CELL PHONE SECURITY SYSTEM

A Design Project Report
Presented to the Engineering Division of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering (Electrical)

by
Jason Chiang
Project Advisor: Bruce Land

Degree Date: May 2006

Abstract

Master of Electrical Engineering Program
Cornell University
Design Project Report

Project Title: Cell Phone Security System

Author: Jason Chiang

Abstract: Keyless entry has been a luxury whose availability is confined primarily to vehicles. The cell phone security system takes this idea of keyless entry and transforms it into a convenient, versatile security system that utilizes cell phone technology and the landline telephone network. By taking advantage of caller identification and dual-tone multi-frequency signaling, the security system has the ability to introduce two-levels of security. The first level will be decoding the calling party's identification information while the second level would consist of the user attempting a password entry over the phone. The system also has the ability to provide feedback to the user regarding the state of the system through a special user mode. By combining the mobility of this telecommunication medium with microcontrollers, the system achieves a secure, convenient, and automated form of security for a place of residence.

Report Approved by

Project Advisor: ____Date: ____________

Executive Summary

The cell phone security system was able to achieve its goals of introducing an automated and versatile security system using the telephone network through principles of caller identification and DTMF signaling. It successfully interfaced with both the caller ID and DTMF protocols. By utilizing the HT9032C Caller ID Receiver, the system is able to successfully decode the caller ID data packets sent on the phone line. This decoded caller ID information is then used to verify that the user is part of the preset access list. If the user is not on the access list, they are denied access immediately and disconnected. In the event that the caller is on the access list, the caller is then allowed to attempt a password entry.

The HT9170B DTMF Receiver allowed the system to decode the DTMF signals sent by the user. These decoded DTMF signals had two purposes. They served as entries towards a password attempt as well as input from the user to control the system. User passwords were allowed to be of arbitrary length, however for testing purposes, I hard-coded it to be simply '12345678.' The terminator button was set to be the '#' button. So the user would enter the password in the following sequence: 1-2-3-4-5-6-7-8-#. The system also gave the caller the option of entering and special user mode where they had the ability to control the security system by pressing buttons 1 – 3. The caller would have the ability to check the status of the system, lock the door, or unlock the door.

Pushbuttons were implemented to allow the user to automatically lock or unlock the door at the push of a button.

In an effort to have complete control over my testing environment, I designed a simulation of a phone network to perform my testing on. I decided to implement my own phone network rather than interface with a real phone line to gain a level of control as well as a completely isolated, dedicated phone system. This simulated phone system proved to be an exact replica of a real phone line, conforming to all applicable telephony standards used in the project. My simulated phone line added safety and control to the development of the cell phone security system. The result was a successful cell phone security system I had envisioned.

Table of Contents

1. Introduction

The cell phone security system is the result of a fusion of a creative idea with an attempt to motivate change. Even though modern technology has allowed for the automation of many aspects of domestic lifestyles, from automatic motion sensing lights to automatic garage door openers, home security has not seen much benefit from this revolution. Household entry has long been a very manual routine with little effort to automate the process. Entry into a residence is still primarily limited to a manual process which involves inserting a key into a bolt and physically moving the locking mechanism.

The cell phone security system aims to change this. The system takes advantage of the widespread acceptance of cell phones in today's society in conjunction with the deep-rooted standards of the landline telephone network to introduce automation and convenience. The system will allow a user to use their cell phone to place a call into their home security system. Once the system verifies the caller, the caller is then allowed to attempt a password entry. Upon successfully entering a password, the system will automatically unlock the door and grant entrance. The caller also has the option of entering a user mode where they will be able to check the status of the system (locked or unlocked), lock the door automatically, or unlock the door automatically. This automation introduces a form of secure, keyless entry into a residence along with the convenience of a fully responsive security system monitor.

The system will primarily interface with telephony protocols which include dual-tone multi frequency (DTMF), caller identification (CID), and some applicable telephony circuit standards. Caller identification will be used in the first level of security to verify the caller and DTMF will be used in the second level to decode the button presses that form the password. The telephony standards are necessary for implementing a correct interface that will allow the phone line to go on/off-hook. Properly understanding and interfacing with these standards is of utmost importance due to federal regulations as well as successful operation.

2. Design Problem & Requirements

The fundamental requirements for the cell phone security system remained fixed throughout the design process. My goal was to design a system which would allow the user automated and convenient access to their home security system through a telephone network. The fundamental objectives of the system include:

1. Correctly decode DTMF signals from the user
2. Correctly decode caller identification information from the phone line
3. Allow the user to automatically lock the entryway
4. Allow the user to automatically unlock the entryway
5. Allow the user to check the status of the entryway
6. Conform to all telephone standards (section 3.)

2.1. Design Constraints:

In addition to meeting these goals, I also intended to create a design that would come as close to a real-world product as possible. This meant creating an interface with a live telephone line that a user could just take the design and plug it into a phone line and the system would be ready to go. This interface would have to conform to the federal communication commission's (FCC) standards for use of their telephone networks. During the design and implementation of the system, certain constraints arose that limited my ability to successfully meet this desire to implement a real-world interface. Specifically, limitation on resources available would not allow me to test my design using a real telephone line. In addition, safety is a big issue when testing a design on a live phone line. Since the phone network can transmit signals exceeding 100V, safety is a major consideration.

As a result, an added level of complexity was added to my project, which was to simulate a phone line. Telephone networks are vastly complex, so I only simulated to applicable components, which included DTMF and CID. The details of these simulations will be covered in the design section (5).

3. Telephony Background

The telephone lines in the United States are a federally regulated network that includes countless protocols and standards. These protocols define every aspect of their telephone lines from the impedance of the interface to the signals allowed on the lines. The following is a brief overview of some of the protocols that are applicable to this project.

3.1. United States Phone Line Interface

In an effort to protect the phone networks, FCC has places several regulations on all efforts to interface and interact with phone lines. These regulations range from impedance matching to hearing aid compatibility. Since this project will not require much telecommunication over the phone networks, many of the standards do not apply. Therefore, discussion of only the standards and regulations that do apply will follow:

- <u>Isolation</u> – FCC regulations state that any circuitry connected to their phone lines must be isolated from their network. This is primarily an effort to keep potentially hazardous circuits from damaging their infrastructure. In order to comply with this regulation, a 600:600 ohm telephone coupling transformer can be used to isolate circuitry from the phone line.
- <u>On-hook Impedance</u> – The DC on-hook impedance of any connected device must be greater than 5 mega ohms. To comply with this rule, a simple switch can be used to toggle between the on-hook high impedance circuit and the off hook circuit.
- <u>Off-hook impedance</u> – The DC off hook impedance of any connected device must be 200 ohms while the AC off-hook impedance must be matched to 600 ohms. To comply with this rule, a simple switch will be used to toggle between the on-hook circuit and the off hook matched 600 ohm circuit.

3.2. Caller ID

Calling Number Delivery (CND), better known as Caller ID, has several standards. Each standard utilizes different technology and is used in different countries. These different standards include Bellcore, DTMF, and CCA. Bellcore is the standard used in the United States, so discussion of caller ID will revolve around the Bellcore standard.

The Bellcore standard specifies the timing of data transmission as well as the data packet information itself (including headers and stop bits). The caller ID information is transmitted between ring bursts. The diagram for the required data transmission for caller ID specified by Bellcore standards is as follows:

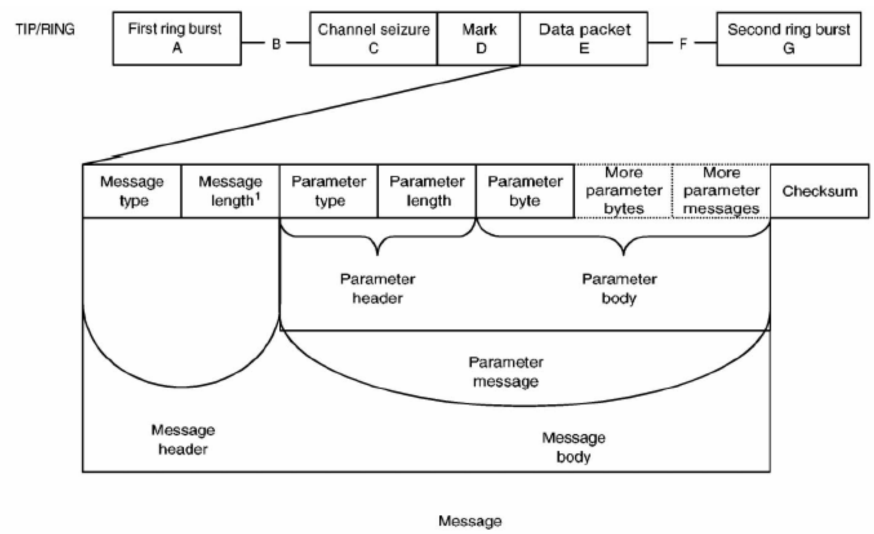


Figure 1. Bellcore Physical Layer Signaling (Type I Application)

The legend to the figure above is described as follows:
1: Message length equals the number of bytes to follow in the message body, excluding the checksum
A: First 2-second ring burst (nominal 0.2~2.2 seconds)
B: At least 0.5 second silent period between the first ring burst and the start of data transmission
C: 300 alternating mark and space bits
D: 180 mark bits
E: Time available for sending data (C+D+E=2.9~3.7 seconds)
F: At least 0.2 second before sending the 2nd ring burst
G: Second ring burst

Figure 1. Caller ID Data Transmission

The details of the transmission are as follows:

- **Channel Seizure** – 30 continuous bytes of 0x55 (01010101).
- **Mark** – Consists of 130 +/- 25 ms of mark data to prepare the receiver for data transmission.
- **Message Type** – Indicates the type of service and capability of the system. For caller ID, this will be 0x04.
- **Message Length** – The number of bytes that will follow (excluding the checksum).
- **Parameter (Data)** – Encoded in ASCII

- **First two bytes** – Month
  - **Next two bytes** – Day of month
  - **Next two bytes** – Local military time (hour)
  - **Next two bytes** – Local military time (minutes after the hour)
  - **Remaining words** – Calling party's directory number
- **Checksum** – The checksum is the modulo-256 sum of the received words. This can be used to detect errors in transmission. If there was an error in transmission, caller ID does not support re-transmission. So the system would need to wait for the transmission between the second and third ring bursts.

