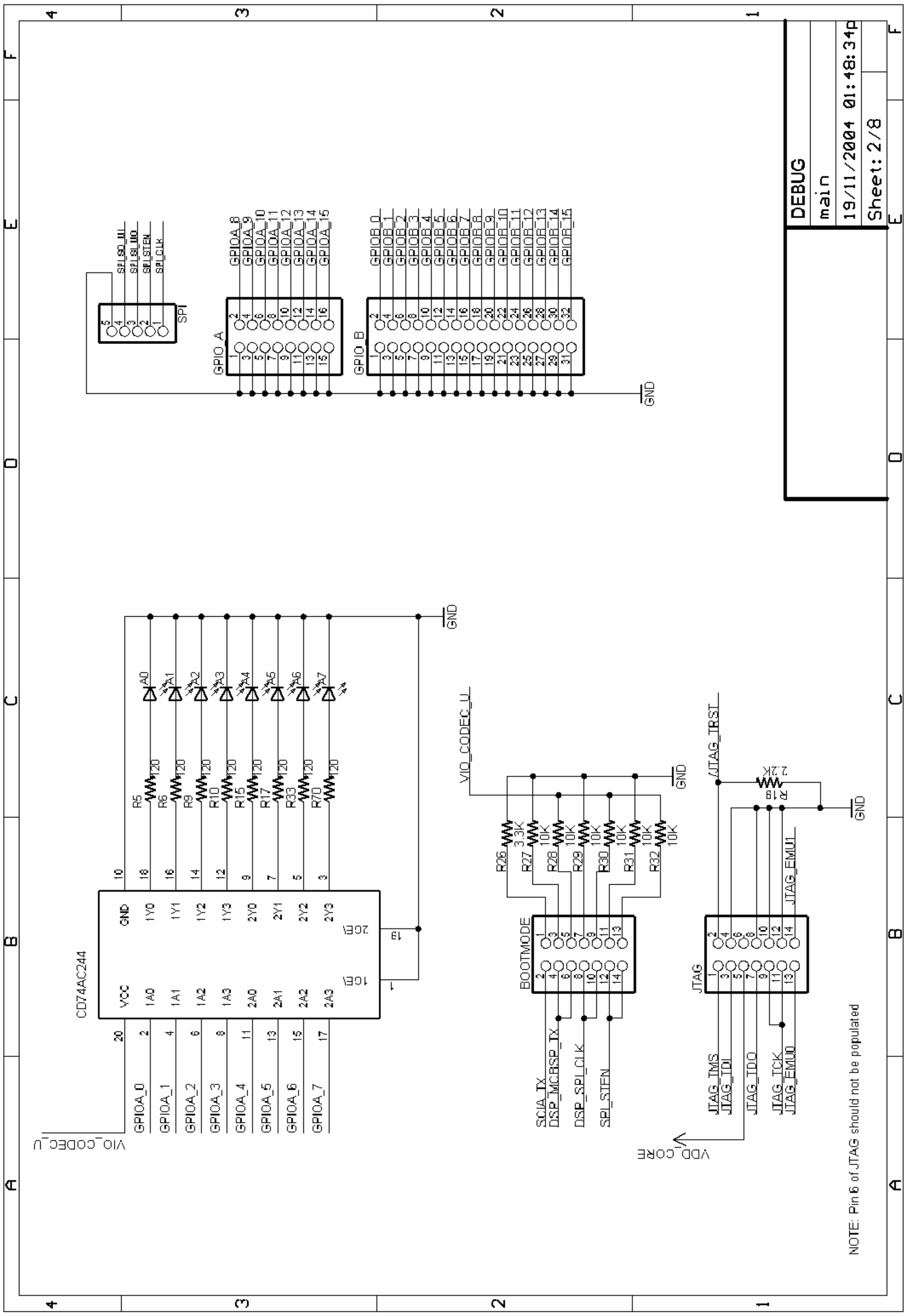
4. T. McDermott, *Wireless Digital Communications: Design and Theory*, TAPR Corporation, 1998.

