

3)

# A Digital Answering Machine Using Analog Caller ID

by

Fred D. Quintana

Submitted to the Department of Electrical Engineering and  
Computer Science

in partial fulfillment of the requirements for the degree of

Bachelor of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1993

© Fred D. Quintana, MCMXCIII. All rights reserved.

The author hereby grants to MIT permission to reproduce and to  
distribute copies  
of this thesis document in whole or in part, and to grant others the  
right to do so.

Author .....  
Department of Electrical Engineering and Computer Science  
May 17, 1993

Certified by .....  
Gregory M. Papadopoulos  
Assistant Professor  
Thesis Supervisor

Accepted by .....  
Leonard A. Gould  
Chairman, Departmental Committee on Undergraduate Theses

MASSACHUSETTS INSTITUTE  
OF TECHNOLOGY

ARCHIVES

JUL 26 1993

# **A Digital Answering Machine Using Analog Caller ID**

by

Fred D. Quintana

Submitted to the Department of Electrical Engineering and Computer Science  
on May 17, 1993, in partial fulfillment of the  
requirements for the degree of  
Bachelor of Science in Computer Science and Engineering

## **Abstract**

The purpose of this thesis is to build an answering machine that combines the speed and convenience of a digital answering machine with the advanced features of Caller ID to provide the telephone user with a level of control never before found in the home. On phone lines equipped with the Caller ID service, the phone number of the caller is encoded and sent out by the phone company. This information is sent between the first and second ring and can be retrieved to provide the callee with the caller's phone number. The device described in this thesis contains an internal database of phone number/name pairs which is scanned for a match as soon as the information becomes available. If the number is found in the database, the corresponding name is displayed on an LCD display; otherwise, the number is displayed. This feature saves the user from having to remember the numbers of frequently calling parties. This database can be updated using the telephone keypad. A log that contains all incoming and outgoing calls is also maintained. As a result, the user will know who called throughout the day, even if the caller does not leave a message on the answering machine.

The other component of this device is the integration of a digital answering machine with the Caller ID decoder. This makes possible specialized outgoing messages, depending on who is calling. For example, you can assign one greeting to your boss, another to your parents, and yet another to your friend. This combination of a digital answering machine and Caller ID makes possible new and discrete ways of screening calls. There is no longer any need to wait for the answering machine to pick up in order to determine who is calling. When the answering machine answers, the appropriate greeting will be sent. Since the device knows who is calling, it can also know how to act on that call.

Thesis Supervisor: Gregory M. Papadopoulos  
Title: Assistant Professor

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
1.1	Motivations . . . . .	6
1.2	Goal . . . . .	7
1.3	Solution . . . . .	8
1.3.1	The Motorola 68HC16 Evaluation Board . . . . .	8
1.3.2	The Motorola Caller ID Receiver . . . . .	9
1.3.3	The AT&T Telephone Answering Device (TAD) Chipset . . . . .	10
<b>2</b>	<b>Development Environment</b>	<b>13</b>
<b>3</b>	<b>Functional Specification</b>	<b>15</b>
3.1	Full Specification . . . . .	15
3.1.1	Mode 1: Wait for Call . . . . .	17
3.1.2	Mode 2: Display Incoming Number . . . . .	17
3.1.3	Mode 3: Display Outgoing Number . . . . .	18
3.1.4	Mode 4: Browse Log . . . . .	18
3.1.5	Mode 5: Play Announcement/Record Message . . . . .	19
3.1.6	Mode 6: Set Time . . . . .	19
3.1.7	Mode 7: Modify Announcements . . . . .	20
3.1.8	Mode 8: Playback Messages . . . . .	21
3.1.9	Mode 9: Administer Announcement/Number Database . . . . .	22
3.1.10	Mode 10: Administer Name/Number Database . . . . .	22
3.1.11	Mode 11: Number Entry Mode . . . . .	23

