AN ABSTRACT OF THE THESIS OF

Alana Sweat for the degrees of Honors Baccalaureate of Science in Electrical and Computer Engineering and Honors Baccalaureate of Science in Mathematics presented on May 27, 2010. Title: Cell Phone Audio Controlled Point of Sale.

Abstract approved: ______________________________

Roger Traylor

A Cell Phone Audio Controlled Point of Sale system was designed and built for an undergraduate electrical and computer engineering senior design project. The system was designed to enable a user to transfer data from any cell phone to a point of sale device using audio tones. The project involved researching the frequency limitations of cellular phone networks and designing and building a point of sale device capable of decoding audio signals. The end result was a functioning prototype which was demonstrated at the 2009 Engineering Expo. This document describes the design process, implementation, and testing results of the project.

Key Words: engineering, audio, cell phone

Corresponding e-mail address: sweata@onid.orst.edu

Cell Phone Audio Controlled Point of Sale

by

Alana Sweat

A PROJECT

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degrees of

Honors Baccalaureate of Science in Electrical and Computer Engineering
(Honors Scholar)
Honors Baccalaureate of Science in Mathematics (Honors Scholar)

Presented May 27, 2010
Commencement June 2010

Honors Baccalaureate of Science in Electrical and Computer Engineering and Honors Baccalaureate of Science in Mathematics project of Alana Sweat presented on May 27, 2010.

APPROVED:

Mentor, representing Electrical and Computer Engineering

Committee Member, representing Electrical and Computer Engineering

Committee Member, representing Mathematics

Head, School of Electrical Engineering and Computer Science

Dean, University Honors College

Alana Sweat, Author

ACKNOWLEDGEMENT

First, I would like to thank my project team members, Maggie Watkins and Lin Fu, since this project would not have been completed without their efforts. I would also like to thank Adam Kuenzi, our project mentor from GE Security, for his continued support throughout the entire project.

CONTRIBUTION OF CO-AUTHORS

This paper is based on the final specification document written for a senior design group project, and hence includes work done by Maggie Watkins and Lin Fu.

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

INTRODUCTION

Project Description

Picture this: you've been searching forever to find a parking place downtown, and you finally see one. You park the car and get out to add money to the parking meter, only to realize that you don't have any change. However, you do have your cell phone. You dial an 800 number, enter a few codes at the prompt of the voice on the phone, and hold your cell phone up to the parking meter. A moment later, a tone comes out of the phone, the parking meter decodes it and adds the amount of money you specified to the meter, and you're free to go about your business.

The Cell Phone Audio Controlled Point of Sale system will consist of three main components—a cell phone, a tone generator, and a point of sale device.

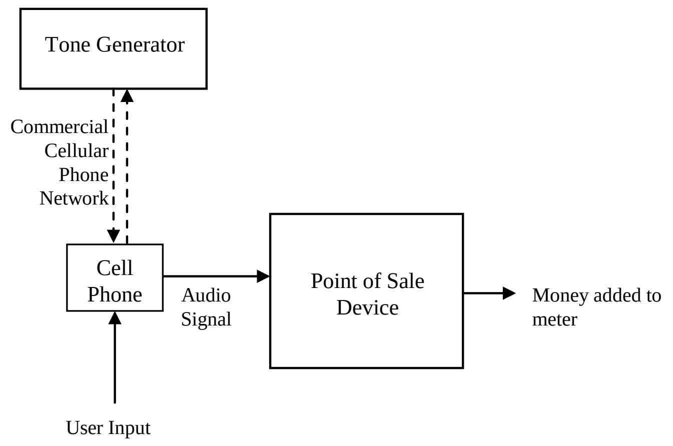

Figure 1: Simple Diagram of POS system

To use the system, a user places a call from a cell phone to an 800 number. The tone generator receives the call, uses caller ID to recognize the caller, and then asks the caller for a PIN, the ID number of the point of sale system, and dollar amount to add to the meter. Next, the tone generator uses a secure encryption algorithm to encode the dollar amount of the purchase into an audible sequence of tones and sends it to the cell phone. As the tones play out the cell phone's speaker, the caller holds the speaker up to a microphone on the point of sale device (e.g. parking meter) to transfer the message. The microphone receives the message and passes it to a decoder algorithm. Once the message is decoded, the money is added to the parking meter. Since the data is sent as a sequence of tones over the phone, the system will work from any cell phone on any network.

Senior Design Class Expectations

A prototype of the project described above was completed for a senior design class, and as such there were specific requirements and procedures that had to be completed. We worked in a group of three students and had a project mentor from the sponsoring company, GE Security. I was primarily responsible for all microcontroller and computer programming but was heavily involved in all aspects of the project.

The first term of the year long class focused on designing the project without building any of the components. This allowed us to focus on having a solid, well-documented plan in place before building anything. During this time we did all of our product research and completed the majority of the project design specification. The second term was focused on finalizing the design and building our project, with the goal of having a working system at the end of the term. The third term was geared primarily towards putting the finishing touches on the project, as well as preparing for and participating in the Engineering Expo, where we demonstrated our project to interested spectators.

Sponsor Company Expectations

The project sponsor, GE Security, was primarily interested in the algorithms and necessary hardware to encode a message, send it over an audio voice call on any cell phone network, and receive and decode the message at the decoding unit. Once developed, the technology could be used in any unmanned security system as a way for a user to provide personal identification. Because the message transferred could be sensitive (i.e. credit card number, ID number, etc.), data encryption security and system reliability are the essential requirements for this project. Other requirements that were important to the project sponsor include a way to measure the success rate and a functional prototype that works with any cell phone.

SYSTEM DESIGN

Overall System

Comparable Systems

Before beginning any design work, our group first researched pre-existing systems similar to our project. We found two, GE Security's TRACkey, which transmits audio over landline phones, and the old acoustically coupled modems, which also transmitted over landline connections.

Table 1: Comparable Systems

Product	**Description**	**Data Transfer Method**	**Direction of Data Transfer**	**Operating Distance**
GE Security TRACkey [9]	Electronic key that records data about what and when it opened a lock. The data is transmitted to a server via a touch-tone telephone.	Touch-tone telephone	User to server	1/4" to 1/2"
Acoustically Coupled Modem [10]	Converts data (ones and zeros) into audible tones and converts audible tones into data.	Telephone	Bi-directional	As close as possible

GE Security's TRACkey [9] is very similar to this project in that it transmits data over an audio phone call. A TRACkey user holds the TRACkey handheld device up to a telephone receiver, presses a sequence of keys, and waits for the audible message to be transmitted to a data server at the other end of the telephone. The most obvious difference is the direction of data transfer. The TRACkey sends data from the key through the phone to the server, but the audio POS system will send data from the server through the phone to the POS device. Since the data direction of the audio POS system is reversed compared to the TRACkey, the encoding and decoding responsibilities are also reversed. The

TRACkey encodes the data and the server decodes the data. Whereas, the server will encode the data and the POS device will decode the data for the audio POS system.

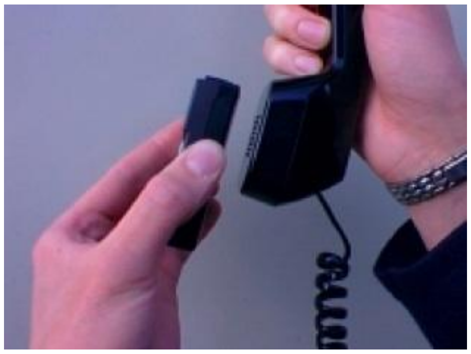

Figure 2: TRACkey in use [9].