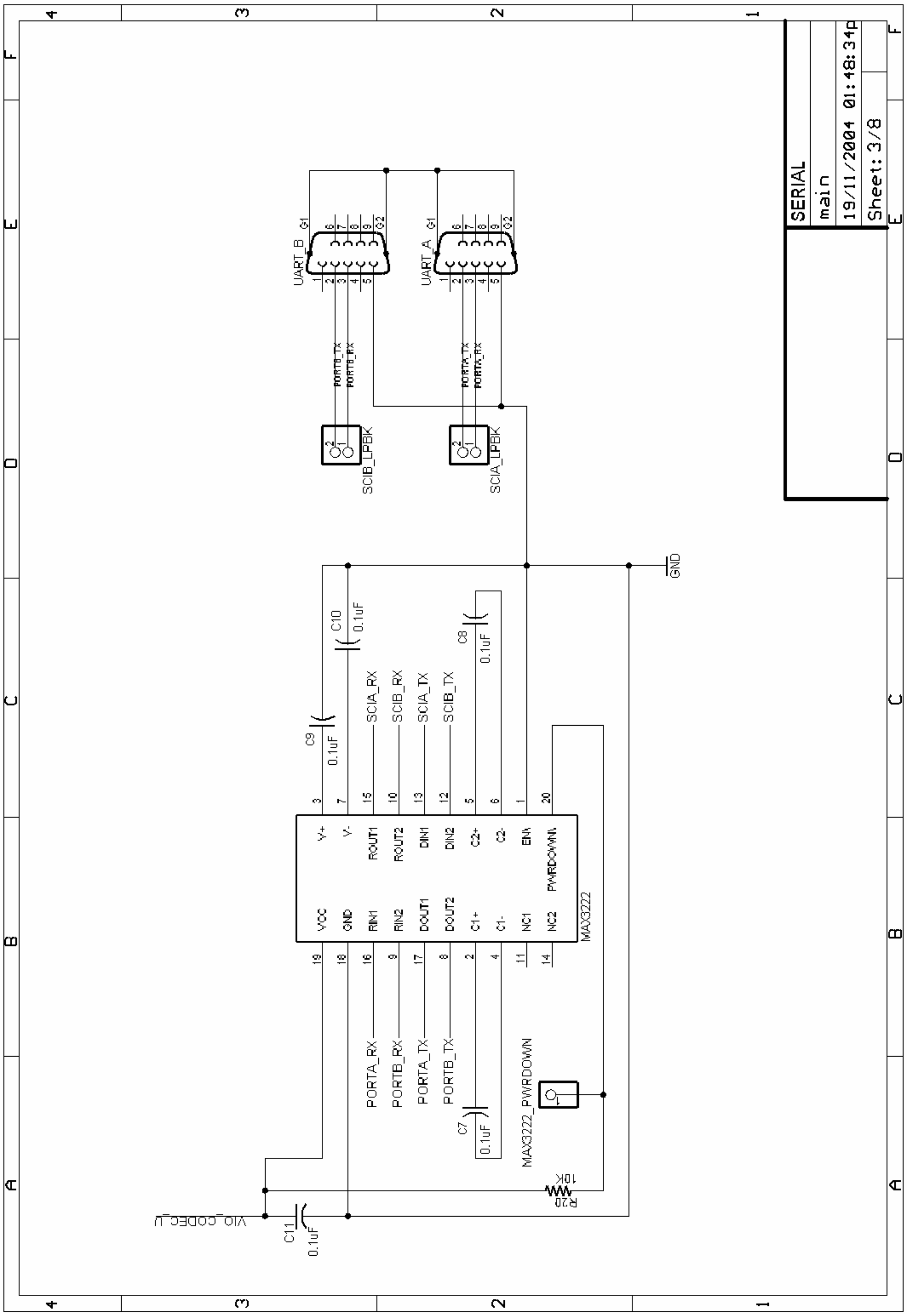
Appendix A Updated Budget

NEED						TEAM FUND
CATEGORY	ITEM	PRICE EA	QTY	UNIT	TOTAL	TOTAL
Prototype Material					$973	$973
	PCB	N/A	5	EA	$300	$300
	DSP	$36	8	EA	$292	$292
	Parts for prototype boards (excluding DSPs)	$160	4	sets	$640	$640
	1/8" 8"x8" aluminums grounding back plane	$7	4	EA	$28	$28
	Power adapters	$5	3	EA	$14	$14
	Portable radios	$70	1	pair	$70	$70
Education Material					$316	$316
	"Wireless Digital Communications" text	$64	4	EA	$256	$256
	Practice soldering kit	$30	2	EA	$60	$60
Services					$200	$200
	Poster	$200	1	EA	$200	$200
GRAND TOTAL					$1,489	$1,489