3.1.12	Mode 12: Name Entry Mode . . . . .	24
3.2	Current Implementation . . . . .	24
<b>4</b>	<b>Hardware Specification</b>	<b>26</b>
4.1	The 68HC16 Evaluation Board . . . . .	26
4.2	The Motorola Caller ID Receiver . . . . .	27
4.3	The DTMF Touch-Tone Decoder . . . . .	27
4.4	The LCD Display . . . . .	28
4.5	The Analog Phone-line Interface . . . . .	29
<b>5</b>	<b>Software Specification</b>	<b>30</b>
5.1	GPIP pins . . . . .	30
5.2	The DTMF Receiver . . . . .	30
5.3	LCD Display . . . . .	31
5.4	CID Receiver . . . . .	32
5.5	Touch-Tone Data Input . . . . .	33
5.6	Clock Maintenance . . . . .	33
5.7	Main Dispatch Routine . . . . .	34
5.8	Utility routines . . . . .	34
<b>6</b>	<b>Conclusion</b>	<b>35</b>
6.1	Current Implementation . . . . .	36
6.2	Future Directions . . . . .	36
<b>A</b>	<b>Software</b>	<b>38</b>
<b>B</b>	<b>Schematics</b>	<b>88</b>

# List of Figures

1-1	Typical Home Configuration . . . . .	8
1-2	High Level View of Hardware . . . . .	9
1-3	CID Single Message Format . . . . .	11
1-4	CID Multiple Message Format . . . . .	12
3-1	Faceplate of Product . . . . .	16
3-2	Display for Mode 1 . . . . .	17
3-3	Display for Mode 2 . . . . .	18
3-4	Display for Mode 3 . . . . .	18
3-5	Display for Mode 4 . . . . .	18
3-6	Display for Mode 5 . . . . .	19
3-7	Display for Mode 6 . . . . .	19
3-8	Display for Mode 7 . . . . .	20
3-9	Display for Mode 8 . . . . .	21
3-10	Display for Mode 9 . . . . .	22
3-11	Display for Mode 10 . . . . .	23
3-12	Display for Mode 11 . . . . .	23
3-13	Display for Mode 12 . . . . .	24
4-1	DTMF Keypad Layout . . . . .	28
B-1	Schematic of Hardware . . . . .	89

# Chapter 1

## Introduction

### 1.1 Motivations

As the national phone network becomes more advanced, many new features are becoming available to the customer, including the ability to determine who is calling and to act on that information in a variety of ways. This information is obtained by way of a service called Caller ID (see [3]). Caller ID is a way for the phone company to send information about who is calling, over the same analog phone lines that are currently used for telephone calls. This information is sent out half a second after the first ring, so the caller can be identified before the second ring even begins. With this information in hand, a person has much more control over how incoming calls are handled than ever before. Presently, the Caller ID service is available in roughly 30 of the 50 states, and before long it will be offered everywhere. In addition to subscribing to this service, the customer must also obtain a device that decodes this information and displays it in a useful fashion. This thesis will address this need.

As the need for smaller and smaller computers grows, the chips that go into these smaller computers are also becoming smaller, more powerful, and more inexpensive. As a result, many of today's products are based on these powerful, inexpensive digital circuits. However, the answering machine market is still slow to accept this solution, and most of them have remained analog. That is, they still require a cassette tape to record the incoming and outgoing messages. Besides the fact that moving parts

wear out much faster than solid-state components, the tape results in slow access time to the messages and doesn't allow for selective deleting of messages. This lack of selective deletion can result in saving seven unwanted messages just because the eighth message is important, for example. It would be much more convenient if all the messages were stored in digital memory, eliminating the need for tapes and making message maintenance much easier.

## 1.2 Goal

The purpose of this thesis is to combine the flexibility given by Caller ID with the convenience of a digital answering machine to deliver capabilities not available anywhere else today. It will consist of a device that goes between the phone and the wall phone jack, in the same way that a typical answering machine is connected (see Figure 1-1). When a call comes in and the device detects valid Caller ID information on the line, it decodes the data and logs the call. It then consults a database of name and number associations and displays the name and number on the LCD display. If the number is not in the database, only the number is displayed. This same procedure is followed for outgoing calls. As a result, the last 30 incoming and outgoing calls are logged, allowing the user to browse them at any time. Even if the caller doesn't leave a message, their call is logged as long as they let the phone ring at least once.