The TRACkey is specifically designed to transmit data over a landline, not over a cell phone network. Because cell phone networks apply audio compression to the signals sent, the signal encoding method and data transfer rate required for a successful message transfer will likely be different for the audio POS system than for the TRACkey.

Starting in the late 1960s, acoustically coupled modems were used to enable computers to transmit and receive data over a telephone line using a standard telephone handset [10]. The acoustic coupler would connect to the computer and the telephone handset would fit into two rubber seals. The computer would send data in the form of ones and zeros to the coupler. The coupler converted the data into audible tones that could be picked up by the telephone and transferred to another computer at the other end of the phone. The other computer could send data back using the same form of communication. Data transfer rates for acoustically coupled modems were usually about 300 bps [16].

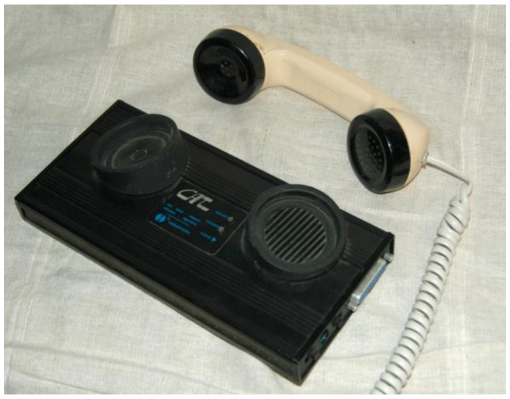

Figure 3: Acoustically coupled modem [16]

The acoustically coupled modem is very similar to the audio POS system in the way it transfers data. The main difference is that the acoustically coupled modem was designed to work with a landline, not over a cellular network. Even though a data transfer rate of 300 bps is slow relative to modern transfer rates, it may be too fast for a cellular network to transmit without losing some data.

Top-Level Design

The end goal of this senior design project was to demonstrate a functioning project in the form of a prototype. This prototype shall include all of the features listed in the Project Requirements section. In general, we were looking to attain an 80% data transmission success rate, as in 8 out of 10 bits of data shall be displayed correctly by the POS device. Future prototypes outside of class could incorporate additional features, such as increased security (discussed in Appendix D). In order to demonstrate the functionality of our prototype, we were required to create the Signal Generation portion of our system, which generates the encoded signals and sends them to the user’s cell phone. Part of creating this testing setup included researching which encoding algorithm and baud rate resulted in optimal data transmission, particularly in noisy environments.

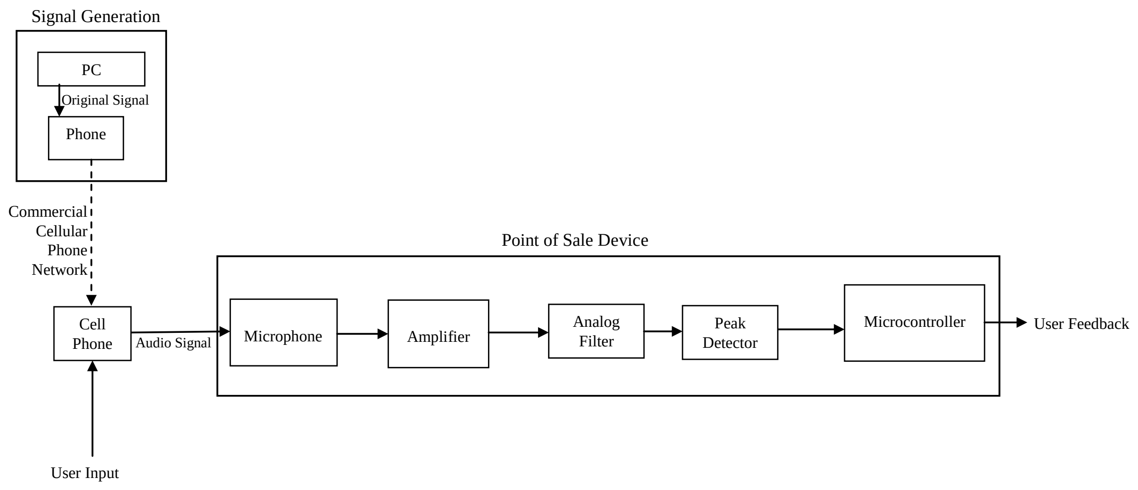

Figure 4: Top Level Block Diagram

When testing our system, the Signal Generation block first sends a 16-bit audio encoded message to the user's cell phone. The microphone of the POS device then picks up the resulting audio signal from the cell phone's speaker. This audio signal is then amplified, filtered, and DC rectified before entering the microcontroller, where it is decoded and displayed on an LCD.

Project Requirements

To have our project be judged successful, we had to pass the majority of the following requirements to the satisfaction of our project mentor. Our final results will be discussed in the Results section of the paper.

The first requirement is that a message shall be transmitted using audible tones over an audio voice call. To test this requirement, we used the system to transmit a message. It passed if the system used only audio to transmit the message.

The second requirement is that a 15-bit message shall be sent with an 80% probability of success. To test this requirement, we held the cell phone speaker up to the microphone at a distance to allow for maximum sound-to-noise ratio. Then we transmitted 10 sets of data, each of length 100-bits. We then compared the decoded results on the LCD with the original data sent and counted the number of bits transmitted incorrectly. We then calculated the bit-error-rate, which is the number of incorrect bits divided by the total number of bits sent (1000). We then determined the probability that 15 consecutive bits would be transmitted correctly to be $(1 – \text{bit-error-rate})^{15}$. To pass, that probability had to be at least 80%.

The third requirement is that the system shall successfully decode at least one message when the cell phone speaker is within ½ inch of the microphone without using a speakerphone. To test this requirement, we maximized the volume of the cell phone and ensured that the speakerphone was off. We then held the cell phone speaker ½ inch away from the microphone, and sent a 16-bit message. To pass, we had to be able to send at least one entire message correctly at that distance.

The fourth requirement is that the system shall be able to successfully send 100-bits of data. To test this requirement, we held the cell phone speaker up to the microphone at a distance to allow for maximum sound to noise ration. We then sent a message containing 100-bits of data. To pass, we had to be able to send all 100-bits correctly at least once.

The fifth requirement is that a working prototype shall be created that uses a microcontroller to decode the audio signal and provides a way to view the decoded message. To test this requirement we made a voice call from a cell phone to another phone that plays an encoded signal. We then held the phone up to the microphone at a distance to allow for maximum sound-to-noise ratio, and compared the decoded message on the LCD to the message sent. To pass, the original message had to be displayed correctly 2 out of 3 times.

The sixth requirement is that the system shall meet the minimum success requirement of 80% when a recording of the traffic sounds at 3rd and Van Buren or the ambient noise at an event similar to the senior design expo is playing near the system at the average sound pressure level that the noise was recorded. To test this requirement, we created 2 minute recordings of traffic and voices, noting the average sound pressure level in dB with a meter. We then played back the recordings at the recorded average sound pressure level as we performed the same test described in the second requirement. To pass, the probability of sending 15 consecutive bits correctly had to be at least 80%.

Signal Generation

This section is part of the testing setup required to verify functionality of our project.

Research

A large part of our project was determining the optimal frequencies, baud rate, and encoding algorithms to use to successfully transmit data over a wireless phone network. Since we were gathering this data experimentally prior to building our prototype, we used a computer with Matlab to do all signal processing. The setup for our testing was simple: a microphone and amplifier hooked up to the computer to receive the audio signal from a cell phone, and a landline phone connected to the headphone jack on the computer to send out the signal to the cell phone.