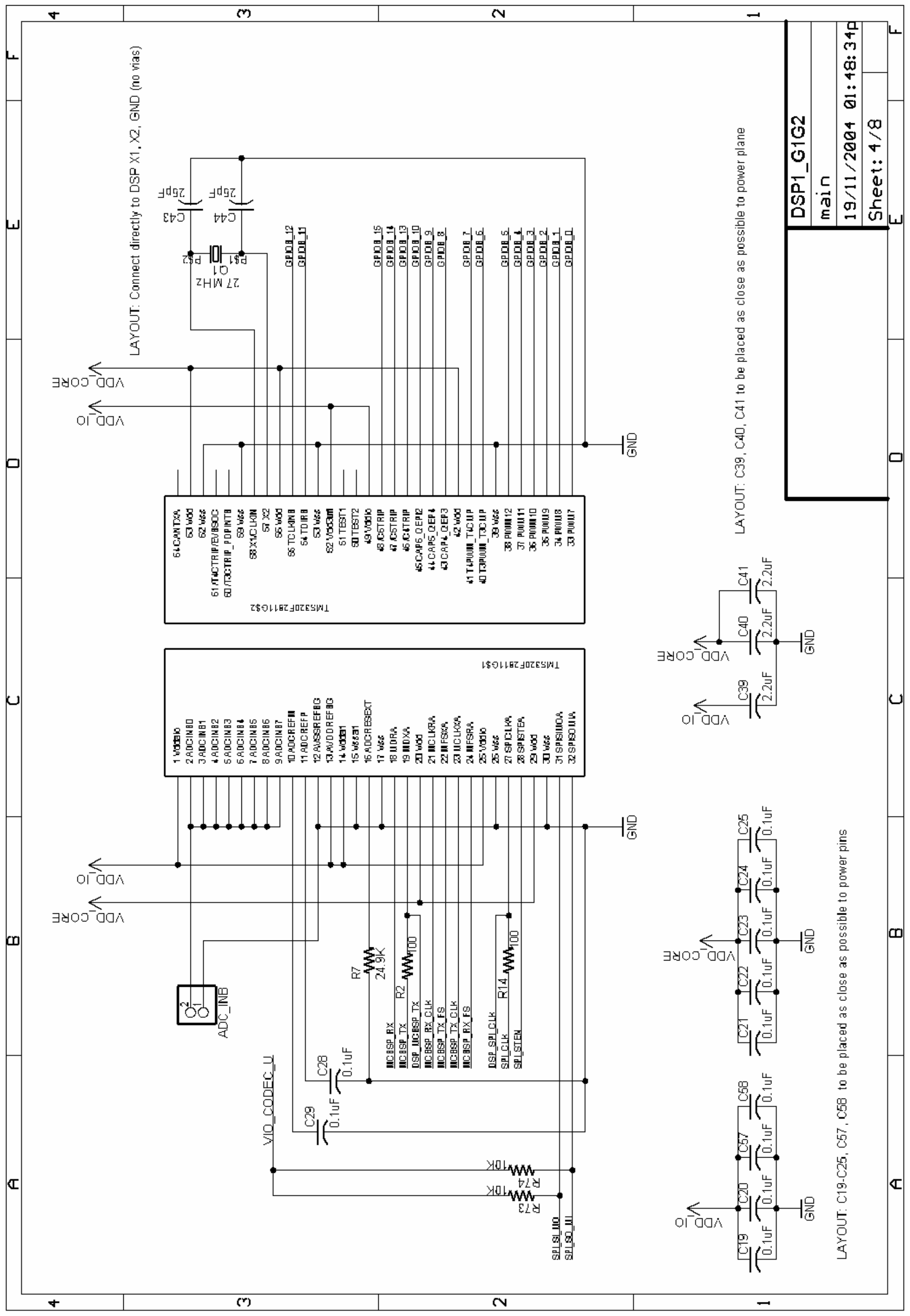
Appendix B PCB Schematic



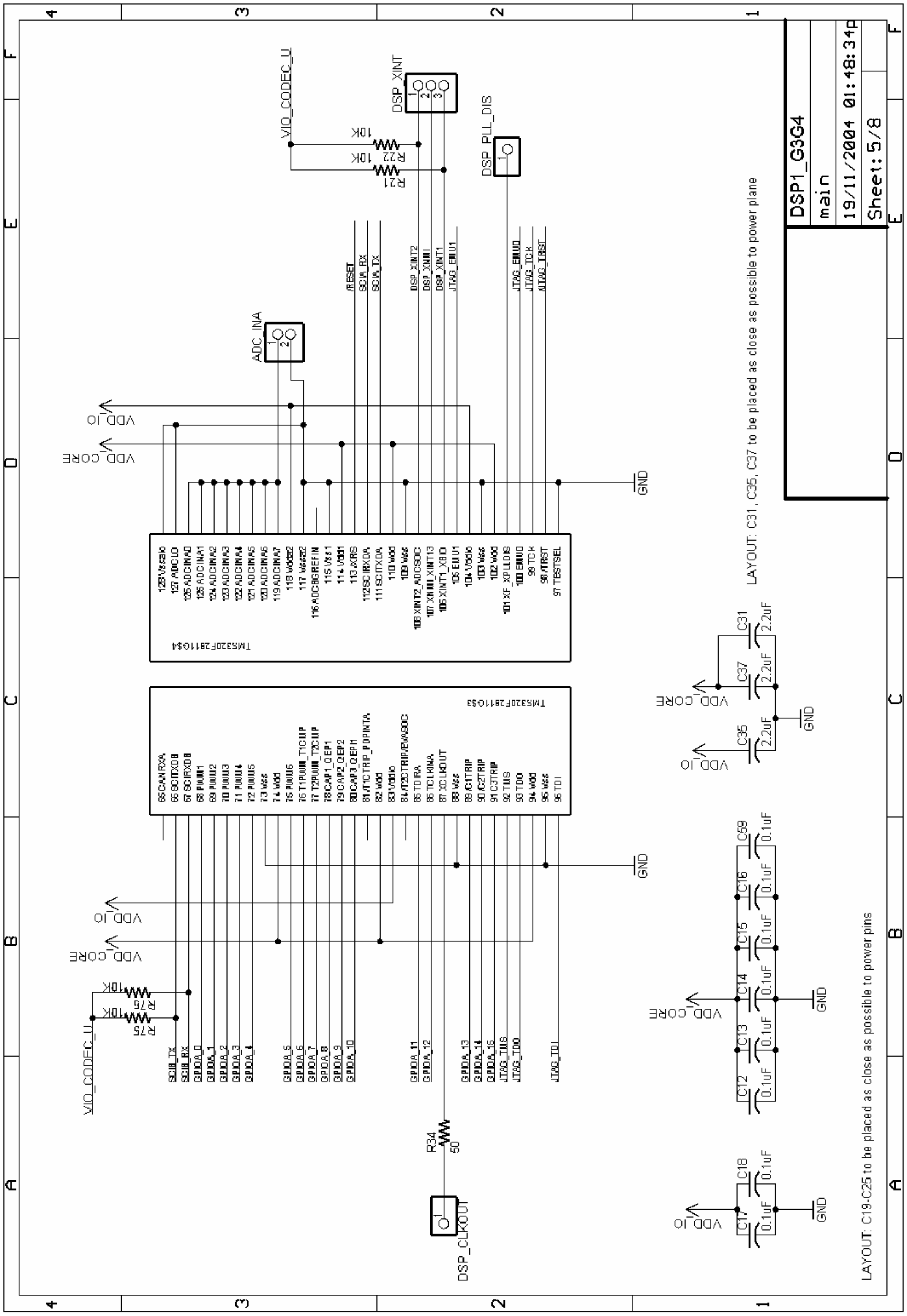

DEBUG
main
19/11/2004 01:48:34p
Sheet: 2/8
SPI
GPIO_A
GPIO_B
GND
CD74AC244
VIO_CODEC_U
BOOTMODE
JTAG
/JTAG_TRST
JTAG_EMU1
JTAG_TMS
JTAG_TDI
JTAG_TDO
JTAG_TCK
JTAG_EMU0
SCIA_TX
DSP_MCBSP_TX
DSP_SPI_CLK
SPI_STEN
VDD_CORE
R19 2.2K
NOTE: Pin 6 of JTAG should not be populated