The device will also maintain a database of number and outgoing message associations, so that different callers can receive different greeting messages from the answering machine. So, for example, your boss can receive one message, your friends another, and everyone else yet another. This gives the user unprecedented control over how their calls are handled. This is only possible using a digital answering machine, since winding a tape to the desired greeting would be both slow and unreliable. And since the messages are stored in computer memory, message retrieval time is instantaneous and it is possible to selectively delete any of the messages.

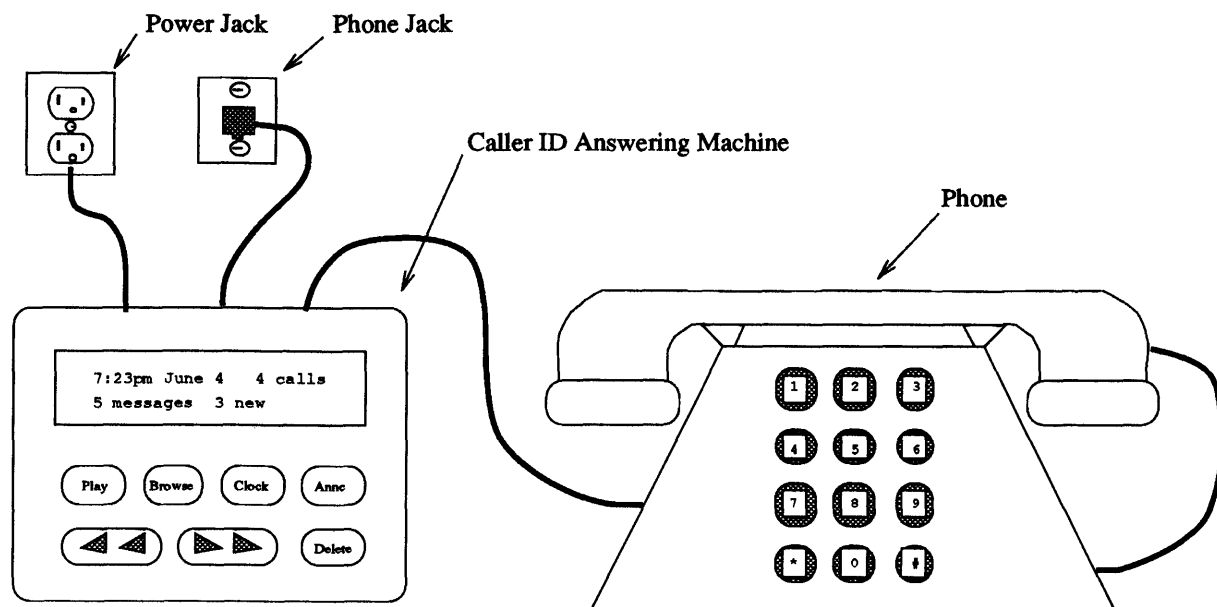


Figure 1-1: Typical Home Configuration

## 1.3 Solution

This prototype is composed of three major components, the Motorola 68HC16 evaluation board, the Motorola Caller ID receiver, and the AT&T Telephone Answering Device (TAD) chipset. Refer to Figure 1-2 for a high level view of how these pieces fit together.

### 1.3.1 The Motorola 68HC16 Evaluation Board

The 68HC16 (see [4]) is a 16-bit microcontroller that runs at 16 MHz. It is an inexpensive microcontroller and was chosen for this project mainly because the evaluation board allows development on any PC-compatible, eliminating the need for expensive microprocessor emulators and dedicated development environments. The 68HC16 contains a serial input/output module as well as several timers and chip-selects. The chip-selects are useful in that they allow the mapping of external devices onto the 68HC16 memory map without requiring external components to select the devices. The integration of all these modules makes the 68HC16 microprocessor well suited