The three different algorithms we looked at were OOK (on-off keying), FSK (frequency-shift keying), and DTMF (dual-tone multi-frequency). OOK is the simplest algorithm. Basically, an audio pulse at one given frequency represents a binary ‘1’, and a ‘0’ is represented by no audio pulse. FSK is similar to OOK, only with ‘0’ being represented by an audio pulse at a second given frequency. DTMF is the most complicated, and is what is heard when dialing numbers on phones. Basically, each decimal number 0-9, as well as several other special characters, is represented by the presence of a different combination of two frequencies. In a typical phone application, there are seven different frequencies used, but only 12 combinations are assigned values.

We had very little success when trying to test DTMF. As can be seen in the summary of our testing results below, our best result was a 66% error rate (for DTMF, a ‘bit’ is actually a decimal number).

Table 2: DTMF Testing Results

Baud Rate (bits/sec)	**Number of bits sent**	**Number of bits received incorrectly**	**% Error**
8	250	207	82%
10	250	166	66%
20	250	186	74%
50	250	235	94%
100	250	243	97%

In general, we saw much better results when testing OOK and FSK. After running several tests on each algorithm, it became very apparent that performance degraded significantly as the baud rate increased, so we limited most of our testing to only 8, 10, and 20 bits/second. OOK allowed us to get an idea what range of frequencies we could use in our project. Based on the results below, as well as several other quick tests that we didn’t record, we saw a significant degradation of performance below 750 Hz and above

1.6 kHz, and so we kept our FSK testing within that range. Though we tried to decrease random anomalies from static or interference in the call by sending large amounts of data, our results still tended to be a little unpredictable. For OOK, our best results occurred in the 1 k - 1.25 kHz range at baud rates of 8-10 bits/second.

Table 3: OOK Testing Results

Frequency (Hz)	**Baud Rate (bits/sec)**	**Number of bits sent**	**Number of bits received incorrectly**	**% Error**
750 Hz	8	1000	207	21%
	10	1000	141	14%
	20	1000	217	22%
1 kHz	8	1000	91	9.1%
	10	1000	124	12%
	20	1000	84	8.4%
	50	1000	219	22%
	100	1000	393	39%
1.25 kHz	8	1000	192	19%
	10	1000	160	16%
	20	1000	204	20%
1.5 kHz	8	1000	218	22%
	10	1000	216	22%
	20	1000	241	24%

For FSK, we saw similar performance trends as in OOK, such as decreased performance as baud rate increased. From the results below and other tests, we determined that a larger difference in frequencies between the ‘1’ and ‘0’ signals lead to better performance. We had the best results at baud rates of 8-10 bits/second and the frequency pairs 1k/1.5 kHz and 1k/1.25 kHz.

Table 4: FSK Testing Results

Frequencies (Hz)	**Baud Rate (bits/sec)**	**Number of bits sent**	**Number of bits received incorrectly**	**% Error**
1 kHz = ‘0’ 1.5 kHz = ‘1’	8	1000	81	8.1%
	10	1000	361	36%
	20	1000	406	41%
	50	1000	338	34%
	100	1000	438	44%

500 Hz = ‘0’ 1 kHz = ‘1’	8	1000	430	43%
	10	1000	444	44%
	20	1000	495	50%
1 kHz = ‘0’ 1.25 kHz = ‘1’	8	1000	146	15%
	10	1000	346	35%
	20	1000	408	41%

Implementation

After analyzing our collected data and discussing the design requirements for each algorithm, we ended up using OOK at 10 bits/second at 1.25 kHz for our project. We choose OOK over FSK primarily for the ease in implantation, as the performance for the two algorithms was similar during our testing.

In our final prototype, the user enters the desired decimal string (such as ‘1234’) into Matlab, and the function we wrote to generate an OOK signal (see Appendix B) converts that to 4-bit binary and encodes the binary string into an audio signal using the presence or absence of sine waves of 1.25 kHz. This audio signal is then played on the computer so that the audio is routed to the headphone jack. We connected the headphone jack on the computer to the microphone jack on a landline phone, and were then able to play the file on the computer and have it transferred into the landline phone, through the commercial cellular phone network, and into the user’s cell phone.

Microphone & Amplifier

This part of the system is located in the POS device, and its purpose is to receive and amplify the audio signal emitted by the user’s cell phone.

Research

When researching microphones, there is an extremely large variety to choose from. For our project, the microphone had to pick up and not distort our signal, which was in the 1k-2k Hz range, and it had to be sensitive enough to pick up a signal coming from cell phone speakers (which we had trouble quantifying). Several microphones we looked at can be seen in Table 5.

Table 5: Microphones

Part Number	Manufacturer	Frequency Response	Supply Voltage	Impedance	Other Features	Frequency Range	Price
WM-63PRT [7]	Panasonic-ECG	Flat < 5k Hz, sloped after	2V	2200 ohms	Omni-directional	20-16k Hz	$3.47
CMR-5054TB-A [5]	CUI, Inc.	Flat under specific conditions	1.5 V	2200 ohms	Noise-cancelling	100-20k Hz	$2.22
WP-23502-P16 [8]	Knowles Acoustics	Flat	1.3V	4400 ohms	Waterproof	100-6k Hz	$37.22

We ended up choosing a microphone similar to the Panasonic in the table. We determined that for our prototype, waterproofing is not important or worth the extra expense, as we will only be testing it indoors. Also, since the audio signal coming from the cell phone is fairly quiet, we didn't want it to be mistaken for noise and get dampened by the noise-cancelling microphone.

There were many options for amplifiers we could have used in our circuit, from simple op-amp designs to the powerful stereo audio power amplifiers used in portable electronics. We ended up using a SSM2166 IC amplifier from Analog Devices which our group had prior experience using. Part of this project was experimenting with the audio signal to determine what conditions produce the purest decodable signal, so we chose an amplifier with extra signal processing features. For example, the amplifier has a built in noise gate, which can filter out background noise below an adjustable threshold, which ended up being very useful in our final prototype.

Implementation

The microphone circuit is very simple. Its purpose is to bias the non-linear microphone capsule at an appropriate voltage. The microphone circuit picks up the audio signal from the cell phone and converts it into an electrical signal. The output voltage has amplitude of approximately 5mV. Capacitor C_c is a decoupling capacitor that removes the DC bias in the signal. The microphone datasheet shows that the microphone can accept more than 2V, but it recommends 2V input. The microphone circuit is shown in Figure 5, with the output Small_Analog_Signal feeding into the amplifier.

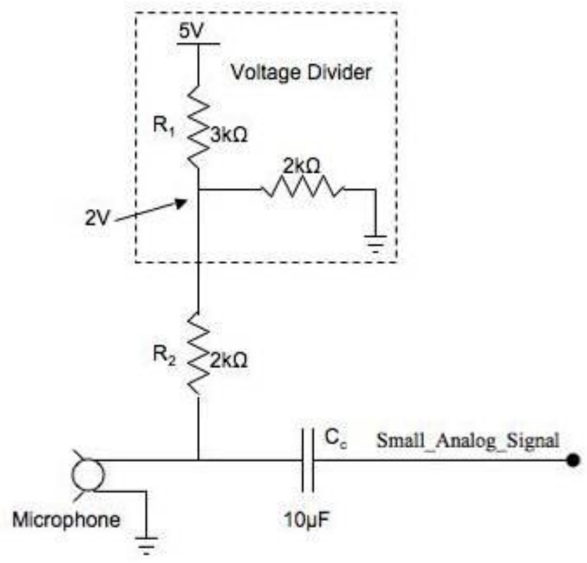

Figure 5: Microphone Biasing Circuit

To build the amplifier circuit, we thought of using one of the evaluation boards available for the SSM2166, but they are only available in large quantities. However, the chip and all of the external parts are large enough to solder onto a prototyping board by hand. Since a PCB development board is usually designed to eliminate unwanted parasitic capacitances that can hinder performance of the amplifier, we took precautions to prevent possible problems caused by using a prototyping board. We placed the bypass capacitor very close to the V+ pin on the IC and built the circuit as small as is practical to help eliminate parasitic capacitances that can build up between wire leads and the solder pads on the board.