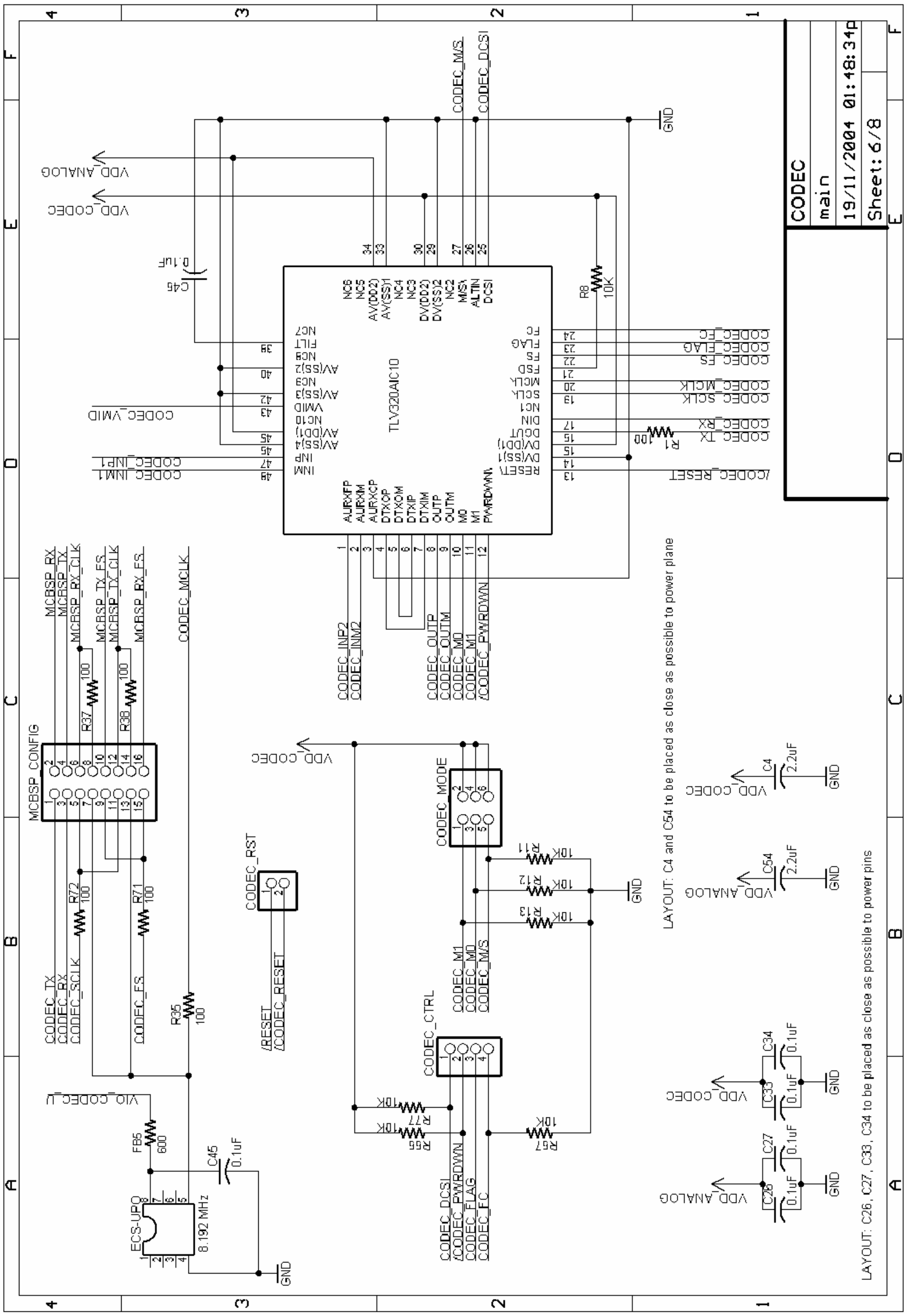


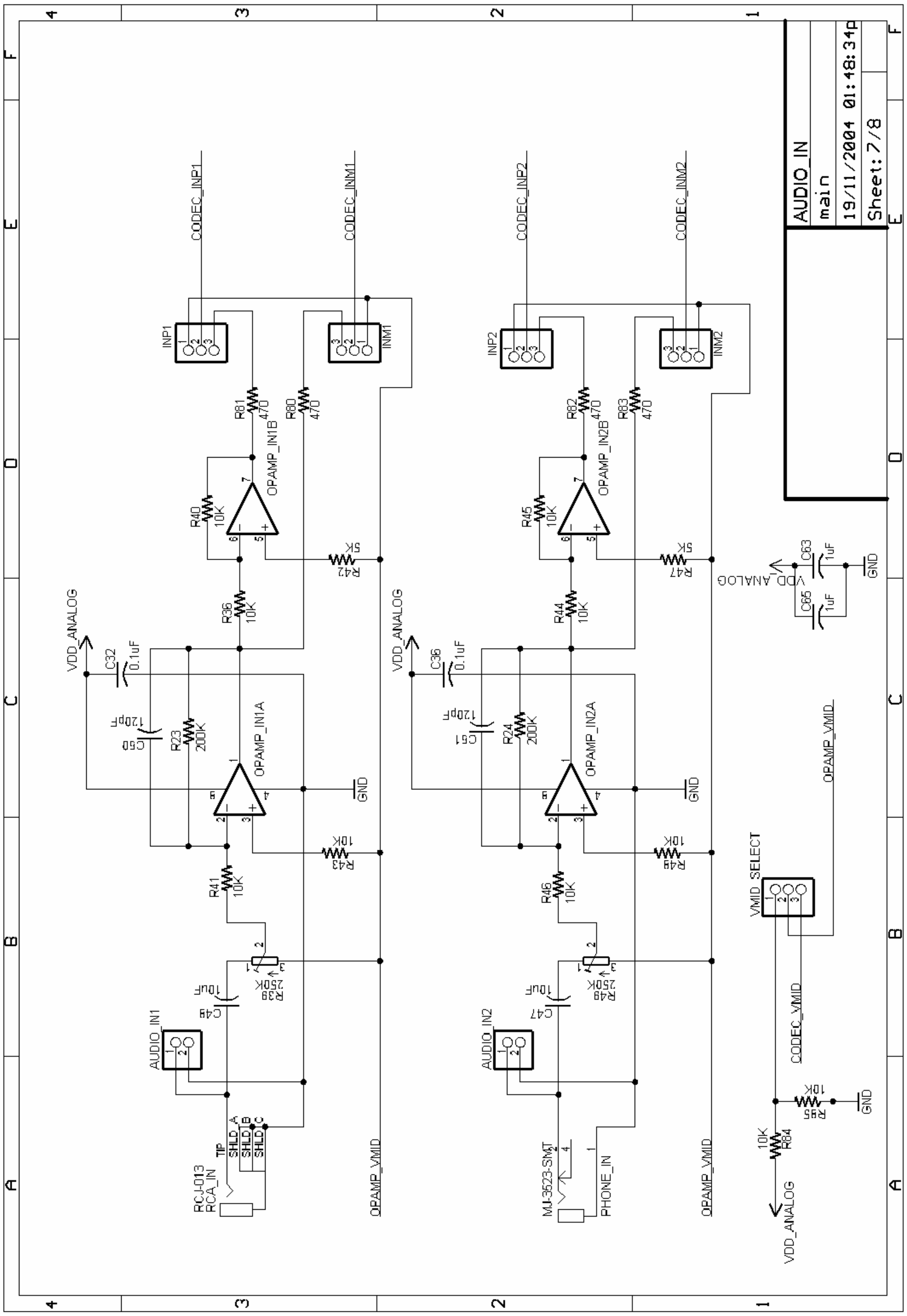
SERIAL
main
19/11/2004 01:48:34p
Sheet: 3/8
UART_B
UART_A
PORTB_TX
PORTB_RX
PORTA_TX
PORTA_RX
SCIB_LPBK
SCIA_LPBK
GND
C10
0.1uF
C8
0.1uF
C9
0.1uF
SCIA_RX
SCIB_RX
SCIA_TX
SCIB_TX
MAX3222
VCC
GND
RIN1
RIN2
DOUT1
DOUT2
C1+
C1-
NC1
NC2
V+
V-
ROUT1
ROUT2
DIN1
DIN2
C2+
C2-
EN\
PWRDOWN\
PORTA_RX
PORTB_RX
PORTA_TX
PORTB_TX
C7
0.1uF
MAX3222_PWRDOWN
R20
10K
C11
0.1uF
VIO_CODEC_U