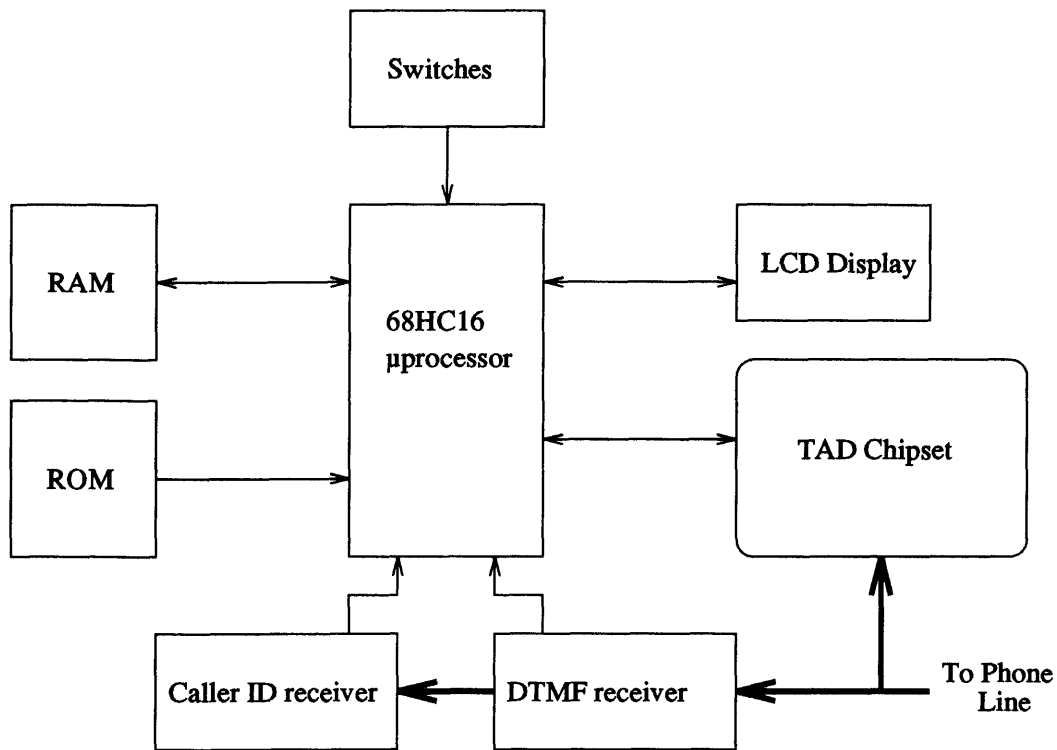


Figure 1-2: High Level View of Hardware

for a stand-alone application such as this one.

### 1.3.2 The Motorola Caller ID Receiver

As mentioned in section 1.1, analog Caller ID (see [3]) is a way for the phone company to send information about who is calling, over the same analog lines that are currently used for telephone calls. The information is frequency modulated using a technique known as Frequency-Shift-Keying<sup>1</sup> (FSK) encoding, then sent between the first and second rings. So, on the customer's end, there needs to be a device that can listen during these two rings and decode the information into serial data. That is what the Motorola Caller ID chip (the MC145447) does. There are two formats for the Caller ID messages, single message and multiple message (see Figures 1-3 and 1-4). The single message format includes only the phone number, date, and time. The multiple

<sup>1</sup>FSK is how most modems send data over the phone lines. They take a stream of ones and zeros, and encode a one as a certain frequency, and a zero as another frequency.

message format can include various strings as well as the number, date and time. Once the Caller ID chip recognizes actual data on the line, it demodulates it and sends it out as serial data. It also has two output pins that can be checked to see if the phone is ringing, and whether or not the chip detects a valid carrier on the line.

### **1.3.3 The AT&T Telephone Answering Device (TAD) Chipset**

The TAD chipset is a collection of three AT&T chips that together provide the functionality for an all-digital answering system. It offers high compression, storing up to 20 minutes of speech in 1 Mbyte of RAM, with up to 80 minutes of total storage. It does all the message management itself, while its microprocessor interface gives the microprocessor full control of all the functions of the chipset (see [5]).