3.3. Dual-Tone Multi Frequency (DTMF)

DTMF has become the standard for sending signals over the telephone line. Prior to DTMF, telephone networks used a technique called pulse dialing or loop disconnect. Pulse dialing was a method which worked by rapidly connecting and disconnecting the phone line to form the desired signals. This technique had several limitations including signal degradation over long transmissions. In an effort to improve upon pulse dialing's flaws, Bell Labs created dual-tone multi frequency. DTMF proved to be a much more versatile and effective method at transmitting signals over the telephone networks.

A DTMF signal is generated by the real-time linear combination of two sine waves; one low frequency and one high frequency. The DTMF protocol gives the frequency definitions for the standard 4 x 4 keypad. The frequencies for these 16 keys are:

	1209 Hz	**1336 Hz**	**1447 Hz**	**1633 Hz**
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

Table 1. DTMF Frequency Values for 4 x 4 Keypad

For example, pressing '1' would produce a 1209 Hz sine wave linearly added with a 697 Hz sine wave. The frequencies were chosen such that it would be very difficult to reproduce by a human voice, thereby avoiding potential complications during speech. The following are the specifications for the frequencies that correspond to aspects of the phone line other than the key-presses:

Event	**Low Frequency**	**High Frequency**
Busy Signal	480 Hz	620 Hz
Dial Tone	350 Hz	440 Hz

Ringback Tone	440 Hz	480 Hz

Table 2. DTMF Frequency Values for Other Aspects

4. Range of Solutions

In every design, there is always a range of alternatives that could be utilized in arriving at a solution. These alternatives can be in the form of components, approaches, or algorithms. Some design alternatives for the cell phone security system include the following:

4.1. Physical Locking Mechanism

Actual Implementation – A simple interface using LEDs was used to inform the user whether the system was in the locked state or unlocked state. I chose the most intuitive colors to represent each state, red for locked and green for unlocked. The following is a picture of this interface:

Figure 2. Locked & Unlocked LEDs

Alternatives - There exists a wide variety of choices for the actual locking mechanism. These options range from magnetic, solenoid, or motor driven locks. There are many advantages and drawbacks to each choice. However, the deciding factor is both price and importance. I decided against using electric or magnetic driven locks after doing research and discovering they cost in the hundreds of dollars. However, if this was a real product to be released (and we had funding) an electric or magnetic lock would be very nice and look something like:

Electrical bolt *Magnetic lock*

Figure 3. Locking Mechanism Alternatives

4.2. Locking the Door (when not using a cell phone)

Actual Implementation – I decided to implement pushbuttons that would allow the user to either lock the door or unlock the door at the push of a button. I also chose to color the buttons such that it would be intuitive which button did what; red locks the door, green unlocks it. (note: the first button is unimplemented):

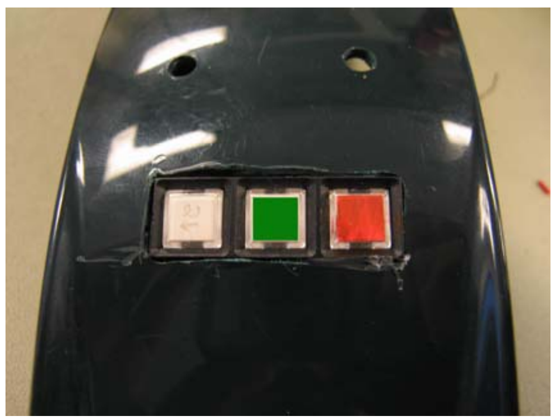


Figure 4. Lock/Unlock Pushbuttons

Alternatives – Alternatively, the system could allow the user to lock the door by moving a lever or some sort of bolt. This alternative would require sensors to detect the position of the bolt in order to inform the system control so it always knows the state of the lock. This alternative is just as effective, but was ultimately decided against because of the nature of the rest of the system. Since the project's goal is to deliver convenience and automation, an automatic locking mechanism was favored.

4.3. Idle Behavior

Actual Implementation – No idle behavior was implemented. In the event that they system was in either locked or unlocked state, it would stay in that state until an external event triggered a change, such as pushing a button or a caller activating the system to lock or unlock the door. I decided not to incorporate any idle behavior because of the possibility of accidental lock-out. An idle mode action would make sense in a more professional setting, such as an business office where a lab room would lock after inactivity. But my intent was to have this deployed in the residential setting. Locking the user out of their house may get frustrating.

Alternative – The system could have the ability to turn on and off this idle mode behavior. Allowing the user to enable or disable this feature would give the flexibility to prevent this lockout while giving users who desire this feature to utilize it. Either way, an 'always-on' lockout feature was ultimately not a good idea so I decided to scrap the entire idea and go with simplicity.

5. Design & Implementation

5.1. Simulated Phone Line

Due to constraints on resources available to me, I did not have access to a live telephone line that I could test my design on. As a result, the requirements for my project evolved to include the need to design and simulate a telephone line that would be as close to a real phone line as possible. This simulated phone system would have to provide identical signals to a real phone line in order to be convincing of an accurate simulation. As already discussed, the telephone networks are extremely complex and as a result, I only simulated the components that were applicable to this project. In specific, it was necessary for me to simulate the caller identification as well as the DTMF dialing.

5.1.1. Caller ID Simulation:

As discussed in the telephony background section (3), caller identification information is transmitted over the phone line as an ASCII encoded data packet sent between ring bursts. The information contained within this data packet varies depending on the service provider. However the calling party's phone number is the most important field and is always transmitted. It is rather simple to simulate the caller ID data packet sent by the telephone provider. By simply using an ASCII encoding scheme, it is possible to encode all the information pertaining to the user such as calling number, month, day, year, etc.

My implementation of this simulation is somewhat simplified, however. Since I know that the caller ID information is simply a stream of binary bits which represents the ASCII encoded information, I simplified this in an effort to make testing easier. Since encoding the data packet to include all that information could render the data packet illegible by humans, I simplified the data packet to simply be the phone number of the calling party. For example, the phone number (555) 123-4567 would be encoded as '5551234567' in my simulation. This simulation, while simplified, is still a valid representation of the caller ID protocol used on real telephone networks. As long as it is understood that there is some more information within the packet, it seems acceptable to simulate CID in this fashion. Discussion of the interaction between this data and the hardware/software will be discussed in the Hardware and Software sections.

5.1.2. DTMF Simulation

The next component of the telephone line that I was required to simulate was the DTMF signals transmitted over the phone line that represented the keys pressed by the user. I accomplished this simulation using the PWM on the Mega32 MCU. As we know, DTMF signals are comprised of a low frequency sine wave component linearly added with a high frequency sine wave component. To generate these sine waves, I used an algorithm which uses an increment value that is added to an accumulator which in turn is used to index into a sine table to calculate an overflow value (OCR0) for the timer 0 interrupt. The accumulator was a 32-bit long value which was calculated by the following:

accumulator = accumulator + increment

The increment was also a 32-bit long value and was calculated using:

$$\text{increment} = \frac{f \cdot 2^x \cdot N}{CLK}$$

- f = desired sine wave frequency (in hertz)
- x = number of bits in accumulator (32 bits)

- N = number of CPU clocks per PWM cycle ($256 = 2^8$)
- CLK = system clock (16 MHz)

The high byte of this 'accumulator + increment' value was taken as the index into the sine table which held values of coefficients for 256 discrete values of sine between 0 and 2π. The sine table was calculated at initialization and utilizes the math library's sin() function to compute coefficients. The following code segment computes these 256 discrete sine table entries (coefficients):

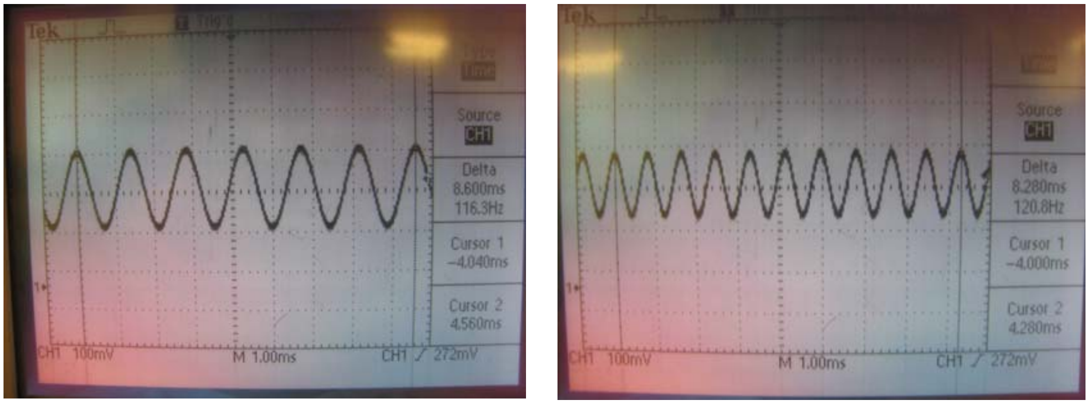
```
for(i = 0; i < 256; i++)
begin
     sineTable[i] = (char)(63.5 * sin(6.283*((float)i)/256.0));
end
```