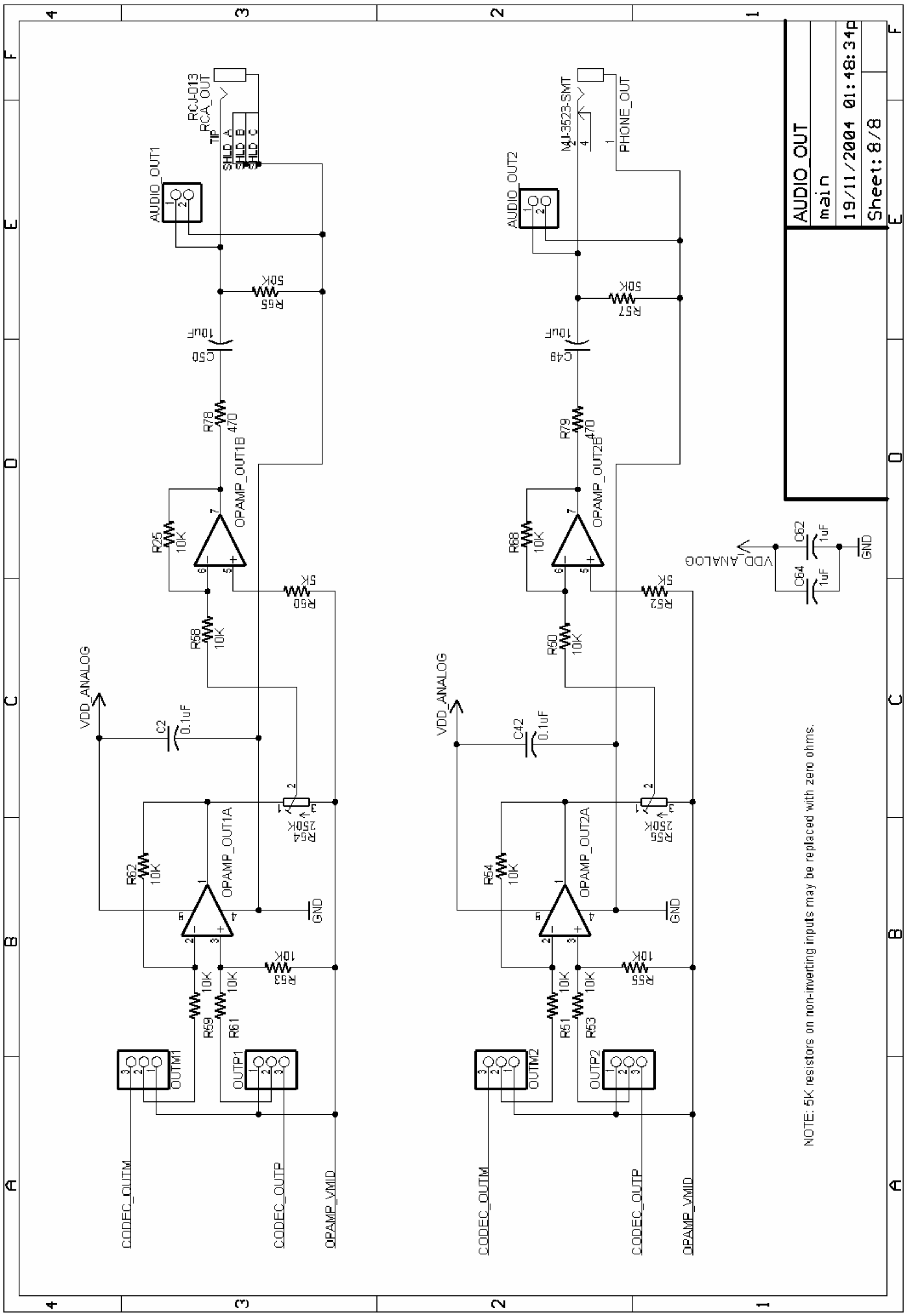


DSP1_G1G2
main
19/11/2004 01:48:34p
Sheet: 4/8
LAYOUT: Connect directly to DSP X1, X2, GND (no vias)
LAYOUT: C39, C40, C41 to be placed as close as possible to power plane
LAYOUT: C19-C25, C57, C58 to be placed as close as possible to power pins
TMS320F2811Q$1
TMS320F2811Q$2
27 MHz
Q1
C43
C44
25pF
VDD_CORE
VDD_IO
GND
VIO_CODEC_U
ADC_INB
R7
24.9K
R2
100
R14
100
R73
R74
10K
C28
C29
0.1uF
C39
C40
C41
2.2uF
C19
C20
C21
C22
C23
C24
C25
C57
C58
GPIOB_0
GPIOB_1
GPIOB_2
GPIOB_3
GPIOB_4
GPIOB_5
GPIOB_6
GPIOB_7
GPIOB_8
GPIOB_9
GPIOB_10
GPIOB_11
GPIOB_12
GPIOB_13
GPIOB_14
GPIOB_15
MCBSP_RX
MCBSP_TX
DSP_MCBSP_TX
MCBSP_RX_CLK
MCBSP_TX_FS
MCBSP_TX_CLK
MCBSP_RX_FS
DSP_SPI_CLK
SPI_CLK
SPI_STEN
SPI_SI_MO
SPI_SO_MI