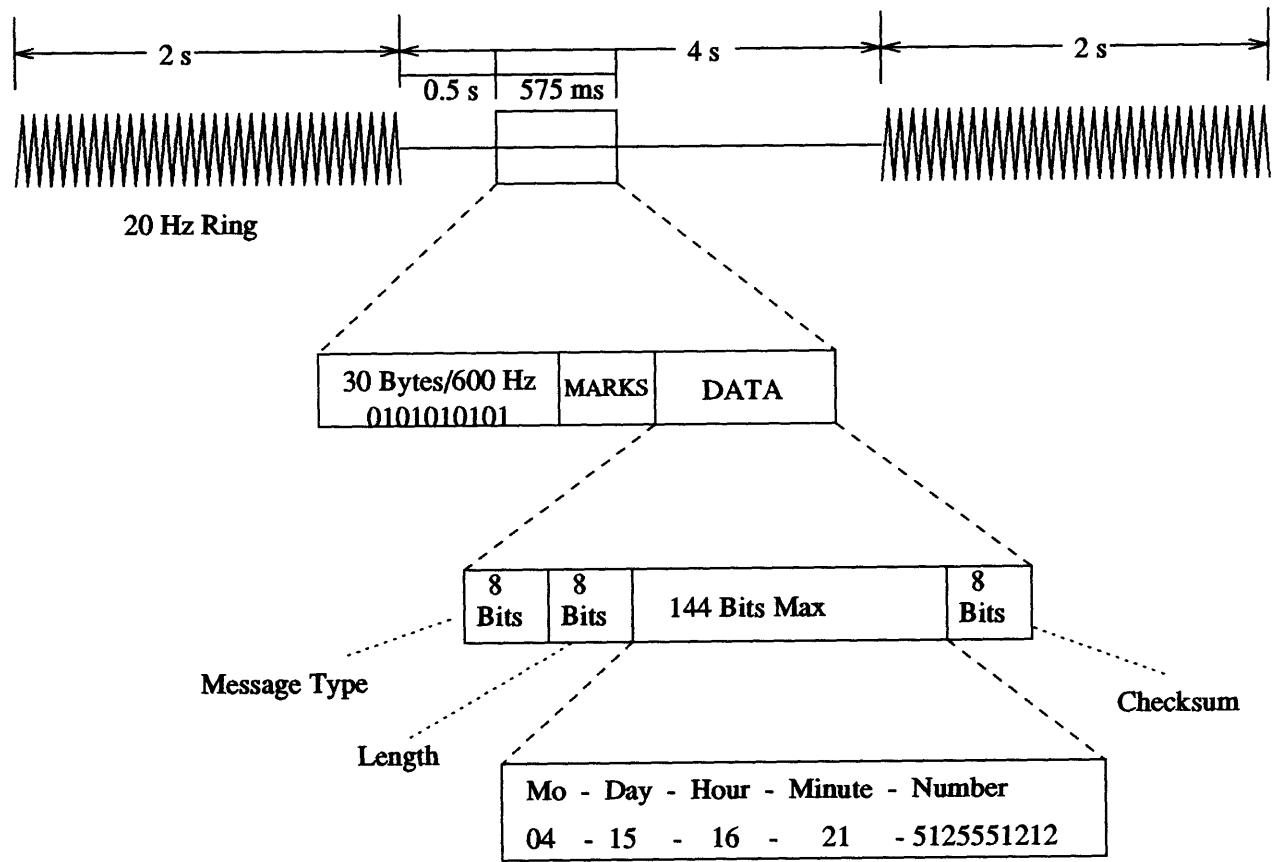