As we saw before, the DTMF table for the standard keypad assigns frequencies to every row and column. In the simulated phone model, I used a 3 x 4 keypad rather than a complete 4 x 4 keypad. The reason behind this decision was because a 3 x 4 keypad is closer to a real phone keypad that consists of the digits 0 – 9 and the characters "*" and "#." Therefore, the increment values for the applicable frequencies were:

Frequency (Hz)	**Increment**
697	47,897,475
770	52,913,997
852	58,548,994
941	64,665,028
1209	83,081,847
1336	91,809,221
1447	99,437,083

Table 3. Increment values for DTMF frequencies

Using these values to generate the signals, I was able to obtain a very clean, precise sine wave. Here are two screen shots of the sine waves I produced for button 1, which include a 697 Hz sine wave + 1209 Hz sine wave:



697 Hz 1209 Hz

Figure 5. Sine Waves Generated

It can be seen from the oscilloscope readout, the (ideally) 697 Hz signal produced is $f_{697} = 116.3 * 6 =$ <u>697.8 Hz</u> (an error of $\frac{697.8-697}{697}*100 = 0.115$ %) and the (ideally) 1209 Hz signal produced is $f_{1209} = 120.8 * 10 =$ <u>1208 Hz</u> (an error of $\frac{1208-1209}{1209}*100 =$ 0.0827%). These error ranges are well within the acceptable deviations from the ideals, which is 1.5%. However, it is important to note that these errors present on the signals are not necessarily a result of inaccurate generation. Rather, the errors are most likely a result of the oscilloscope's error range. The timer-driven, hardware generation of the DTMF signals are highly precise.

Linearly adding these two sine components generates the complete DTMF signal for "1," which is as follows:

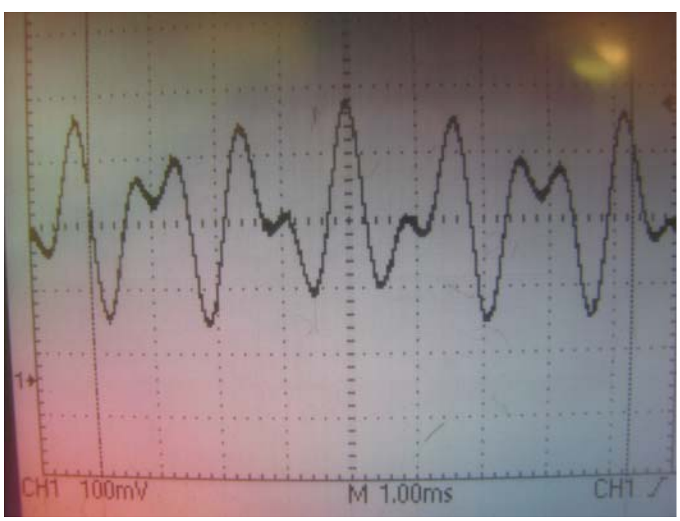


Figure 6. DTMF signal for "1"

(A complete catalog of all DTMF signals is included in the appendix)

This DTMF generator was packaged within an old telephone receiver. I was able to utilize the speaker as an audio output for the DTMF signals. As a result, the simulated phone system acted exactly like a real telephone. A status LED was used to indicate when the phone was turned on/off. The output from the DTMF generator was sent through a standard RJ-11 output connector at the bottom of the phone. The following is an image of this connector on the simulated phone:

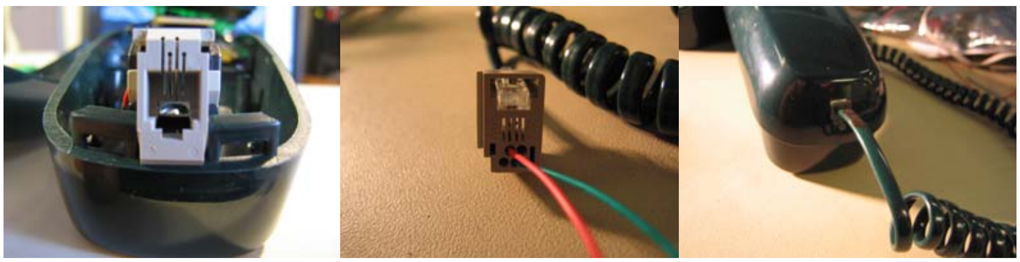


Figure 7. RJ-11 Connector

My final result was a completely self-contained simulated phone system that acted exactly as a real phone would. Audio output simulated real dialing feedback and the RJ-11 connector acted as a real-life interface with the system (tip and ring lines).

Figure 8. Simulated Phone

5.1.3. Not Simulated

In creating a simulated phone system, I decided that some aspects of the telephone network were not necessarily important to simulate. One component that was not simulated was the on-hook/off-hook impedance requirements. The reason this aspect of the phone line was not simulated was because of it would be impossible to create an artificial network that would react like a real-world phone system. As discussed in the telephony background, the telephone network detects whether a line is on-hook or off-hook by detecting changes in the effective impedance on the specific line. This detection of impedance and current draw is accomplished using a complex series of sensor circuits that can monitor sections of the phone line to detect these changes. This is not to say that my circuit does not comply with these specifications. My circuits take this into account and the necessary hardware to achieve isolation is available. In order to achieve isolation, I obtained a matched 600:600 ohm transformer. This transformer would be attached between my circuitry and the tip and ring lines.

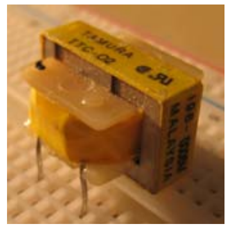


Figure 9. Transformer

5.2. Hardware

5.2.1. Mega32 Microcontroller

The Atmel Mega32 microcontroller was used as the main component in this lab. The mcu was set up and soldered onto the standard PCB layout offered by Professor Land for ECE 476. Using the dremel, excess parts of the PCB board were cut off in an effort to allow packaging (discussed in packaging section 6). A serial connector and a MAX233 serial interface controller were also attached to allow for caller ID transmission. The ports were set up as follows:

PORT	PIN	Direction	Purpose
A	0	Input	Pushbutton Input (Lock)
	1	Input	Pushbutton Input (Unlock)
	2…7	-	Not used
B	0	-	Not used
	1	Output	Status LED for locked state
	2…6	-	Not used
	7	Output	Status LED for unlocked state
C	0…3	Input	4 bit decoded DTMF value from HT9170B
	4	-	Not used
	5	Output	Power-down output to DTMF decoder
	6	-	Not used
	7	Input	Valid bit from DTMF decoder
D	0…7	-	Only D.0 & D.1 used for serial jumper

Table 4. Port Setup

LEDs were connected to their appropriate ports to show the state of the system. 330-ohm resistors were used to limit the current to the LEDs to about 20 mA. A power LED was also implemented to indicate when the system is on/off. The following is the final PCB with the mcu, status LEDs, power switch, power adaptor, and serial controller and connector:

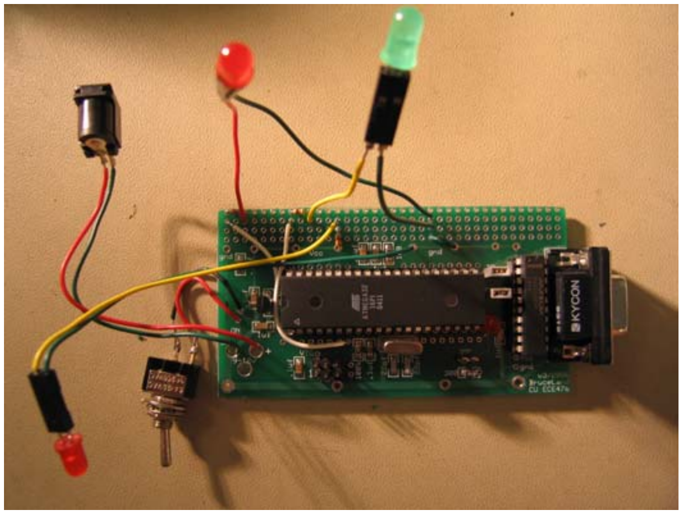


Figure 10. PCB Board with MCU and Components

5.2.2. Switching Circuitry

In order to switch the phone line from the caller ID circuit to the DTMF circuit, a switch can be used. I decided to use a single-pole double-throw (SPDT) switch, as this would satisfy the needs of my circuit. A single-pole double-throw type of switch would connect one line to either of two other lines. I obtained the ADG636 from analog devices which is a "Dual SPDT Switch." A schematic of a SPDT switch is as follows:

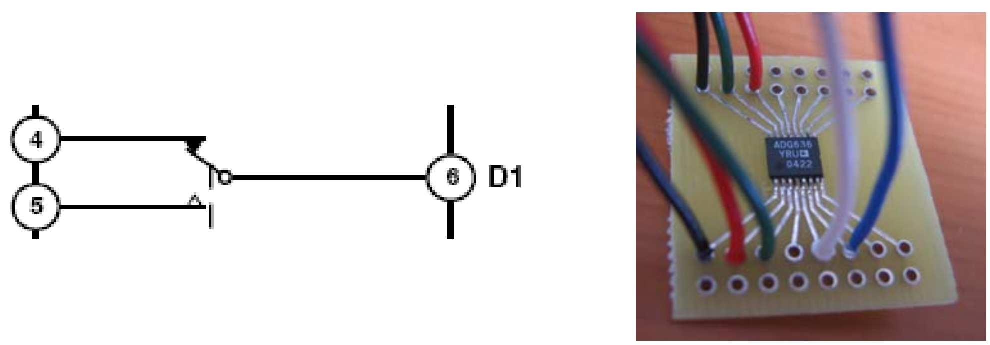


Figure 11. ADG636 SPDT Switch

D1 would be connected to the Tip line and D2 would be connected to the Ring line. S1A and S2A would be connected to the caller ID circuit and S1B and S2B would be connected to the DTMF decoding circuit. The enable pin was always high (enabling the outputs) and the logic table is as follows:

A0	A1	Circuit
0	0	Caller ID
0	1	Not Used
1	0	Not Used
1	1	DTMF

Table 5. Switch Logic Table

This switch successfully allowed the tip and ring signals to be switched from the caller ID circuit to the DTMF circuit. In the end, I took the switch out of the circuit and wired the tip and ring lines directly to the DTMF decoder. Since I was interfacing with a simulated telephone system, the caller ID decoder component did not directly interface with the tip and ring lines. Instead they interfaced with a serial connection to a PC to simulate the caller ID. As a result, only the DTMF decoder would directly interface with the tip and ring lines. I decided to remove the switch since the second pole would not be connected to anything. However, I was able to verify that this switch was functional and worked as I wanted in the circuit.

5.2.3. Caller ID Decoding

In order to decode the caller ID information, I used the HT9032C. The HT9032 is a Calling Line Identification Receiver, which extracts the caller ID data packet from in between the ring bursts and sends it as an output serially to an MCU. The following is a pin out of the HT9032C IC:

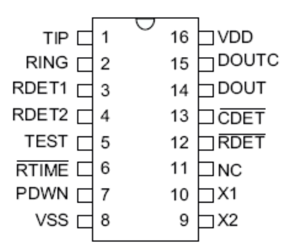


Figure 12. HT9032C Pin out

The serial data is transmitted at a rate of 1200 baud on the output pin 14. Pin 7 is a power-down pin that could be used to save some power. In this system, it would not make sense to power down the caller ID unit since it must monitor for incoming calls at all times; the system cannot miss any calls. The only time this power-down functionality can be used is when a user is verified and control is passed along to the DTMF circuitry. In this case, the caller ID decoder is no longer needed for that period of time and can be powered down. After the DTMF component completes, control is handed back to the caller ID unit and the power-down is disabled. The circuit to decode caller ID information was as follows:

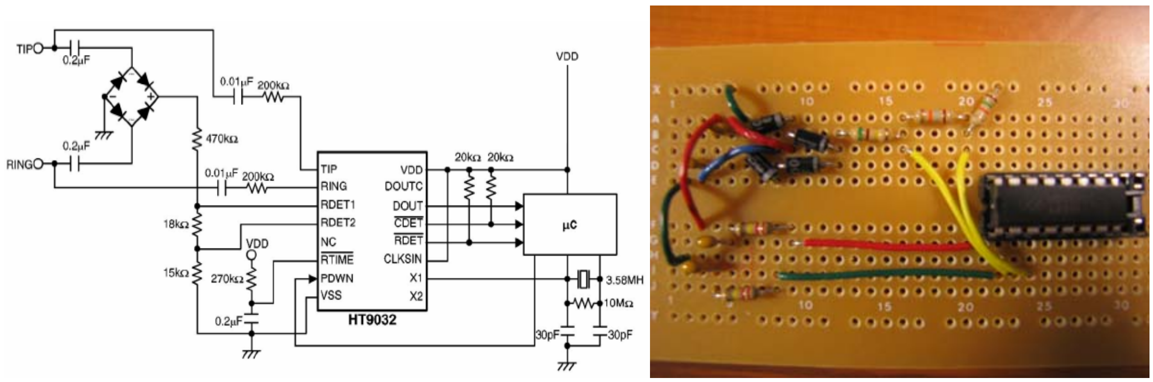


Figure 13. Caller ID Circuit

As previously mentioned, the caller ID information was simulated. However, fully simulating caller ID proved to be very difficult. Primarily because of the need to generate correct ring bursts signals as well as encode the information for transmission between bursts on a simulated telephone line. As a result, I was not able to use this circuit in its entirety. Instead, I took advantage of the output of the HT9032C and simulated the caller ID information from there. Since I knew this circuit would simply extract the caller information from the phone line and transmit it serially, I simply encoded the caller information on hyper term and transmitted the data to the mcu through the serial connection at the specified baud rate of the HT9032C IC. This is exactly the same as using this IC since the same information would be transmitted at the same rate using the same transmission medium. Testing at 1200 baud was successful, but slow so I ultimately increased the rate to 9600 baud to make testing faster. However, the system works fine at 1200 baud.

5.2.4. DTMF Decoding

To decode the DTMF signals, I was able to obtain an HT9170B IC. The HT9170B is a DTMF decoder which takes the DTMF signal as an input and outputs a 4-

bit binary representation of the key that was pressed. The following is a pin out for the HT9170B:

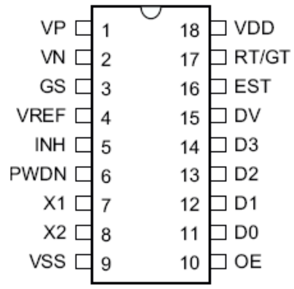


Figure 14. HT9170B Pin Out

This is a very versatile component and also offers many additional useful features. One such feature is a data valid bit, which indicates when a valid DTMF signal is being transmitted and has been successfully decoded. This was useful to distinguish between consecutive button presses (which will be discussed in the software section). Another useful feature of this component was the power down capability. Since the caller ID circuit must remain on and constantly ready to decode a caller's information, I was not able to take advantage of the caller ID's power down feature. With the DTMF decoder on the other hand, I did take advantage of the power down available on the HT9170B. Since the DTMF decoder only has to be active once a caller passes the caller ID check, I was able to place the HT9170B in power down mode until it was needed.

The DTMF decoder IC has an internal op amp which needs to be biased and fed with the input DTMF signal. It also must be in non-inverting mode. This is the resulting schematic of the circuit:

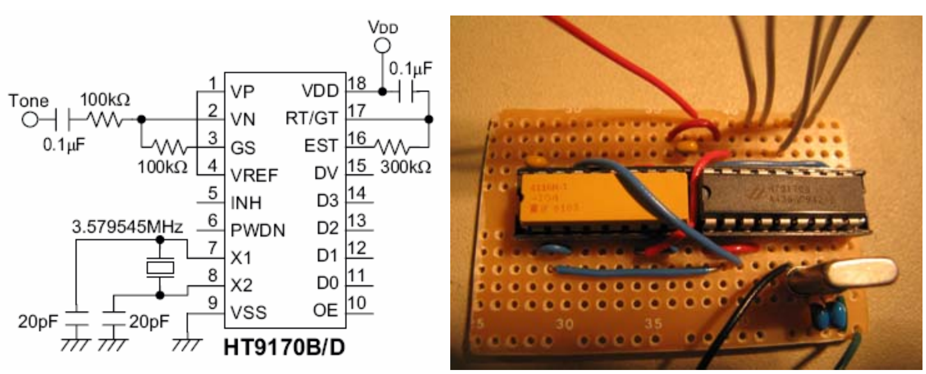


Figure 15. DTMF Decoder Circuit

I utilized a resistor pack in an effort to obtain matched resistor values for accuracy. D3…D0 represent the 4 bit binary representation of the decoded DTMF signals and DV is the data valid bit. These 5 outputs (4 bit binary + 1 bit valid) are fed to the appropriate ports of the Mega32. The representation of the standard 3 x 4 keypad is as follows:

Key	D3…D0
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
*	1010
0	1011
#	1100

Table 6. DTMF Decoder Output

5.3. Software

I utilized the Atmel Mega32 microcontroller for this design. I took advantage of the 8-bit timer 0, 3 data ports, and serial USART (universal synchronous and asynchronous serial receiver and transmitter).

5.3.1. Timing

I set up timer 0 to produce an output compare match interrupt at 1ms intervals by setting the clock prescalar to 64 and output compare register (OCR0) to 249. This interrupt was used to decrement three time variables (t1, t2, t3) which kept a reference time within the system for polling various components.

Variable	Value	Poll
t1	100 ms	Caller ID
t2	20 ms	DTMF Decoder
t3	30 ms	Pushbuttons

Table 7. Timing Values

Every iteration through the main loop, the mcu is put into the idle state until an internal interrupt occurs. This output compare match interrupt for timer 0 acts as the interrupt that wakes the mcu up. Therefore, the mcu will go idle for approximately 1 ms every cycle through the loop, then execute some code, then return to idle state. This is an effort to conserve processing power when it is not needed.