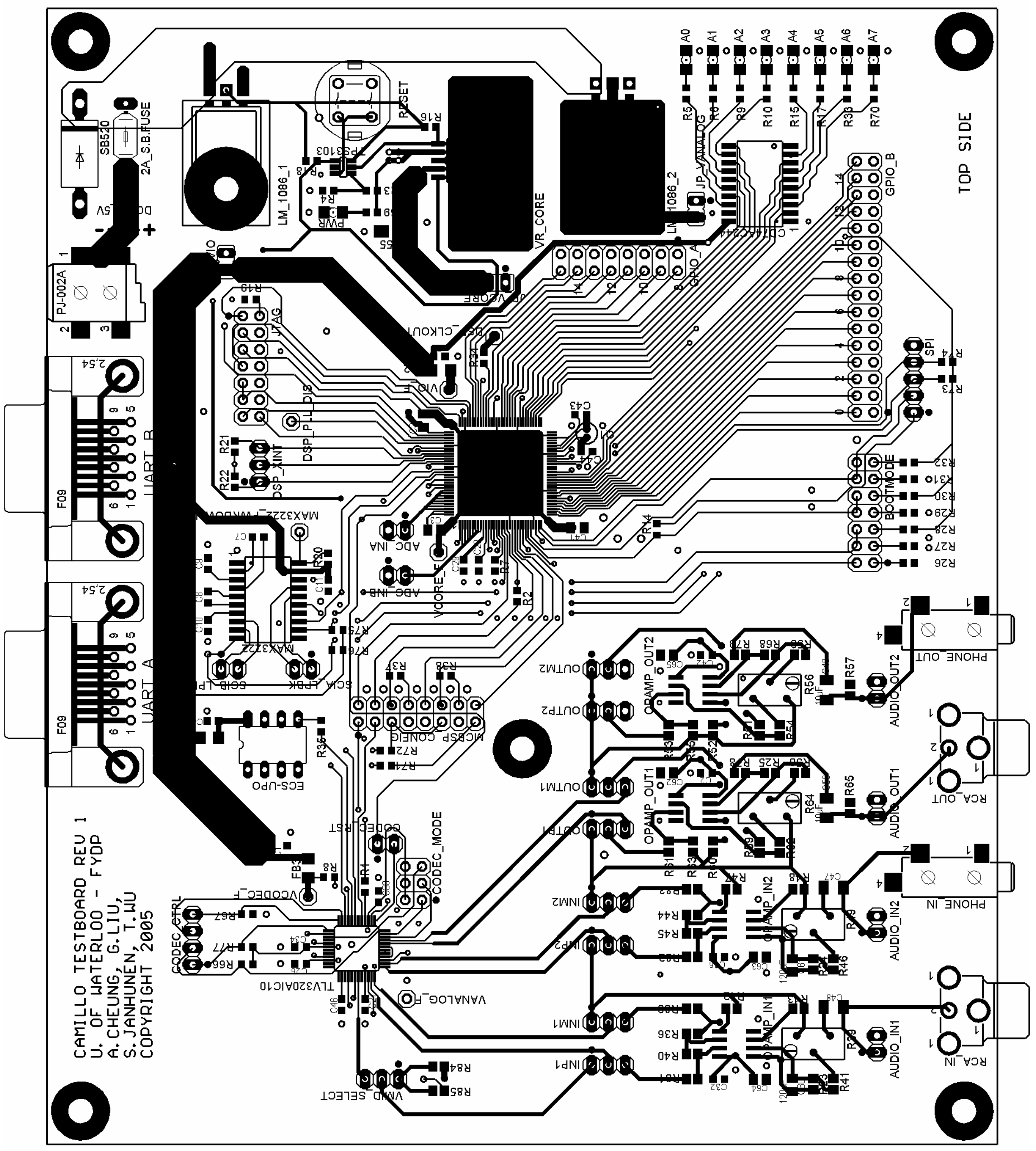

DSP1_G3G4
main
19/11/2004 01:48:34p
Sheet: 5/8
LAYOUT: C31, C35, C37 to be placed as close as possible to power plane
LAYOUT: C19-C25 to be placed as close as possible to power pins
TMS320F2811$4
TMS320F2811$3
VDD_IO
VDD_CORE
GND
VIO_CODEC_U
DSP_XINT
DSP_PLL_DIS
ADC_INA
DSP_CLKOUT
R21 10K
R22 10K
R75 10K
R76 10K
R34 50
/RESET
SCIA_RX
SCIA_TX
DSP_XINT2
DSP_XNMI
DSP_XINT1
JTAG_EMU1
JTAG_EMU0
JTAG_TCK
/JTAG_TRST
SCIB_TX
SCIB_RX
JTAG_TMS
JTAG_TDO
JTAG_TDI
C31 2.2uF
C37 2.2uF
C35 2.2uF
C59 0.1uF
C16 0.1uF
C15 0.1uF
C14 0.1uF
C13 0.1uF
C12 0.1uF
C18 0.1uF
C17 0.1uF

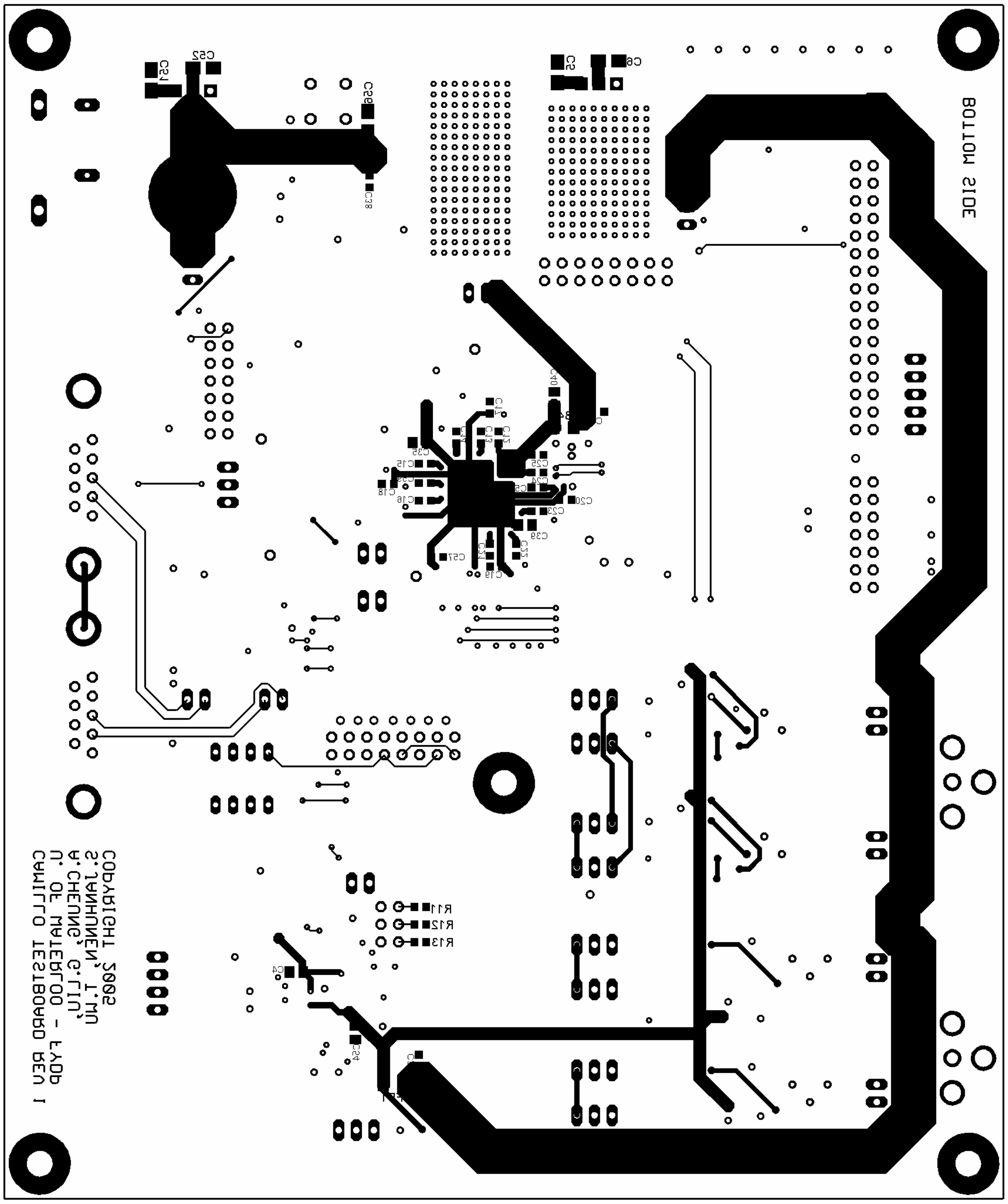

CODEC
main
19/11/2004 01:48:34p
Sheet: 6/8
TLV320AIC10
VDD_ANALOG
VDD_CODEC
GND
CODEC_M/S
CODEC_DCSI
CODEC_FC
CODEC_FLAG
CODEC_FS
CODEC_MCLK
CODEC_SCLK
CODEC_RX
CODEC_TX
/CODEC_RESET
CODEC_VMID
CODEC_INP1
CODEC_INM1
CODEC_INP2
CODEC_INM2
CODEC_OUTP
CODEC_OUTM
CODEC_M0
CODEC_M1
/CODEC_PWRDWN
MCBSP_CONFIG
MCBSP_RX
MCBSP_TX
MCBSP_RX_CLK
MCBSP_TX_FS
MCBSP_TX_CLK
MCBSP_RX_FS
CODEC_MCLK
CODEC_RST
/RESET
CODEC_MODE
CODEC_CTRL
VIO_CODEC_IF
ECS-UPO
8.192 MHz
LAYOUT: C4 and C54 to be placed as close as possible to power plane
LAYOUT: C26, C27, C33, C34 to be placed as close as possible to power pins


AUDIO_IN
main
19/11/2004 01:48:34p
Sheet: 7/8
CODEC_INP1
CODEC_INM1
CODEC_INP2
CODEC_INM2
INP1
INM1
INP2
INM2
R81 470
R80 470
R82 470
R83 470
OPAMP_IN1B
OPAMP_IN2B
OPAMP_IN1A
OPAMP_IN2A
R40 10K
R45 10K
R42 5K
R47 5K
R36 10K
R44 10K
VDD_ANALOG
C32 0.1uF
C36 0.1uF
C63 1uF
C65 1uF
C50 120pF
C51 120pF
R23 200K
R24 200K
GND
R43 10K
R48 10K
R41 10K
R46 10K
OPAMP_VMID
VMID_SELECT
CODEC_VMID
C49 10uF
C47 10uF
R39 250K
R49 250K
AUDIO_IN1
AUDIO_IN2
RCJ-013
RCA_IN
TIP
SHLD_A
SHLD_B
SHLD_C
MJ-3523-SMT
PHONE_IN
R84 10K
R85 10K