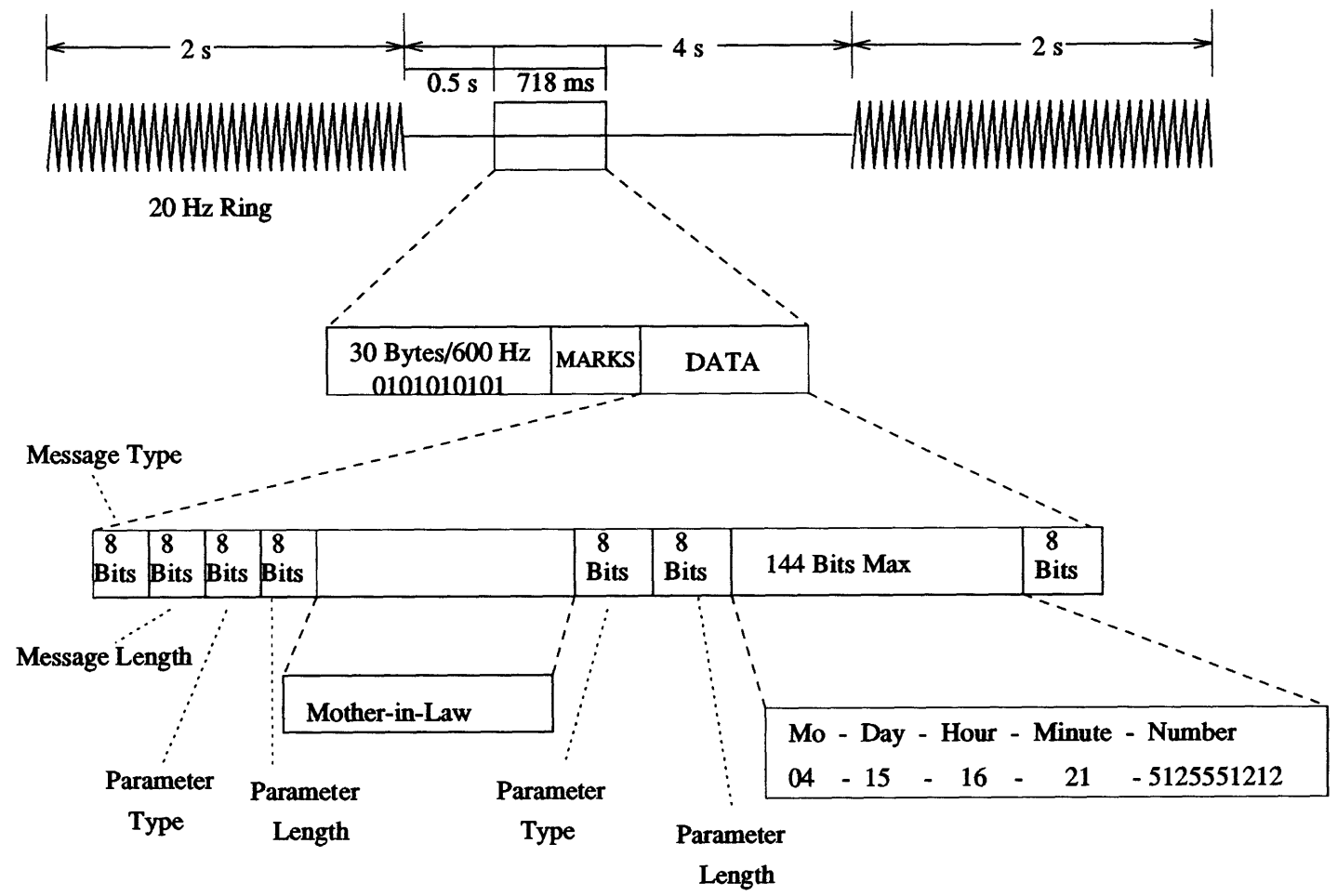