The SSM2166 provides an internal gain of 0dB to 60dB, and an external op-amp on the output (pin 13) can increase the gain by an additional 20dB. Together the SSM2166 and the external op-amp OP213 provide an adjustable gain of 0dB to 80dB (or a voltage gain of 0 to 10,000). The complete circuit and pin descriptions are shown in Figure 6 and Table 6, with the input coming from the microphone circuit connected to pin 7. The frequency response is flat between 0Hz and 10k Hz [1].

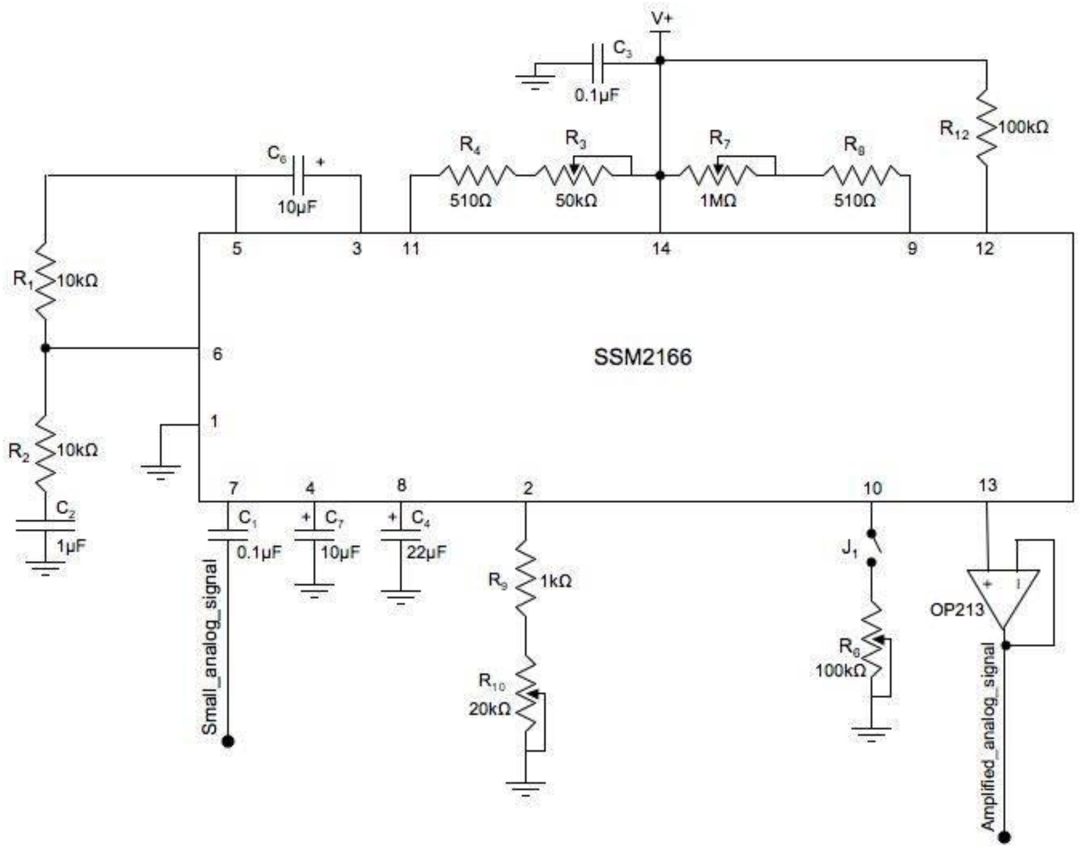

Figure 6: Amplifier Circuit Design [1]

Table 6: Amplifier Pin Descriptions [1]

Pin Number	Description
1	Ground
2	Gain adjust
3	VCA$_{in}$ (Input to variable controlled amplifier)
4	VCA$_R$ (Nonground reference for the audio signal)
5	Buf Out (Internal input buffer amplifier output pin)
6	-IN (Inverting input to the buffer)
7	Audio +in (Input audio signal)
8	Avg cap (Detector averaging capacitor)
9	Noise Gate Set (Threshold set point)
10	Comp Ratio Set
11	Rotation Set (Limiting set point)
12	Power Down
13	Output signal
14	V+ (5V Nominal)

Analog Filter

The purpose of the analog filter is to attenuate undesired signal frequencies while retaining the desired signal frequency**.** The input is from the amplifier, and the output feeds into the peak detector.

Research

There are many implementations of filters that we could have chosen, such as active filters, passive filters and switched-capacitor filters. We chose to use active filters for our project, as they are usually easier to design than passive filters, and as a group we had the most experience working with them. Active filters use amplifying elements, especially op-amps, with resistors and capacitors in their feedback loops, to synthesize the desired filter characteristics. They will generate noise due to the amplifying circuitry, but this can be minimized by the use of low-noise amplifiers and careful circuit design.

Implementation

Since we only want to pass through one frequency, 1.25 kHz, we used one band-pass filter. Since we used an active filter, the desired frequency gets amplified, while frequencies outside the pass-band are not.

The circuit we used is a very simple op-amp based filter, as seen in Figure 7. The input signal, V_{in}, is the output of the amplifier stage. The output signal, V_o, is the input to the peak detector stage. R3 and C2 form a differentiator like in a high-pass filter, while C1, R1 and R2 form an integrator like in a low-pass filter. Because we choose to use OOK algorithm for our system, we are building only one band-pass filter since we only need to amplify one frequency and we want any other frequencies to be as close to zero in amplitude as possible. Our final filter design ended up with a bandwidth of approximately 100 Hz and a gain of about 10.

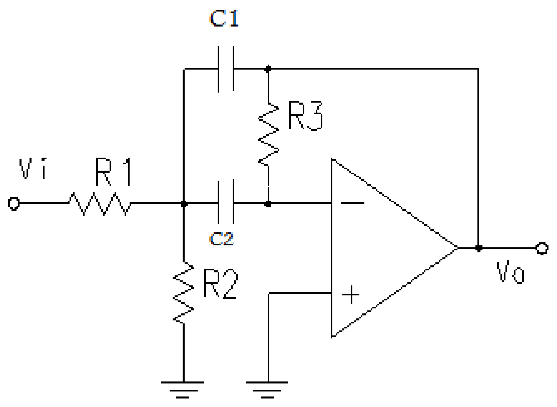

Figure 7: Schematic of active band-pass filter [13]

Peak Detector

Without a peak detector, when the analog signal from the filter stage is fed into the microcontroller, it will see wildly fluctuating values making it difficult to determine when the signal is high or low. The peak detector smoothes the sine wave-based analog signal into something closer to a digital, square wave based-signal.

Implementation

We used a simple full-wave rectifier with smoothing capacitor design. The four diodes in a bridge arrangement rectify the signal, and the voltage divider (R3 & R2) provides a DC offset. The capacitor smoothes the resulting pulsating DC signal into a steady state DC signal.