AUDIO_OUT
main
19/11/2004 01:48:34p
Sheet: 8/8
NOTE: 5K resistors on non-inverting inputs may be replaced with zero ohms.
RCJ-013
RCA_OUT
MJ-3523-SMT
PHONE_OUT
AUDIO_OUT1
AUDIO_OUT2
OPAMP_OUT1A
OPAMP_OUT1B
OPAMP_OUT2A
OPAMP_OUT2B
VDD_ANALOG
GND
CODEC_OUTM
CODEC_OUTP
OPAMP_VMID
OUTM1
OUTP1
OUTM2
OUTP2
C50 10uF
C49 10uF
C2 0.1uF
C42 0.1uF
C62 1uF
C64 1uF
R55 50K
R57 50K
R78 470
R79 470
R25 10K
R68 10K
R58 10K
R50 10K
R60 5K
R52 5K
R62 10K
R54 10K
R64 250K
R56 250K
R59 10K
R61 10K
R51 10K
R53 10K
R63 10K
R55 10K

Appendix C PCB Layout




BOTTOM SIDE
COPYRIGHT 2005
S.JANHUNEN, T.WU,
A.CHEUNG, G.LIU,
U. OF WATERLOO - FYDP
CAMILLO TESTBOARD REV 1
R11
R12
R13

Appendix D Firmware Interface Listing

```
//********************************************************************************
//
// HwInterface.h: Interface to all hardware features of the DSP.
//
//********************************************************************************

#ifndef __HWINTERFACE_H__
      #define __HWINTERFACE_H__

#include "GlobalDefs.h"

//********************************************************************************
//  G L O B A L   D E F I N I T I O N S
//********************************************************************************

typedef enum
{
      HW_UART_ID_A,
      HW_UART_ID_B,
      NUM_HW_UART_ID
} HW_UART_ID_TYPE;


typedef enum
{
    HW_UART_PARITY_NONE,
    HW_UART_PARITY_EVEN,
    HW_UART_PARITY_ODD,
    NUM_HW_UART_PARITY
} HW_UART_PARITY_TYPE;


typedef enum
{
      HW_GPIO_ID_A,
      HW_GPIO_ID_B,
      NUM_HW_GPIO_ID
} HW_GPIO_ID_TYPE;


typedef enum {
    HW_CODEC_GAIN_0_DB,
    HW_CODEC_GAIN_MINUS_36_DB,
    HW_CODEC_GAIN_MINUS_30_DB,
    HW_CODEC_GAIN_MINUS_24_DB,
    HW_CODEC_GAIN_MINUS_18_DB,
    HW_CODEC_GAIN_MINUS_12_DB,
    HW_CODEC_GAIN_MINUS_9_DB,
    HW_CODEC_GAIN_MINUS_6_DB,
    HW_CODEC_GAIN_MINUS_3_DB,
    HW_CODEC_GAIN_PLUS_3_DB,
    HW_CODEC_GAIN_PLUS_6_DB,
    HW_CODEC_GAIN_PLUS_9_DB,
    HW_CODEC_GAIN_PLUS_12_DB,
    HW_CODEC_GAIN_PLUS_18_DB,
    HW_CODEC_GAIN_PLUS_24_DB,
    HW_CODEC_GAIN_MUTE,
    NUM_HW_CODEC_GAIN
} HW_CODEC_GAIN_TYPE;
```

```
typedef enum {
    HW_CODEC_SAMPLE_RATE_4_KHZ,
    HW_CODEC_SAMPLE_RATE_8_KHZ,
    HW_CODEC_SAMPLE_RATE_16_KHZ,
    NUM_HW_CODEC_SAMPLE_RATE
} HW_CODEC_SAMPLE_RATE_TYPE;


typedef enum {
    HW_CODEC_RX_CHANNEL_1,
    HW_CODEC_RX_CHANNEL_2,
    NUM_HW_CODEC_RX_CHANNEL
} HW_CODEC_RX_CHANNEL_TYPE;


//******************************************************************************
//  F U N C T I O N   P R O T O T Y P E S
//******************************************************************************


//******************************************************************************
// Hardware initialization upon reset
//******************************************************************************
void HwInit(void);


//******************************************************************************
// UART Interface
//******************************************************************************
void HwUartOpen(HW_UART_ID_TYPE uartId, DWORD baud, HW_UART_PARITY_TYPE parity);
void HwUartClose(HW_UART_ID_TYPE uartId);
WORD HwUartRd(HW_UART_ID_TYPE uartID, char *buffer, WORD length);
WORD HwUartRdAvail(HW_UART_ID_TYPE uartId);
WORD HwUartWr(HW_UART_ID_TYPE uartID, const char *buffer, WORD length);
WORD HwUartWrAvail(HW_UART_ID_TYPE uartId);


//******************************************************************************
// Push-to-Talk Radio Interface
//******************************************************************************
void HwGpioSetPTT(BOOL enabled);
void HwGpioSetLEDs(WORD state);


//******************************************************************************
// GPIO Interface
//******************************************************************************
void HwGpioSet(HW_GPIO_ID_TYPE gpioId, WORD pattern);
void HwGpioClear(HW_GPIO_ID_TYPE gpioId, WORD pattern);
void HwGpioSetOutputEnable(HW_GPIO_ID_TYPE gpioId, WORD pattern);
void HwGpioClearOutputEnable(HW_GPIO_ID_TYPE gpioId, WORD pattern);
WORD HwGpioRead(HW_GPIO_ID_TYPE gpioId);


//******************************************************************************
// Audio Codec Interface
//******************************************************************************
void HwCodecOpen(HW_CODEC_SAMPLE_RATE_TYPE sampleRate);
void HwCodecClose(void);
BOOL HwCodecSetRxPGAGain(HW_CODEC_GAIN_TYPE gain);
BOOL HwCodecSetTxPGAGain(HW_CODEC_GAIN_TYPE gain);
BOOL HwCodecSetRxChannel(HW_CODEC_RX_CHANNEL_TYPE channel);
BOOL HwCodecSetLoopback(BOOL enable);
WORD HwCodecRdAvail(void);
WORD HwCodecRd(WORD *buffer, WORD count);
WORD HwCodecWrAvail(void);
WORD HwCodecWr(WORD *buffer, WORD count);


//******************************************************************************
// Hardware Timer Interface
//******************************************************************************
void HwTimerConfigure(WORD delayInMS, void (*funcPtr)(void));
void HwTimerEnable(void);
void HwTimerDisable(void);
DWORD HwTimerGetCurrentTicks(void);
DWORD HwTimerGetTicksPerMS(void);

#endif
```