5.3.2. Serial Interface

As discussed earlier, the serial interface was used to simulate the caller ID data transmission in the simulated phone network. Since the HT9032 caller identification receiver IC serially transmits the CID data at 1200 baud, the serial interface on the Mega32 was set up to reflect this baud speed by setting the UBRRH and UBRRL registers. The formula to calculate the value for this control register is: $\frac{f_{osc}}{16 \cdot Baud} - 1$.

Plugging in the correct values (f_{osc} = 16 MHz, Baud = 1200), I obtained UBRRX = 832 (0x340). Splitting this value into the high and low control registers, I set up the USART as follows:

```
UBRRL = 0x40;    // Set baud rate to 1200 baud
UBRRH = 0x03;
```

In the end, 1200 baud proved to be a little slow for testing purposes so I ultimately set the baud rate to 9600 baud (UBRRL = 103). I thoroughly tested at both speeds however.

I set up the serial transmit and receive as interrupt driven routines in an effort to avoid blocking. Because of the nature of this project, a transmit routine was not actually necessary. In a real implementation without a simulated phone network, the serial interface would only need a receive routine to receive the caller ID information. However, in this simulated environment, I utilized the transmit routine to provide detailed feedback to the user.

5.3.3. Caller ID Decoding

The serial receive interrupt routine would constantly poll the serial USART for caller ID transmission. In the event that data is transmitted, a flag is set to signal the successful receipt of a calling party's information. The function `get_CID()` would check for the successful receipt of caller data every 100 ms. Once received, the calling party's phone number is compared, digit by digit, to the preset access list. I decided to store the access list in RAM rather than EEPROM because of the combination of two factors. First, the EEPROM has a limited write cycle lifetime and second, this is a development environment. Since I was actively designing the system, I knew the benefits of a non-volatile memory were not vital to testing. Also, since I would be writing to the EEPROM over and over during development, this would diminish its lifetime unnecessarily. In a production environment, writing this access-list to EEPROM would make sense due to its non-volatile nature.

The access list is setup such that there are 5 possible authorized callers. The caller's phone number is compared to every possibility and once an error is encountered in the current comparison, the check moves directly to the next authorized caller. This checking method is a little more efficient and saves a little time, compared to comparing through even after an error is detected. If the calling party fails authorization check, the caller is automatically disconnected (by picking up the line and hanging up). In the event that the caller is authorized, the line will go off-hook (pick up the call) and the DTMF decoder is powered up and control is handed over to the decoder.

In the event that another caller tries calling in after a caller has already been verified and the call is off-hook, a busy signal will be sent to the other caller. This is reflected in the simulated system by transmitting the appropriate feedback to the user over the serial interface.

5.3.4. DTMF Decoding

Once the caller passes the first level of security, the system would call the function `check_dtmf()` every 20 ms to poll for button presses by the user. This function would only be called once the caller is authorized, in an effort to optimize processing power.

The following is a flow diagram of the system once the caller is verified:

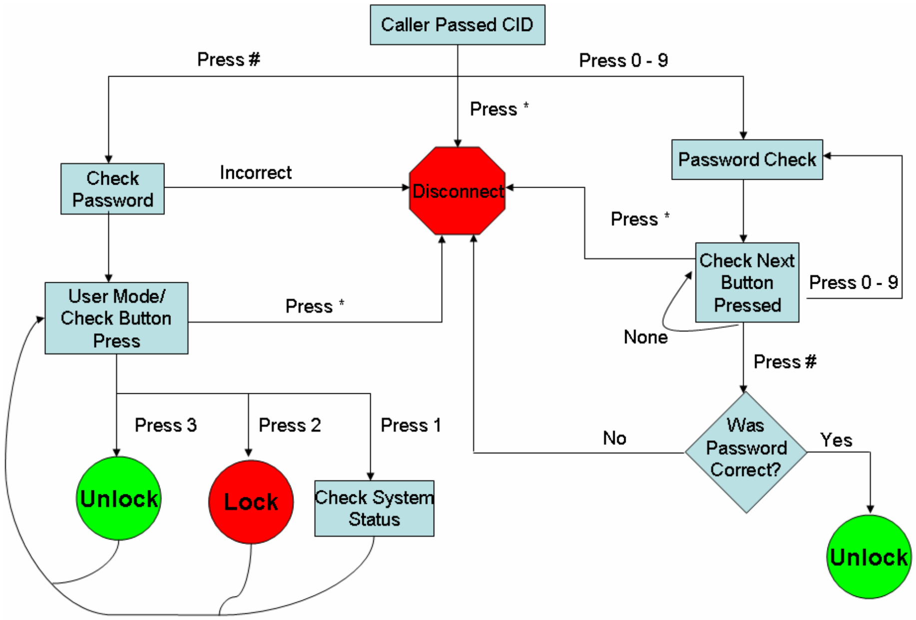


Figure 16. Flow Diagram

The output from the HT9170B DTMF decoder is used in this stage. As you can see, there are three paths that the user can take in operating the system.

1. Pressing '*' at any time automatically ends the call and disconnects the user, returning primary control to the caller ID decoder and shutting down the DTMF decoder.
2. If the user presses any digit 0 – 9, the password checker begins parsing the password and comparing it against the accepted password. The user must press '#' to terminate the password entry sequence. Once the user terminates the password entry, the system will either unlock the door if the password is correct, or disconnect the user if the password is incorrect. It is important to note that this system does not allow another password attempt. The user must call back in to attempt another password.
3. If the user presses '#' without pressing any digits, they will enter user mode, where they must first enter their password. After successfully entering the password, the user can then choose to check the status of the system, lock the door, or unlock the door.

5.3.5. Pushbuttons

Since one of the goals of this project was automation, I incorporated pushbuttons to allow the user to lock and unlock the door. Pressing the rightmost, red button locks the system and the middle, green button unlocks the system. These pushbuttons are polled every 30 ms and are independent of the rest of the system. This means that the system operates fully simultaneously. A user can be connected checking the state of the system while another user presses the lock button. The changes will be reflected to both users and is fully concurrent.

6. Packaging

When I thought of this project, I wanted to design the system such that I would be able to enclose it in a well-made package instead of on a circuit board with components everywhere. I obtained an old corded phone and took out all the parts inside. With a little help from the dremel, I was able to cut the casing perfectly to allow all of my components including LEDs, power adaptor, power switch, PCB boards, speaker, and all ICs necessary in the design.

The simulated phone system is completely enclosed within the receiver as shown below. It contains the following components:

1. 3 x 4 keypad
2. Complete DTMF generator system
3. Speaker
4. RJ-11 connector

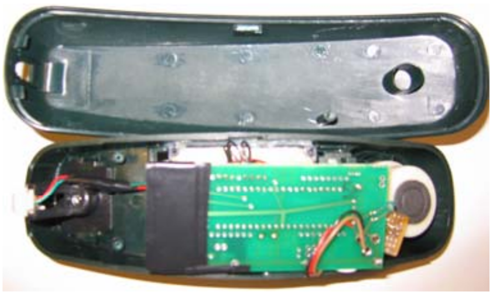


Figure 17. Packaged Simulated Phone System

The security system is completely enclosed in the accompanying base enclosure. The package contains:

1. RJ-11 connector
2. Mega32 MCU
3. DTMF decoder circuit

4. Power LED
5. System state LED (Locked/Unlocked)
6. Power adaptor
7. Power switch
8. Serial connector
9. Pushbuttons

The complete system is shown below:

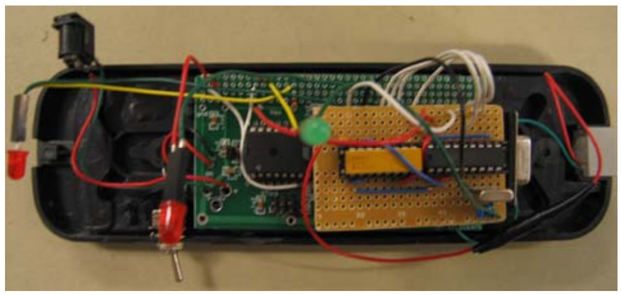


Figure 18. Enclosed System

Figure 19. The Cell Phone Security System

7. Results

I was very satisfied with the results I obtained. I set out to design a system that would decode caller ID information as well as DTMF signals from a phone line, and I accomplished just that.

Despite the fact that I was not able to test on a real phone line, I was very confident in the fact that my simulations mirrored a real phone line exactly. The DTMF signals I generated were clean and virtually free of noise. I verified this by using the oscilloscope to capture the signal generated and checking for noise spikes. Also, the DTMF decoder circuit verified that the signals were genuine DTMF signals since they were able to be decoded. To be completely satisfied with the caller ID simulation, I ran the serial communication at the rate of the HT9032 and encoded the information in the way that a real phone line would. I ultimately changed that data packet to allow for readability; however I did parse the data packet successfully at one point in the simulation.

The other aspect I was extremely pleased with was the packaging of the entire system. I initially purchased the old phones with the intention of salvaging parts within it. I ended up scrapping all of the parts inside the phone and using its casing as an enclosure for my entire system. As you can see from figure 19, the complete system is enclosed within the old phone with no loose components or wires needing connections. Even the serial connector is part of the enclosure.