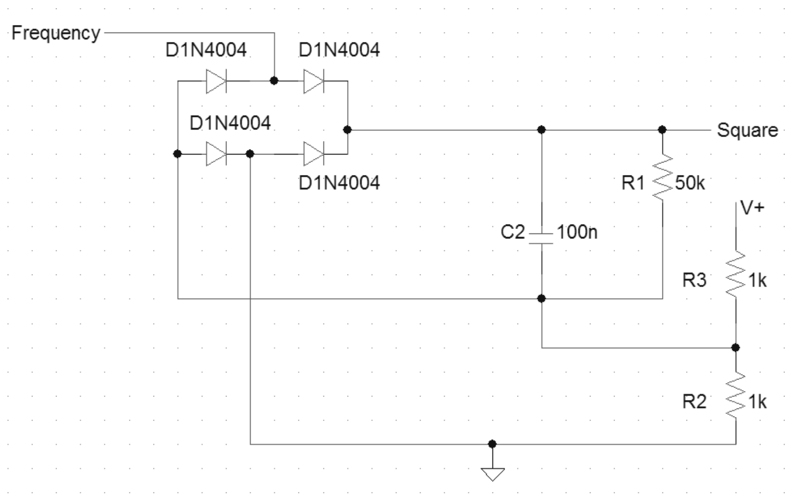

Figure 8: Schematic of Peak Detector

Microcontroller Hardware & Display

The purpose of the microcontroller is to take the signal from the peak detector stage, convert it into a binary string, and decode the string into decimal numbers, which are then displayed on the LCD display.

Research

The microcontroller used in the design needed to, at a minimum, be able to sample using an A/D converter the incoming signal 500 times per second (50 samples per bit of data at 10 bits per second). It needed an A/D converter with at least 2-channels, as when we choose a microcontroller we were still debating between using FSK, which needs two filters and hence two signals, and OOK which only needs one. When researching the microcontrollers, we were also interested in possibly creating digital filters instead of using analog filters, which would be easier on a specialized DSP microcontroller. To that end, we looked at two DSP microcontrollers, but after attempting to learn more about how to program digital filters, we determined that we could create acceptable analog filters with a lot less time and effort.

Table 7: Microcontrollers

Part Number	Manufacturer	Speed	Supply Voltage	SRAM	A/D # Channels	A/D Max Sample Rate	Data width	Price	Notes
ATmega 128 [3],[15]	Atmel	16 MHz	4.5V – 5.5 V	4K Bytes	8	3.8 kHz	8-bit	$6.73	General usage
STM32 (F101RB) [11],[12]	STMicro-electronics	36 MHz	2.0V – 3.6 V	16K Bytes	10	unknown	32-bit	$8.13	Optimized for DSP
AT32A P7000 [2],[4]	Atmel	150 MHz	3.0V – 3.6 V	32K Bytes	2	50 kHz	32-bit	$10.77	Optimized for DSP

We ended up using the ATmega128, as it met all of our requirements and we had a lot of prior experience working with it.

Implementation

We used the Tekbots development board, which contains the Atmega128 and a 5V dc regulator which can be powered using either AC or DC. It also contains various useful peripherals, such as a DB-9 connector for connecting to a computer, buttons and an integrated LCD display [14]. The LCD display used is the Hitachi HD44780U, which can display 16 digits on 2 lines [6]. The microcontroller board comes with pre-written functions for using the LCD display. Power is supplied to the board via a standard 9VDC power supply/wall wart, and the regulated 5V is distributed to all other circuits in the POS device. The pins used on the microcontroller are all for the A/D converter or power.

Microcontroller Code

The microcontroller that we chose can be programmed in C, assembly, or possibly Java. We chose to use C as that is what we were the most familiar with. The code was developed with GCC on a Linux operating system, and then transferred to the microcontroller via a USB in-system programmer using AVRdude.

The microcontroller, when a button is pressed, uses the analog to digital converter to sample the incoming signal 50 times per bit of data. Since we are using a baud rate of 10 bits per second, the microcontroller samples the incoming signal at 500 Hz. To meet our testing requirements, we need to be able to either send a 16-bit message or a 100-bit message. So, depending on the button pressed, the microcontroller will either sample for the duration of 16 bits of data or 100 bits of data. As the data is sampled, the microcontroller checks for a start condition, which we set to be ‘011’. We considered a longer start condition, but for our prototype the 3-bit code gave us acceptable accuracy. After the microcontroller sees the start condition, it knows exactly where to look for each bit of data, and it can then decode each successive sample as it is received. After all 16 or 100 bits of data have been received, the microcontroller can perform data recovery checks, such as parity checking, to ensure data accuracy. Then, the data is decoded into decimal form, and written out to the LCD display. For detailed code see Appendix C.

Case

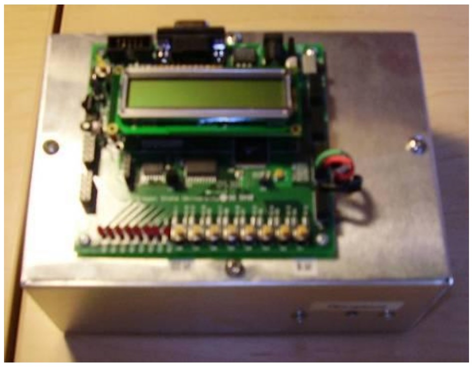

Figure 9: Photo of Case

The case for POS device is a simple metal box, which has the advantage of helping to protect the circuits inside from electromagnetic interference from the cell phone. We mounted the microcontroller board on top to allow the user to push buttons and read the display as needed. The microphone is recessed into the front of the case and accessed with a simple hole.

RESULTS

According to our testing results, feedback from our mentor, and our class professor, our project was very successful. We were able to satisfactorily demonstrate for our GE Security mentor, class professor, and Engineering expo audience a working system, capable of transmitting up to 100 bits of data (binary form) or 25 decimal numbers through a cell phone using only audio tones.

The table below summarizes the results of our official testing at the end of the senior design sequence. The detailed test procedures and pass/fail criteria were described in the “Overall System – Testing Requirements” section earlier in the paper. For all testing, we used OOK at 1250 Hz as the encoding algorithm, with a bit rate of 10 bits/second.

Table 8: Testing Results

	Description	Results	Notes
Requirement #1	A message shall be transmitted using audible tones over an audio voice call.	Pass	Used OOK, 1500Hz, 10 bits/second, 16 bit message length
Requirement #2	A 15-bit message shall be sent with an 80% probability of success.	Pass	94.4% success rate
Requirement #3	The system shall successfully decode at least one message when the cell phone speaker is within ½ inch of the microphone without using a speakerphone.	Fail	Could not reliably decode message at ½” distance without speakerphone on
Requirement #4	The system shall be able to successfully send 100-bits of data.	Pass	100-bit message @ 10 bits/second sent accurately
Requirement #5	A working prototype shall be created that uses a microcontroller to decode the audio signal and provides a way to view the decoded message	Pass	3 out of 3 16-bit messages correct when using prototype

Requirement #6	The system shall meet the minimum success requirement of 80% when a recording of the traffic sounds at 3rd and Van Buren or the ambient noise at an event similar to the senior design expo is playing near the system at the average sound pressure level that the noise was recorded.	Pass	Traffic success rate: 91.1% Voices success rate: 92.5%

During ideal conditions (no background noise, speakerphone on, cell phone held against microphone), we saw a probability of sending 15 bits of data consecutively to be around 94%. We saw little difference in the probability of success when we played our recorded background noise of traffic and people talking, with around 92% for both. We were also able to send 100 bits of data consecutively correctly more than once under ideal conditions, although it was not 100% correct every time we attempted to do so.

The only test we failed required us to send a 16-bit message at a distance of ½" without using the speakerphone on the cell phone. We were, however, able to send at either ½" with the speakerphone or very close without the speakerphone. We attempted to adjust the amplification gain and voltage thresholds in the microcontroller, but if we were able to make it work with very soft signals, the performance with any kind of background noise decreased drastically.