Appendix E Bill of Materials

Part Description	Manufacturer	Manufacturer Part #	Qty per Board
BUMPER RECESSED #6 SCREW GREY	Keystone Electronics	724	5
STANDOFF HEX 6-32THR ALUM .750"	Keystone Electronics	2211	5
FUSE 2A 125V SLOW AXL BULK MS	Bel Fuse Inc	0676-2000-01	1
HEAT SINK TO-220 .250" COMPACT	Aavid Thermalloy	577002B00000	1
CONN D-SUB RECPT R/A 9POS PCB AU	AMP/Tyco Electronics	747844-6	2
IC OPAMP DUAL R-R I/O 8-SOIC	Analog Devices	AD8532AR	4
CAP .10UF 16V CERAMIC X7R 0603	Kemet	C0603C104K4RACTU	38
OCTAL LINE DRIVER	Texas Instruments	CD74AC244M96	1
27 MHZ CRYSTAL	Citizen	CSA309-27.000MABJ	1
SWITCH PB SQUARE WHT MOM SPST NO	ITT Industries	D6 C 00	1
PROGRAMMABLE OSCILLATOR (1-150 MHz)	ECS	ECS-UPO-8PIN	1
FERRITE 500MA 600 OHM 1206 SMD	HZ1206E601R-00	HZ1206E601R-00	5
IC REG POSITIVE 1.5A LDO TO-220	National Semiconductor	LM1086IT-3.3	2
LED RED CLEAR 1206 SMD	Lite-On Trading USA, Inc.	LTST-C150CKT	9
IC DRVR/RCVR MULTCH RS232 20SOIC	Texas Instruments	MAX3222IDWR	1
CONN AUD JACK MONO 3.5MM SMD	CUI Inc	MJ-3523-SMT-1	2
CONN POWER JACK 2.1MM SMD	CUI Inc	PJ-002A-SMT-1	1

Part Description	**Manufacturer**	**Manufacturer Part #**	**Qty per Board**
SCREW MACHINE PHILLIPS 6-32X3/8	Building Fasteners	PMS 632 0038 PH	10
POT 250K OHM TRIM 25TRN TOP ADJ	Murata Electronics North America	PV36Y254A01B00	4
CONN RCA JACK METAL R/A WHT PCB	CUI Inc	RCJ-013	2
RECTIFIER SCHOTKY 5A 20V DO201AD	General Semiconductor	SB520	1
IC 22-KSPS DSP CODEC 48-QFP	Texas Instruments	TLV320AIC10IPFB	1
IC DSP W/FLASH 128-LQFP	Texas Instruments	TMS320F2811PBKA	1
IC SUPERVISORY CIRCUIT SOT-23-6	Texas Instruments	TPS3103K33DBVR	1
IC REG LDO LINR 1.8V 1A SOT223-5	Texas Instruments	TPS72518DCQR	1
CONN HDR BRKWAY .100 40POS VERT	AMP/Tyco Electronics	4-103185-0	70
CONN HDR BRKWAY .100 80POS VERT	AMP/Tyco Electronics	4-103186-0	49
SHUNT LP W/HANDLE 2 POS 30AU	AMP/Tyco Electronics	881545-2	31
CAP 10000PF 50V CERM X7R 0603	Panasonic - ECG	ECJ-1VB1H103K	38
CAPACITOR TANT 10UF 16V 20% SMD	Kemet	T491A106M016AS	5
CAPACITOR TANT 22UF 10V 20% SMD	Kemet	T494A226M010AS	1
CAP 120PF 50V CERM CHIP 0805 SMD	Panasonic - ECG	ECJ-2VC1H121J	2
CAP 2.2UF 6.3V CERAMIC 0603 Y5V	Panasonic - ECG	ECJ-1VF0J225Z	1
CAP 10UF 6.3V CERAMIC X5R 1206	Kemet	C1206C106K9PACTU	4

Part Description	**Manufacturer**	**Manufacturer Part #**	**Qty per Board**
CAPACITOR TANT 22UF 10V 20% SMD	Kemet	T491A226M010AS	2
CAP CER 1.0UF 10V 20% X5R 0805	Murata Electronics North America	GRM219R61A105MA01D	4
CAP 2.2UF 25V CERAMIC F 0805	Panasonic - ECG	ECJ-2FF1E225Z	8
CAP 10UF 10V CERAMIC F 0805	Panasonic - ECG	ECJ-2FF1A106Z	8
CAP CERAMIC 27PF 50V 0603 SMD	Panasonic - ECG	ECJ-1VC1H270J	2
CAP CERAMIC 16PF 5% 50V C0G 0603	Rohm	MCH185A160JK	2
CAP CERAMIC 36PF 5% 50V C0G 0603	Rohm	MCH185A360JK	2
RES 100 OHM 1/10W 5% 0603 SMD	Rohm	MCR03EZPJ101	9
RES 51 OHM 1/10W 5% 0603 SMD	Rohm	MCR03EZPJ510	9
RES 0.0 OHM 1/10W 5% 0603 SMD	Rohm	MCR03EZPJ000	9
RES 120 OHM 1/10W 5% 0603 SMD	Rohm	MCR03EZPJ121	9
RES 470 OHM 1/8W 1% 0805 SMD	Rohm	MCR10EZHF4700	6
RES 51 OHM 1/8W 5% 0805 SMD	Rohm	MCR10EZHJ510	6
RES 10K OHM 1/10W 5% 0603 SMD	Rohm	MCR03EZPJ103	20
RES 1.0K OHM 1/10W 5% 0603 SMD	Rohm	MCR03EZPJ102	4
RES 10.0K OHM 1/8W 1% 0805 SMD	Rohm	MCR10EZHF1002	24
RES 11.0K OHM 1/10W 1% 0603 SMD	Yageo America	9C06031A1102FKHFT	1
RES 2.2K OHM 1/10W 5% 0603 SMD	Rohm	MCR03EZPJ222	1
RES 200K OHM 1/8W 1% 0805 SMD	Rohm	MCR10EZHF2003	2
RES 24.9K OHM 1/10W 1% 0603 SMD	Yageo America	9C06031A2492FKHFT	1

Part Description	Manufacturer	Manufacturer Part #	Qty per Board
RES 3.3K OHM 1/10W 5% 0603 SMD	Rohm	MCR03EZPJ332	1
RES 38.3K OHM 1/10W 1% 0603 SMD	Yageo America	9C06031A3832FKHFT	1
RES 51.0K OHM 1/8W 1% 0805 SMD	Rohm	MCR10EZHF5102	2
RES 5.1K OHM 1/10W 5% 0603 SMD	Rohm	MCR03EZPJ512	2
RES 5.10K OHM 1/8W 1% 0805 SMD	Rohm	MCR10EZHF5101	8
RES 0.0 OHM 1/4W 5% 1206 SMD	Yageo America	9C12063A0R00JLHFT	5