8. Conclusions

My cell phone security system was successful in meeting the goals I intended for it. It allowed for a more convenient and automated form of security for a residential setting. The user mode with system feedback proved to be a very useful feature that I was very happy with.

The telephone system within the United States is a very complicated network. Before this project, I was not aware of all the restrictions and FCC regulations placed on interfacing with the phone line. They place a lot of effort into isolating their lines from any users in order to protect their infrastructure. It is an understandable measure and they go to great lengths to enforce these rules.

The principles that were utilized in this project were also very interesting. DTMF is a very clever algorithm of transmitting data across the telephone line without much signal degradation. The frequencies chosen for the DTMF system are also clever since they prevent accidental generation from human voice. Caller ID is a relatively new feature on phone systems. It was interesting to learn how well the data transmission worked with the existing infrastructure. By transmitting the data between ring bursts, no extra hardware or changes to the phone network had to be made, saving them the cost of new hardware and gaining a valuable feature.

The security system will hopefully fulfill its other primary goal which is to influence a change within the market - a change towards a more convenient, responsive, automated security system such as the cell phone security system.

9. Appendix A – Glossary of terms

On-hook – When the phone line is not in use (idle) and any attached circuitry is draining (ideally) no current.

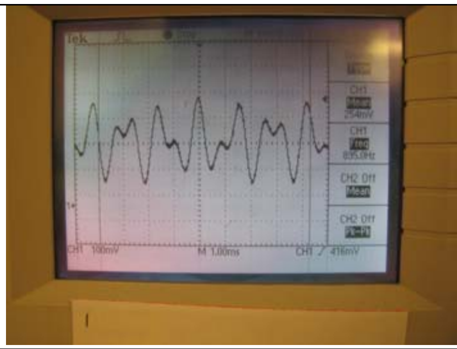
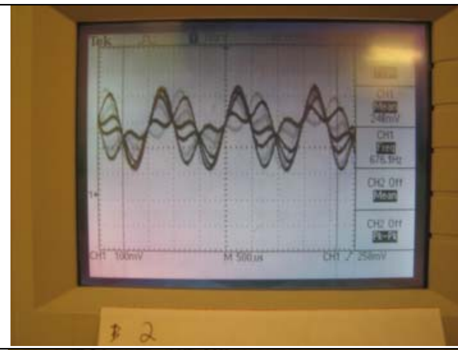
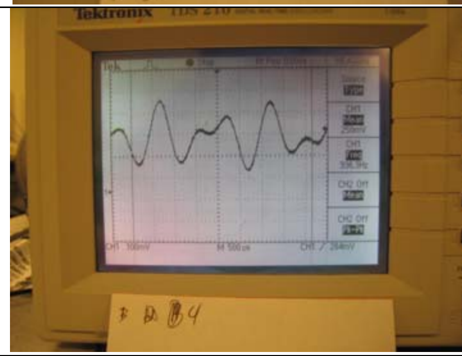
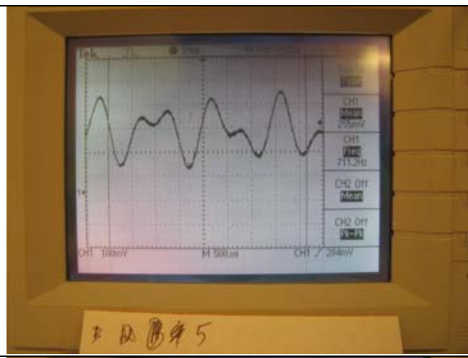
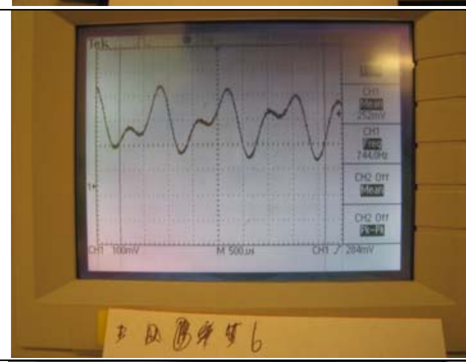
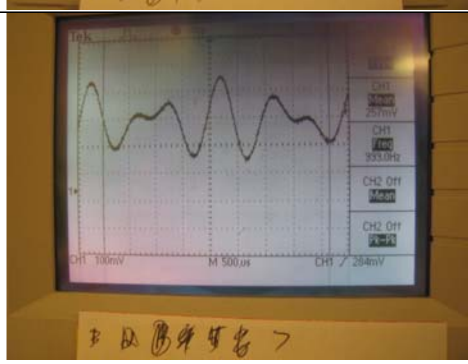
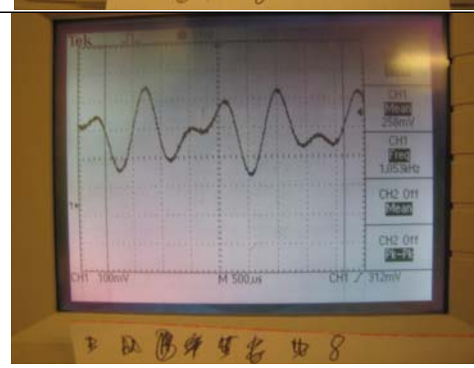
Off-hook – When the phone line is actively in use (data being transmitted) and attached devices are draining a non-zero amount of current.

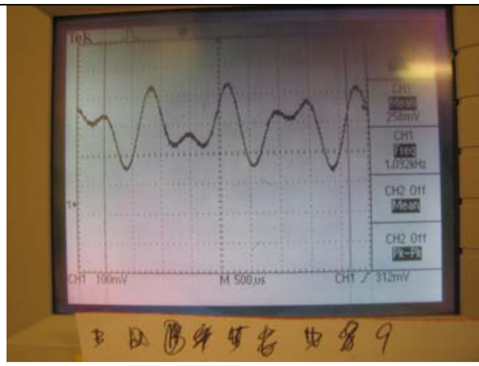
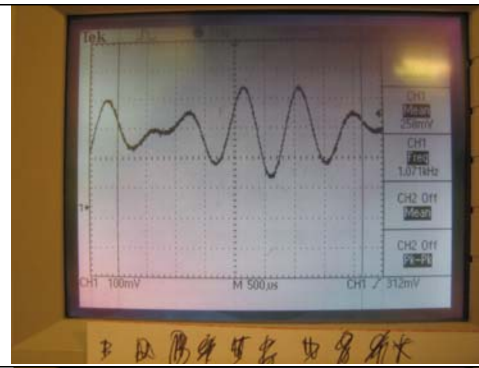
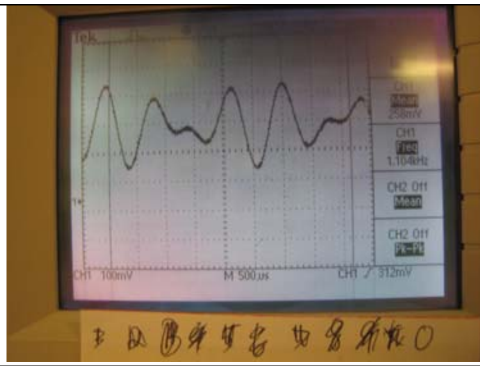
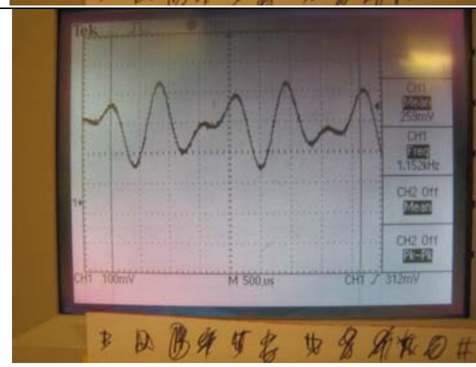
DTMF – Dual tone multi-frequency. A standard for encoding data used for transmission of messages over telephone networks.

Tip/Ring – The telephone network consists of two primary wires names “tip” and “ring.” Conventionally, the positive (+) terminal is the tip side, while the negative (-) terminal is the ring side. They are also colored green and red respectively.

FCC – Federal Communications Commission. The government agency that regulates interstate and international communications either by radio, television, wire, satellite, and cable.