A common theme throughout our project was inconsistency in the cell phone network. The amount of background static and quality of reception in general changed with every cell phone call we made, and lead to some inconsistencies in our testing data. We also saw a decrease in accuracy if the background noise was extremely loud (for example, clapping near the microphone). Future versions of the project would need to include tighter filters and/or noise gates, as well as accuracy checking in the microcontroller code, to help eliminate problems.

BIBLIOGRAPHY

[1] Analog Devices, “Datasheet for SSM2166,” [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/SSM2166.pdf [Accessed: Dec 4, 2008].

[2] Atmel, “AT32AP7000 Datasheet,” October 2007 [Online]. Available: http://www.atmel.com/dyn/resources/prod_documents/doc32003.pdf [Accessed: Nov. 1, 2008].

[3] Atmel, “ATmega128 Datasheet,” June 2008 [Online]. Available: http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf [Accessed: Nov. 1, 2008].

[4] Atmel, “ATNGW1000 Network Gateway Kit,” 2008 [Online]. Available: http://www.atmel.com/dyn/products/tools_card.asp?family_id=682&family_name=AVR32+32%Dbit+MCU&tool_id=4102 [Accessed: Nov. 1, 2008].

[5] Datasheet for CUI, Inc. CMR-5054TB-A, June 2008 [Online]. Available: http://media.digikey.com/pdf/Data%20Sheets/CUI%20Inc%20All%20Brands%20PDFs/CMR-5054TB-A.pdf [Accessed Oct. 18, 2008].

[6] “Datasheet for Hitachi HD44780,” [Online]. Available: http://www.esiee.fr/~perrotol/LCD-HD44780.pdf [Accessed: Nov 1 2008].

[7] Datasheet for WM-63PRT [Online]. Available: http://industrial.panasonic.com/www-data/pdf/ABA5000/ABA5000CE3.pdf [Accessed: Nov. 1, 2008].

[8] Datasheet for WP-23502-P16, June 24, 2006 [Online] Available: http://www.knowles.com/search/prods_pdf/WP-23502-P16.pdf [Accessed: Oct. 18, 2008].

[9] GE Security, “Key Holder User’s Guide,” *GE Security Products*, [Online]. Available: http://www.gesecurity.com/portal/GESDownload?ID=1505&DID=1505&documenttype=User%20Manual [Accessed: Nov 1, 2008].

[10] Hutchinson Encyclopedia, “Acoustic Coupler,” [Online]. Available: http://encyclopedia.farlex.com/Acoustically+coupled+modem [Accessed: Nov 1, 2008].

[11] STMicroelectronics, “STM32 Evaluation Boards,” June 2007 [Online]. Available: http://www.st.com/mcu/contentid-100-110-STM3210B_EVAL.html [Accessed: Nov. 1, 2008].

[12] STMicroelectronics, “STM32 MCU family,” September 2008 [Online]. Available: http://www.st.com/stonline/products/promlit/pdf/brstm320808.pdf [Accessed: Nov. 1, 2008].

[13] Swarthmore, “Frequency Response and Active Filters,” [Online]. Available: http://www.swarthmore.edu/NatSci/echeeve1/Ref/FilterBkgrnd/Filters.html [Accessed: Dec 3, 2008].

[14] Tekbots, “mega 128 Board,” August 2007 [Online]. Available: http://eecs.oregonstate.edu/education/products/mega128.2/ [Accessed: Mar 10, 2009].

[15] TekBots Microcontroller Kit (mega 128.3), 2008 [Online]. Available: http://eecs.oregonstate.edu/education/products/mega128.2/ [Accessed: Oct. 18, 2008].

[16] Wikipedia, “Acoustic Coupler,” [Online]. Available: http://en.wikipedia.org/wiki/Acoustic_coupler [Accessed: Nov 1, 2008].

APPENDICES

APPENDIX A: NAMING CONVENTIONS AND GLOSSARY

A/D – Analog to digital

Band-pass Filter – A circuit which allows only a specific band of frequency signals to pass through without significant attenuation

D/A – Digital to analog

DSP – Digital signal processing

DTMF – Dual tone multi frequency

FSK – Frequency shift keying.

IC – Integrated circuit

IVR – Interactive voice response

Modem – A device that modulates and demodulates signals [10].

OOK – On-off keying

Op-amp – Operational Amplifier

PCB – Printed circuit board

POS – Point of sale

Resonant Frequency – the frequency at which the output of an active filter is at maximum amplitude

SPI – Serial Peripheral Interface

APPENDIX B: MATLAB CODE

This code was used to encode a string of decimal numbers into an audio signal form that can be played by Matlab.

```
function [xx] = ook(input,dur,f_on)
%input is a string of numbers and decimal points
%dur is the duration of each tone
%f_on is the frequency for a 1

binary = num2bin(input);
fs = 8000;
tt = 0:1/fs:dur;

xbuff = [];
x = cos(2*pi*f_on*tt); %start sequence - 101
xbuff = [xbuff; x(:)];
x = 0*tt;
xbuff = [xbuff; x(:)];
x = cos(2*pi*f_on*tt);
xbuff = [xbuff; x(:)];

for i=1:length(binary)
    if(binary(i)==1)
        x = cos(2*pi*f_on*tt);
    else
        x = 0*tt;
    end
    xbuff = [xbuff; x(:)];
end
xx = xbuff;
sound(xbuff,fs);


function [x] = num2bin(input)
%input is a text string of numbers and '.'
%[x] is an array of ones and zeros in row format

xbuff = [];
for i=1:length(input)
    if strcmp(input(i),'0')==1
        xbuff = [xbuff;0;0;0;0];
    elseif strcmp(input(i),'1')==1
        xbuff = [xbuff;0;0;0;1];
    elseif strcmp(input(i),'2')==1
        xbuff = [xbuff;0;0;1;0];
    elseif strcmp(input(i),'3')==1
        xbuff = [xbuff;0;0;1;1];
    elseif strcmp(input(i),'4')==1
        xbuff = [xbuff;0;1;0;0];
    elseif strcmp(input(i),'5')==1
        xbuff = [xbuff;0;1;0;1];
    elseif strcmp(input(i),'6')==1
        xbuff = [xbuff;0;1;1;0];
    elseif strcmp(input(i),'7')==1
```

```
        xbuff = [xbuff;0;1;1;1];
    elseif strcmp(input(i),'8')==1
        xbuff = [xbuff;1;0;0;0];
    elseif strcmp(input(i),'9')==1
        xbuff = [xbuff;1;0;0;1];
    else
        xbuff = [xbuff;1;0;1;0]; % decimal point
    end
end
x = xbuff;
```

APPENDIX C: MICROCONTROLLER CODE

```
#define F_CPU 16000000 // cpu speed in hertz

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <util/twi.h>
#include "lcd.h"

#define NUMBITS 100
//THLD = voltage * 50
#define THLD1 25 //.5 volts - same bit
#define THLD2 50 //1.5 volts - 1 to 0 or vice versa

volatile uint8_t counter = 0;
volatile uint16_t counter0 = 0;
uint16_t counter1 = 0;
uint16_t counter2 = 0;
volatile int start_is_seen = 0;
volatile int decode_arriving = 0;
volatile int look_for_start = 0;
volatile int waiting = 1;
uint8_t avg_data = 0;
uint8_t collected_data[17];

/*****************************************************************
                            chk_buttons
Checks the state of the button number passed to it. It shifts in
ones till the button is pushed. Function returns a 1 only once
per debounced button push so a debounce and toggle function can
be implemented at the same time. Adapted to check all buttons
from Ganssel's "Guide to Debouncing". Expects active low
pushbuttons on PINA port. Debounce time is determined by external
loop delay times 12.
*****************************************************************/
uint8_t chk_buttons(uint8_t button) {
  static uint16_t state[]={0,0,0,0,0,0,0,0};//holds present state
                                            // of buttons
  state[button]=(state[button]<<1)|(!bit_is_clear(PIND,button))|0
  xE000;
  if (state[button] == 0xF000) return 1;
  return 0;
}

/*****************************************************************
                              spi_init
Initializes the SPI port on the mega128 and sets the bar graphs
to 0.
*****************************************************************/
void spi_init(){
  DDRB |= 0x07;      //set SCK,MOSI,SS_N as output, MISO as input
  SPCR|=(1<<SPE)|(1<<MSTR);//spi enabled, master, low polarity,
                         // msb 1st
  SPSR |= (1<<SPI2X);     //run at i/o clock div 2
}

/*****************************************************************
                             timer0_init
*****************************************************************/
void timer0_init(){
  //for duration = .5 seconds or .1 seconds
  TCCR0 = (1<<WGM01)|(1<<CS02)|(1<<CS01);//prescale by 256
  //for duration = .125 seconds or .025 seconds
  //TCCR0 = (1<<WGM01)|(1<<CS02);//prescale by 64
  //for duration = .05 seconds
  //TCCR0 = (1<<WGM01)|(1<<CS02)|(1<<CS00);//prescale by 128
  OCR0 = 124;
  TIMSK |= (1<<OCIE0);             //enable interrupts
}

/*****************************************************************
                             adc_init
*****************************************************************/
void adc_init(){
  ADMUX |= (1<<REFS0)|(1<<ADLAR); //use avcc, left adj
  ADCSRA |= (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}

/*****************************************************************
                              sample
Gathers data from A/D converter
*****************************************************************/
uint8_t sample(int channel){

 ADCSRA |= (1<<ADSC);
 while(bit_is_clear(ADCSRA, ADIF)){} //spin until done
 ADCSRA |= (1<<ADIF); //its done, clear flag
 return ADCH; //return sampled value
}

/*****************************************************************
                              start
Looks for patter of high low high high to indicate start of
Signal. Returns 1 if start sequence has been found, 0 otherwise
*****************************************************************/
int start(){
  int i;
  static int prelim = 0;
  if(counter2 < 13){
    if(counter2 == 1) PORTB |= (1<<PB7);
    collected_data[counter2 - 1] = avg_data;
    if(counter2 == 12) prelim = 1;
  }
  else{
    for(i=0; i<11; i++){
      collected_data[i] = collected_data[i+1];
    }
    collected_data[i] = avg_data;
  }
  if(prelim){//check start condition
    if(abs(collected_data[0]-collected_data[1]) < THLD1){
      if(abs(collected_data[5]-collected_data[6]) < THLD1){
        if(abs(collected_data[10]-collected_data[11]) < THLD1){
             if((abs(collected_data[1]-collected_data[6])>
                 THLD2)&(collected_data[1] > collected_data[6])){
              if((abs(collected_data[6]-collected_data[11])>
                   THLD2)&(collected_data[6]                        <
           collected_data[11]))){
                  collected_data[1] = collected_data[11];
                  counter1 = 0;
                  counter2 = 0;
                  PORTB |= (1<<PB6);
                  prelim = 0;
                  return 1;
              }
           }
        }
      }
    }
  }
  return 0;
}

/*****************************************************************
                             decode_data
****************************************************************/
char decode_data(char prev){
  collected_data[0] = collected_data[1];
  collected_data[1] = avg_data;
  if(abs(collected_data[0]-collected_data[1]) > THLD2){
    if(collected_data[0] > collected_data[1])
      return '0';
    else
      return '1';
  }
  else
    return prev;
}

/****************************************************************/
char *bin2dec(char* binary_str, char* decimal){
  int k;
   int  place_value, index, m, bit;
   int  sum = 0;
   int num_nibbles = NUMBITS/4;

   for ( k = 0; k < num_nibbles; k++){
         char buff[5];
         strncpy(buff, &binary_str[k*4], 4);
         for (index = 0; index <= 3; index++){
               bit = (buff[index] - '0');//convert char to numeric
                                           // value
               place_value = 1;  // initialize or reset the
                                  // place_value
               for(m = 3; m > index; m--)
               {
                     // 1 2 4 8 16 32 64 ... place_values,
                     // reversed here
                     place_value *= 2;
               }
               sum = sum + bit * place_value;
         }
         sprintf(&decimal[k], "%X", sum);
          sum = 0;
   }
   return(decimal);
}

/*****************************************************************
            Timer0 Compare Interrupt Service Routine
*****************************************************************/
ISR(TIMER0_COMP_vect){
 //if duration = .125 or .5, need counter and % 5
 //if duration = .1, .05, .025 just sample every interrupt
 static uint16_t temp0 = 0;
if(1){
   counter0++;
   temp0 = temp0 + sample(0);
   if((counter0 % 10) == 0){
     avg_data = temp0/10;
     temp0 = 0;
     if(start_is_seen){
      counter1++;
      if((counter1 % 5) == 0)
          decode_arriving = 1;
     }
     else{//look for start
       look_for_start = 1;
       counter2++;
     }
     waiting = 0;
   }
 }
}
/*****************************************************************/
int main(){
 int data_being_transmitted = 1;
 int i,j;
 uint8_t button_pressed;
 char decoded_bits[NUMBITS+1];
 char* decimal_data;
 decimal_data = malloc((NUMBITS/4+1)*sizeof(char));
 char line1[17];
 char line2[17];

 //set LCD enable bit as output
 DDRF |= (1<<DDF3);
 PORTF &= 0xF7;

 DDRB |= 0xF0; //LED's 5-8 as output

 spi_init(); //initialize SPI
 lcd_init(); //initialize lcd display
 timer0_init(); //initialize sampling timer
 adc_init(); //initialize adc

 while(1){
   _delay_ms(2);    //debounce delay
   button_pressed = 8;
   for(j=0;j<8;j++){
     if(chk_buttons(j) == 1)
       button_pressed = j;
   }
   if(button_pressed == 0){
     data_being_transmitted = 1;
     sei();
     i=0;
     while(data_being_transmitted){
       while(waiting){}//wait for sample to be taken and averaged
       if(counter1 > NUMBITS*5){
          cli();
          decimal_data = bin2dec(decoded_bits, decimal_data);
          decimal_data[NUMBITS/4] = '\0';
          decoded_bits[NUMBITS] = '\0';
          strncpy(line1, decimal_data, 16);
          strncpy(line2, &decimal_data[16], 16);
          line1[16] = '\0';
          line2[16] = '\0';
          clear_display();
          cursor_home();
          string2lcd(line1);
          home_line2();
          if(NUMBITS <= 16){
            string2lcd(decoded_bits);
          }
          else
            string2lcd(line2);
          PORTB &= 0x3F;
          data_being_transmitted = 0;
          counter0 = 0;
          counter1 = 0;
          look_for_start = 1;
       }
       else if(look_for_start){
          start_is_seen = start();
          look_for_start = 0;
       }
       else if(decode_arriving){//decode signal
          if(i==0) decoded_bits[i] = decode_data('1');
          else decoded_bits[i] = decode_data(decoded_bits[i-1]);
          decode_arriving = 0;
          i++;
       }
       waiting = 1;
     }//while data_being_transmitted
   }//if button_pressed == 0
 }//while
}//main
```

APPENDIX D: SECURITY RISK DISCUSSIONS

Security Risk: Recording a valid transaction, and playing it back at a later time.