10.Appendix B – Complete DTMF Signal Catalog

1, 2		
3, 4		
5, 6		
7, 8		

9, *			
0, #			

11.Appendix C – DTMF Generator Code (dtmf.c)

```
#include <Mega32.h>
#include <delay.h>
#include <math.h> // for sine

#define begin {
#define end   }
#define countMS 62  //ticks/mSec
#define maxkeys 12
//State machine state names
#define NoPush 1
#define MaybePush 2
#define Pushed 3
#define MaybeNoPush 4

unsigned long accumulator1 @0x2f0;
unsigned char highbyte1 @0x2f3;     //the HIGH byte of accumulator1
unsigned long accumulator2 @0x2f4;
unsigned char highbyte2 @0x2f7;     //the HIGH byte of accumulator2
unsigned long increment1;           //increment for acc1 & acc2
unsigned long increment2;
char sineTable[256] @0x300;         //need loc to avoid glitch
unsigned char butnum, flag;
unsigned int ref_time;              // System Reference Time
unsigned char PushFlag;             // Debounce Flag
unsigned char PushState;            // For Debounce State Machine

// Increments corresponding to the DTMF frequencies:
// 697, 770, 852, 941, 1209, 1336, 1447 Hz
flash unsigned long inc[7] = {47897475, 52913997, 58548994, 64665028,
83081847, 91809221, 99437083};

/*key pad scan table - 'key' values (3x4 keypad)
1-0xee  2-0xed  3-0xeb
4-0xde  5-0xdd  6-0xdb
7-0xbe  8-0xbd  9-0xbb
*-0x7e  0-0x7d  #-ox7b */
flash unsigned char keytbl[maxkeys]={0xee, 0xed, 0xeb, 0xde, 0xdd, 0xdb,
0xbe, 0xbd, 0xbb, 0x7e, 0x7d, 0x7b};

interrupt [TIM0_OVF] void sgen(void)
begin
      accumulator1 += increment1;
      accumulator2 += increment2;
      OCR0 = 128 + sineTable[highbyte1] + sineTable[highbyte2];
end

// Timer 1 Compare Interrupt - 1ms
interrupt [TIM1_COMPA] void timer1_comp_isr(void)
begin
      ref_time++;
end

void debounce(void)
begin
      unsigned char key;
```

```
      //get lower nibble
      DDRC = 0x0F;
      PORTC = 0xF0;
      delay_us(5);
      key = PINC;

      //get upper nibble
      DDRC = 0xF0;
      PORTC = 0x0F;
      delay_us(5);
      key = key | PINC;

      switch(PushState)
      begin
            case NoPush:
                  if (key != 0xFF) PushState = MaybePush;
                  else PushState = NoPush;
                  break;
            case MaybePush:
                  if (key != 0xFF)
                  begin
                        for (butnum = 0; butnum < maxkeys; butnum++)
                        begin
                              if (keytbl[butnum] == key)  break;
                        end
                        if (butnum == maxkeys) butnum = 0;
                        else butnum++;  //adjust by one

                        PushState = Pushed;
                        PushFlag = 1;
                  end
                  else PushState = NoPush;
                  break;
            case Pushed:
                 PORTD = ~butnum;

                 if(flag == 0)
                 begin
                    flag = 1;
                    //phase lock the sine gen and timer
                    accumulator1 = 0;
                    accumulator2 = 0;

                    // Setup increment values by row/column
                    if((butnum == 1) || (butnum == 2) || (butnum == 3))
                         increment1 = inc[0]; // 697 Hz row
                    else if((butnum == 4) || (butnum == 5) || (butnum
== 6))
                         increment1 = inc[1]; // 770 Hz row
                    else if((butnum == 7) || (butnum == 8) || (butnum
== 9))
                         increment1 = inc[2]; // 852 Hz row
                    else
                         increment1 = inc[3]; // 941 Hz row
```

```
                    if((butnum == 1) || (butnum == 4) || (butnum == 7)
|| (butnum == 10))
                         increment2 = inc[4]; // 1209 Hz column
                    else if((butnum == 2) || (butnum == 5) || (butnum
== 8) || (butnum == 11))
                         increment2 = inc[5]; // 1336 Hz column
                    else
                         increment2 = inc[6]; // 1477 Hz column

                    TCNT0 = 0;
                    OCR0 = 0;
                    // turn on fast pwm
                    // clear on compare match
                    // CLK sel - CLK (no prescalar)
                    TCCR0 = 0b01101001;
               end

               if (key != 0xFF) PushState = Pushed;
               else PushState = MaybeNoPush;
               break;
            case MaybeNoPush:
                  PORTD = 0xFF;
                  TCCR0 = 0b00000001;
                  flag = 0;
                  if (key != 0xFF) PushState = Pushed;
                  else
                  begin
                        PushState = NoPush;
                        PushFlag = 0;
                  end
                  break;
      end
end

void initialize(void)
begin
      // Setup Port(s)
      DDRD = 0xFF;
      DDRB = 0xFF;
      DDRA = 0x00;

      // Turn off ADC
      ADCSRA = 0x00;

      // Timer 0 Control Register - CLK (no prescaling)
      TCCR0 = 0b00000001;

      // Timer 1 Control Registers
      TCCR1A = 0x00;          // Normal Operation
      TCCR1B = 0b00001011;    // CLK / 64
      OCR1A = 249;            // 250 * (64/16e6) = 1 sec
      TCNT1 = 0;              // Initialize counter to 0

      // Turn on Timer0 Overflow & Timer1 Compare on Match Interrupt
      TIMSK = 0b00010001;

      // Set up initial Debounce States
```

```
	PushFlag = 0;			// no button push
	PushState = NoPush;		// init the state machine

	ref_time = 0;			// Init system reference time
end

void main(void)
begin
	unsigned int i;

	initialize();
	i = 0;
	PORTD = 0xFF;
	flag = 0;

	for (i = 0; i < 256; i++)
	begin
		sineTable[i] = (char)(63.5 * sin(6.283*((float)i)/256.0)) ;
	end

	// Enable Interrupts
	#asm("sei");

	while(1)
	begin
		PORTD = ~PINA;
		// Poll keypad every 20 ms
		if(ref_time == 20)
		begin
			debounce();
			ref_time = 0;
		end
	end
end
```