Overview: The purpose of this discussion is to suggest a way to protect against breeching the security of the point of sale system by recording a valid transaction and playing it back at a later time to add money to the meter without paying for it.

The simplest way to protect this security risk is to include an n-bit counter in each transaction. The computer system behind the scenes would increment the counter with every new transaction, and send it as part of the encoded message. The microcontroller would decode the counter value, and check the value of the counter against the value stored in memory. If the value is greater than the counter value in memory, but not more than a defined value (say 3) away, then the transaction is valid and not a recording. If the counters were slightly out of synch but still valid, the microcontroller would set its counter value to be equal to the value received from the computer system at the end of a successful transaction. There could potentially be a problem when the counter reaches it maximum value (2^n-1) and rolls over to zero, because if someone recorded a transaction, and then played it back 2^n transactions later, the microcontroller would think it is valid. Of course, a larger counter means a smaller chance that someone would play back a recording at exactly the right time.

One way to eliminate the chance that someone would play back the correct counter value at a later time is to include the date and time in every transaction. The computer system that links user accounts and parking meter information would have a running clock and calendar. When an audio message is sent to the parking meter, the first part of the message would be the encoded date and time, followed by the actual data being sent. The microcontroller within the meter would decode the date and time, and compare it to the date and time of the most recent transaction it has in memory. If the new date and time is later than that of the stored date and time, the microcontroller would know that the message is not a recording, and is valid. This scheme only requires the microcontroller to store one date and time in memory, and does not require the microcontroller to be in synch with the computer system.

For either scheme, the microcontroller wouldn't save the new counter value or date/time in memory until it had confirmation that the transaction was correct and complete. Also, in the counter case, the computer system wouldn't increment the counter until the current transaction was complete. This way, if the same message has to be resent because of transmission error, there wouldn't be a problem with the microcontroller thinking duplicate legitimate messages are false. Confirmation of a successful transaction on the microcontroller end could consist of a checksum or parity check (to check internally that the data received is correct) and/or the user pressing a button confirming that the data that was decoded is what the user wanted. If a 4-bit checksum were implemented, there would be 16 possible values for the checksum field. So, if part of a message were incorrect, there would be a 15/16 or 93% chance that the microcontroller would catch the error. If an 8-bit checksum were implemented, there would be a 255/256 or >99% chance that the microcontroller would catch error in a message. After all the checks, if there was error in the transaction, the microcontroller would request that the message be resent.

While sending the full date and time eliminates the chance that someone would play back the correct counter value at a later time, it would take at least 44 extra bits (or

48 bits if second values are sent) to send the complete date and time. This is almost three times longer than the actual data in the message (16 bits). As the message gets longer, the probability that an entire message is correct decreases drastically, so just using a counter would probably be more practical. Even using as short as a 7-bit counter would lead to a less than one percent chance that randomly playing back a recording at a later time would have the correct counter value.

Security Risk: Spoofing the system with a random tone generator.

Overview: The purpose of this discussion is to suggest a way to protect against breaching the security of the point of sale system by using a random tone generator to generate a correct sequence of tones and fraudulently add money to the meter.

Consider the probability of generating the correct sequence of tones. Assume that the system uses either an OOK or FSK modulation scheme and that a valid sequence of tones has the following format.

Start sequence	3-bits (e.g. 111 or 101 in binary)
Data	16-bits (four 4-bit numbers)

If the tone generator is designed to generate only the frequencies and tone duration that the system operates at, then the only challenge to breaching the security of this sequence is generating the correct start bits. The likelihood of correctly generating the start bits is

$$P = (0.5)(0.5)(0.5) = (0.5)3 = 0.125 .$$

With a baud rate of 8 (tone duration = 0.125 seconds), it would take 0.375 seconds for each attempt at generating the sequence and one in eight attempts would be successful. That means it would take only about three seconds to breach the system. To improve security, the sequence could be altered to include a serial number immediately following the start sequence. A table comparing the serial number length and the baud rate to the seconds required to breach the security of the system for several serial number lengths is shown below.

		Maximum time (in seconds) required to breach the system at various baud rates		
Combined length of start sequence and serial number (in bits)	Number of attempts required	Baud rate = 8 (0.125 seconds/bit)	Baud rate = 20 (0.05 seconds/bit)	Baud rate = 100 (0.01 seconds/bit)
4	16	8	3.2	0.64
5	32	20	8	1.6
6	64	48	20	3.9
7	128	112	45	9
8	256	256	103	21
9	512	576	231	46
10	1024	1280 (> 21 mins)	512	103
15	32768	61440 (> 17 hrs)	24576 (> 6 hrs)	4916 (> 1 hr)
19	524288	1245184 (> 14 days)	498074 (> 5 days)	99615 (> 1 day)

A 19-bit combined length, 3-bit start sequence and a 4-digit (16-bit) serial number, significantly improves the security of the system. It would take at least 24 hours to breach the security of the system using a random tone generator.

On average, an attacker would breach the system in half the maximum time. If the system were used on a parking meter with a maximum time limit of two hours, then it would be sufficient to design the system to take an attacker at least a maximum of four hours to breach the security. However, without an additional security measure, such as encryption, the time required to breach the system is not adequate security to protect against random tone generator attacks. If the serial number were encrypted using the date and time, the encrypted serial number would change frequently and an attacker would have to spend at least two hours generating random tones each time he/she wanted to steal two hours of parking time.

Adding an encryption scheme that uses the date and time introduces a new problem of time drift. If the time on the server and the time on the parking meter drift apart, then legitimate tones from the server will be denied by the parking meter. Updating the encryption scheme every hour and syncing the time on the parking meter with the time on the server once a month should solve the time drift problem. Updating the encryption only once an hour means the time on the server and the parking meter could have a 60 minute difference and still work. They could drift apart by almost two minutes per day. Updating the encryption once an hour also means an attacker would have a maximum of one hour to use the sequence of tones before they expire.

Security Risk: Making a phone number appear to be a different number to a caller-ID system

Overview: The purpose of this discussion is to suggest a way to protect a way against callers to lie about their identities, and present false names and numbers to the point of sale system which will make people add money to the meter with someone else’s money.

These days spoofing a phone number is a very easy thing to do. Many phone companies now actually offer caller-ID spoofing as a service that people can pay for. To use it people just simply call the toll free access number like any other calling card. Instead of only being asked what number he wants to call, he is also asked what he wants his caller-ID to be. The service will place the call and caller-ID information will be sent exactly as he wants it to be. So now spoofing a phone number is actually legal.

One way to protect customers’ identity to the parking meter is to use PIN or password. When using a cell phone to call the parking meter network to pay, the network will ask the user to input a PIN number through the keypad on the cell phone for identification purpose. If a PIN number is set to be four digits in length, the probability of an attacker can guess the four digits right would be 1 out of 1000 chance. And four digits are not too long for a customer to remember for his own authentication purpose. Therefore a PIN with four digits is a good choice.

It is also important for the network system to set a regulation time, which would be three times in general. This means that a user has three chances to input the correct PIN number, just in case the user cannot clearly remember his PIN. And for an attacker trying to find the right combination, three times is too few to get it right.

Now within the usage of the regulated time, if the PIN is correct, then the user will continue the process to pay for the parking. If the PIN is not correct, and the user has used all three regulated time, the phone will automatically end and will not continue to pay for the parking. For security purpose, if all three chances have failed, the system will automatically disable the account temporarily until the customer calls back the network to re-enable this account. This way, even if someone intend to use caller-ID spoof to be someone else, the network will have a way to double check on the current phone holder and soon figure out if it matches the authentication of the real customer.