12. Appendix D – Security System Code (security.c)

```
// Cell Phone Security System
// Written by Jason Chiang - 2006
// For Masters of Engineering Design Project 2006
// Project Advisor - Bruce Land

#include <Mega32.h>
#include <stdio.h>

#define begin {
#define end   }
#define POLL_USART 100        // How often to poll the USART
#define POLL_DTMF 20          // How often to poll the DTMF
#define POLL_BUTTONS 30
// States for the system
#define UNLOCKED 0
#define LOCKED   1
// States for DTMF button push
#define NOPUSH 2
#define MAYBEPUSH 3
#define PUSHED 4
#define MAYBENOPUSH 5
// Modes of operation
#define REGULAR 6
#define SPECIAL 7

// Global Variables
unsigned char t1, t2, t3;     // Time in system (1 ms base)
unsigned char off_hook;       // Is phone off-hook?
unsigned char state;          // Keeps track of the state of the system
unsigned char p[8] = {1,2,3,4,5,6,7,8};
unsigned char p_index @0x2AD; // Index within the array for traversal
unsigned char terminator;     // # (terminator) key was pressed
unsigned char p_error;        // Error in password flag
unsigned char PushFlag;       // Debounce Flag
unsigned char PushState;      // For Debounce State Machine
unsigned char temp @0x2AB;    // Temp variable used by some methods
unsigned char mode @0x2AC;    // Regular mode or User Mode

// Access List for CID
unsigned char al[5][11];      // Stores the phone numbers of callers
with permission

// RXC ISR variables
unsigned char r_index;        // Current index in the string
unsigned char r_buffer[11];   // The input string
unsigned char r_ready;        // Flag for receive done
unsigned char r_char;         // Current character

//************************************************************
// Timer0 Compare on Match ISR
interrupt [TIM0_COMP] void timer0_overflow(void)
begin
      if(t1 > 0) t1--;
      if(t2 > 0) t2--;
      if(t3 > 0) t3--;
```

```
end

//*********************************************************
// UART character-ready ISR
interrupt [USART_RXC] void uart_rec(void)
begin
      r_char = UDR;            // get a character from the UDR register
      UDR = r_char;            // Echo it back to hyperterm

      // Build the input string, if it's not carriage return
      if (r_char != '\r')      r_buffer[r_index++] = r_char;
      else
      begin
            putchar('\n');     // use putchar to avoid overwrite
            r_buffer[r_index] = 0x00;     // zero terminate
            r_ready = 1;                  // signal cmd processor
            UCSRB.7 = 0;                  // stop rec ISR
      end
end

void initialize(void)
begin
      //--------------Setup Ports------------------------
      DDRA = 0x00;             // Pushbuttons for manual Lock/Unlock
      DDRB = 0xFF;             // PORTB Output to LEDs for status
      //DDRC = 0x60;           // PORTC I/O from DTMF decoder
      DDRC = 0b01110000;

      //--------------Setup USART Registers---------------
      UCSRB = 0x18;            // Enable TX & RX ISRs
      UBRRL = 103;             // Set baud rate to 9600 baud
      /*UBRRL = 0x40;          // Set baud rate to 1200 baud
      UBRRH = 0x03;*/

      //--------------Setup Timer Registers---------------
      TCCR0 = 0b00001011;      // CLK/64 & Clr-On-Match
      OCR0 = 249;              // (64/CLK) * 250 = 1 mSec
      TIMSK = 2;               // turn on timer 0 cmp-match ISR

      //--------------Set some initial values-------------
      t1 = POLL_USART;         // Poll the USART every 100 ms
      t2 = POLL_DTMF;          // Set t1 to POLL_DTMF for 20 ms poll
      t3 = POLL_BUTTONS;       // Poll the pushbuttons every 30 ms
      off_hook = 0;            // Phone line is NOT off hook initially
      state = LOCKED;          // Initially locked
      p_index = 0;             // Index within password array
      terminator = 0;          // Terminator was not pushed initially
      p_error = 0;             // No error in password entry, yet
      PushFlag = 0;            // No button push
      PushState = NOPUSH;      // No button push state
      r_ready = 0;             // Recieve not complete
      mode = REGULAR;    // Start in regular mode - Cannot query system
      temp = 0;                // Initial temp value

      PORTB = 0xFF;            // Initially
      PORTC.5 = 1;             // Powerdown DTMF decoder initially
```

```
      // Init some random callers
      sprintf(al[0],"5163192579");
      sprintf(al[1],"6071234567");
      sprintf(al[2],"5550000000");
      sprintf(al[3],"0000000000");
      sprintf(al[4],"0000000000");

      #asm("sei");            // Fire up those interrupts
end

// Sets the RCX interrupt
void pollUSART(void)
begin
      r_ready = 0;            // Receive is not yet complete
      r_index = 0;            // Set index of received string to 0
      UCSRB.7 = 1;            // Enable the RX Complete Interrupt
end

// Received the CID packet information, decodes it, and verifies user
void get_CID(void)
begin
      unsigned char i, j;
      unsigned char row_error, error;

      row_error = 0;          // No errors in the row, yet
      error = 0;        // No errors verifying the user, yet
      t1 = POLL_USART;  // Reset function timer

      if((r_ready == 1) && (off_hook == 1))
      begin
            putsf("\r<!>Busy Signal<!>\r\n");
            pollUSART();
      end
      else if(r_ready == 1)
      begin
            putsf("\r<SYSTEM> Call Acknowledged, Checking ID...\r");
            // Traverse access list and verify user
            for(i = 0; i < 5; i++)  // Up to 5 entries in access list
            begin
                  // Traverse each digit of phone number or row i
                  for(j = 0; j < 11; j++)
                  begin
                        // Once it encounters a mismatch, break
                        if(r_buffer[j] != al[i][j])
                        begin
                              row_error = 1;
                              break;
                        end
                  end
                  // No error in row i - verified! So break
                  if(row_error == 0)
                  begin
                        error = 0;
                        break;
                  end
                  else if((row_error == 1) && (i == 4)) error = 1;
```

```
			row_error = 0;		// Reset the row
		end

		if(error == 1)
		begin
			putsf("\r<SYSTEM> UNAUTHORIZED USER - Caller
Disconnected\r\n\n");
			PORTC.5 = 1;
		end
		else
		begin
			printf("\r<SYSTEM> User Authenticated -
 %s\r\n",al[i]);
			printf("\r<SYSTEM> Turn on DTMF Decoder & wait for
password\r\n\n");
			off_hook = 1;
			PORTC.5 = 0;
		end
		pollUSART();
	end
end

// Sets the state of the security system based on paramter passed in
void setState(char s)
begin
	if(s == LOCKED)
	begin
		state = LOCKED;
		PORTB = 0xFD;
	end
	else
	begin
		state = UNLOCKED;
		PORTB = 0x7F;
	end
end

// Helper method returns the state of the system - Locked/Unlocked
char getState(void)
begin
	return(state);
end

// Checks the DTMF decoder every 'POLL_DTMF' ms
// PINC0..3 = Binary value of digit
// PINC.7 = Valid bit
void check_dtmf(void)
begin
	unsigned char key;

	key = PINC & 0x0F;  // Get lower byte (digit that was pushed)

	switch(PushState)
	begin
		case NOPUSH:
			if(PINC.7 == 1) PushState = MAYBEPUSH;
			else PushState = NOPUSH;
```

```
                  break;
            case MAYBEPUSH:
                  if((key != 0x00) && (PINC.7 == 1))  // ... AND valid
                  begin
                        // Increment index in pass array
                        if(PushFlag == 0) p_index++;

      // The following is to distinguish between regular and user mode
      // In user mode, the user and query the state of the system
      // User enters User Mode by just pressing # at the password entry
                        if((p_index == 1) && (key == 12))
                        begin
                              terminator = 1;
                              mode = SPECIAL;
                        end
                        else if(key == 12)
                        begin
                              if((p_index == 9) && (p_error == 0))
putsf("<SYSTEM> Password Verified\r");
                              else putsf("<SYSTEM> Password Incorrect -
Caller Disconnect\r\n");
                              terminator = 1;
                        end

                        PushState = PUSHED;
                        PushFlag = 1;
                  end
                  else PushState = NOPUSH;
                  break;
            case PUSHED:
                  if((key == p[(p_index - 1)]) && (key != 12))
                        p_error = 0;
                  else if(key != 12) p_error = 1;

                  if((key != 0x00) && (PINC.7 == 1))
                        PushState = PUSHED;
                  else PushState = MAYBENOPUSH;
                  break;
            case MAYBENOPUSH:
                  if((key != 0x00) && (PINC.7 == 1))PushState = PUSHED;
                  else
                  begin
                        PushState = NOPUSH;
                        PushFlag = 0;
                  end
                  break;
      end
end

void getKeyPress(void)
begin
      unsigned char key;

      key = PINC & 0x0F;  // Get lower byte (digit that was pushed)

      switch(PushState)
      begin
```

```
            case NOPUSH:
                  if(PINC.7 == 1) PushState = MAYBEPUSH;
                  else PushState = NOPUSH;
                  break;
            case MAYBEPUSH:
                  if((key != 0x00) && (PINC.7 == 1))  // ... AND valid
                  begin
                        PushState = PUSHED;

                        if(key == 0x01)
                        begin
                              if(getState() == LOCKED) putsf("<SYSTEM>
System is LOCKED\r");
                              if(getState() == UNLOCKED)
putsf("<SYSTEM> System is UNLOCKED\r");

                        end
                        else if(key == 0x02) setState(LOCKED);

                        else if(key == 0x03) setState(UNLOCKED);
                        else if(key == 0x0B)    // Disconnect
                        begin
                              putsf("<SYSTEM> User Disconnected\r\n");
                              p_index = 0;      // Reset password index
                              p_error = 0;// Reset password error flag
                              terminator = 0;// Reset terminator flag
                              mode = REGULAR;// Back to regular mode
                              off_hook = 0;// End the call - hookswitch
control
                              temp = 0;   // Reset the temp

                        end
                        else  putsf("<SYSTEM> Unknown Command\r\n");
                  end
                  else PushState = NOPUSH;
                  break;
            case PUSHED:
                  if((key != 0x00) && (PINC.7 == 1)) PushState =
PUSHED;
                  else PushState = MAYBENOPUSH;
                  break;
            case MAYBENOPUSH:
                  if((key != 0x00) && (PINC.7 == 1)) PushState =
PUSHED;
                  else PushState = NOPUSH;
                  break;
      end // end switch
end

void main(void)
begin
      initialize();           // Initialize some stuff

      // Initial "splash-screen"
      putsf("\r\n __________________________________\r");
      putsf("| Welcome to Mobile Security v1.0 |\r");
      putsf("|    by Jason Chiang - 2006       |\r");
```

```
	putsf(" --------------------------------\r\n");

	pollUSART();      // Start polling USART
	setState(LOCKED); // System starts out locked

	while(1)
	begin
		MCUCR = 0b10000000;     // Enable sleep mode
		#asm("sleep");    // Sleep (idle) - Wakes up every 100 ms
		MCUCR = 0b00000000;     // Disable sleep mode

		// Poll CID every POLL_USART ms (100 ms)
		if(t1 == 0) get_CID();  // Poll for the caller ID

		// Poll DTMF Decoder every POLL_DTMF ms (20 ms) only if
off_hook
		if((t2 == 0) && (off_hook == 1))
		begin
			// Caller passed Caller ID check, phone is now off-
hook & active
			t2 = POLL_DTMF;
			if(mode == REGULAR)
			begin
			if((terminator == 0)) check_dtmf();// Not yet hit '#'
			else if(terminator == 1)
			begin
				// In regular mode, the user successfully
unlocked the door
				if((p_error == 0) && (p_index == 9) && (mode ==
REGULAR))
				begin
						setState(UNLOCKED);
				end

				p_index = 0;      // Reset password index
				p_error = 0;      // Reset password error flag
				terminator = 0;   // Reset terminator flag
				mode = REGULAR;   // Back to regular mode
				off_hook = 0;           // End the call -
hookswitch control
				temp = 0;
			end //end else if(terminator == 1)
		end
		else if(mode == SPECIAL)
		begin
			if(temp == 0)
			begin
				putsf("\r\n<SYSTEM> Special User Mode\r");
				putsf("<SYSTEM> Waiting for password...\r");

				temp = 1;   // Disable printing this again
				terminator = 0;// Reset terminator for password
				p_index = 0;// Reset the password index to 0
				p_error = 0;// Reset errors in password entry
			end
			else
			begin
```

```
                        if((terminator == 0)) check_dtmf(); // Not yet
hit '#'
                        else
                        begin
                              // In special mode, the user successfully
entered the password
                              // Now prompt them for what to query/do
                              if((p_error == 0) && (p_index == 9))
                              begin
                                    getKeyPress();
                              end // end if((p_error == 0) && (p_index
== 9))
                              else
                              begin
                                    p_index = 0;// Reset password index
                                    p_error = 0;// Reset password error
flag
                                    terminator = 0;// Reset terminator
                                    mode = REGULAR;// Back to regular
                                    off_hook = 0;// End the call -
hookswitch control
                                    temp = 0;   // Reset the display
                              end
                        end // end else
                  end // end !if(temp == 0)
            end
            end // end if(t2 == 0)

            // Poll the manual override buttons every POLL_BUTTONS ms
(30 ms)
            if(t3 == 0)
            begin
                  t3 = POLL_BUTTONS;
                  if(~PINA.0 == 1) setState(LOCKED);
                  if(~PINA.1 == 1) setState(UNLOCKED);
            end
      end
end
```