Figure 1-3: CID Single Message Format

Figure 1-4: CID Multiple Message Format



## Chapter 2

# Development Environment

The hardware for this project was developed using the 68HC16 evaluation board (see [4]). This board is a printed-circuit board that contains the 68HC16, sockets for RAM and ROM, a wire-wrap area, a parallel port, and a serial port. The serial port is connected to the serial input/output pins of the 68HC16 whereas; the parallel port is used to download programs into the “emulation ROM” on the board. This emulation ROM is actually RAM that sits in the ROM sockets, for development purposes. A program that runs on a PC takes in Motorola S-19 hex-format code and downloads it through the parallel port into this RAM. After the code is finalized it can be burnt into ROMs, which then replace the emulation ROMs.

The software was developed on an IBM-compatible PC and was written in C. The compiler (see [6]) was supplied, free of charge, by Eric Schneider of Eris Systems in Minnesota. This compiler supports C code, assembly, and Isil, a language designed specifically for developing embedded-processor applications. This compiler can generate code for various microprocessors from several different manufacturers. Since the software was written entirely in C, it would not be a difficult matter to port the code to another processor (the 68HC11, for example) in the future. This compiler can generate the S-19 hex-format code required by the 68HC16 download program.

One problem in testing the Caller ID portion of the project is that the local telephone company in Boston does not currently provide the Caller ID service. As a result it was necessary to obtain a Caller ID line simulator, which was purchased

from Rochelle Communications in Texas. This simulator is an 8-bit board that fits inside an IBM-compatible computer. It has a jack into which a phone can be plugged, and sends the caller ID information out to the phone in the same way that the phone company sends it. The Caller ID simulator can simulate both the single and multiple message types, allowing for full and convenient testing of the Caller ID receiver. Since the Caller ID simulator board software needed to be running at the same time as the 68HC16 download program, a second PC was needed to run the board (see [1]).

# Chapter 3

## Functional Specification

This chapter contains the high-level functional specification of the product. In other words, it describes what the user sees, what modes are available, and what the various buttons do. The first part of the specification will cover the long-term goals for the product, while the latter part will describe any differences found between the current prototype and the final product.

### 3.1 Full Specification

This section uses a finite state machine model to characterize all the different possible modes in which the device can be, and to describe what events cause different actions to be taken. When reading through these modes, please refer to Figure 3-1 to see a layout of the faceplate and to see what buttons are available. The different modes of operation are as follows:

1. **Wait for call:** This is the default mode. It displays the number of calls for the day and the number of messages.
2. **Display Incoming Number:** This mode displays the incoming phone number and corresponding name, if available.
3. **Display Outgoing Number:** This mode displays the outgoing phone number and corresponding name, if available.

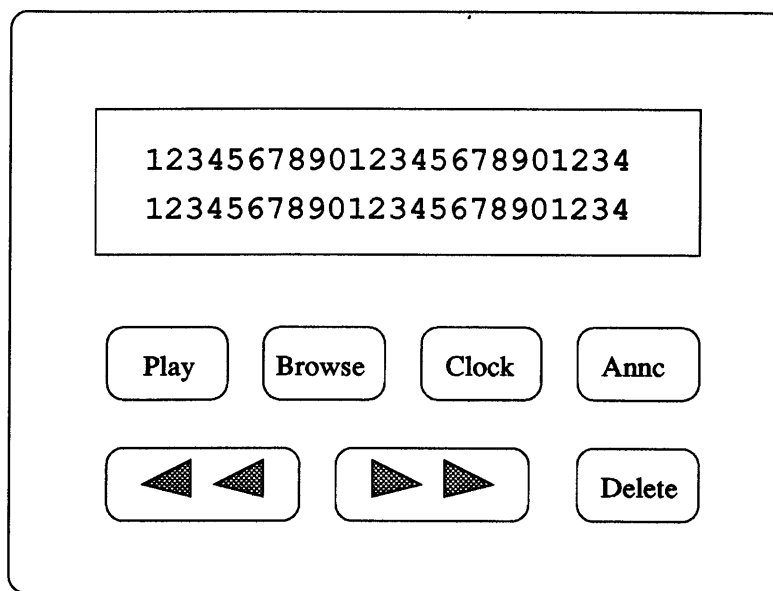


Figure 3-1: Faceplate of Product

4. **Browse Log:** This mode allows the user to browse the log of the last 30 incoming and outgoing calls.
5. **Play Announcement/Record Message:** This is the answering machine mode. Control comes here after three rings and the caller's message is recorded.
6. **Set Time:** This mode allows the user to set the time of the internal clock.
7. **Modify Announcements:** This mode is for announcement maintenance.
8. **Play Messages:** This mode is for playing back the recorded messages.
9. **Administer Name/Number Database:** This mode is for name/number database management.
10. **Administer Announcement/Number Database:** This mode is for announcement/number database management.



```
7:23pm June 4    4 calls
5 messages  3 new
```

Figure 3-2: Display for Mode 1

11. Number Entry Mode: This is a generic mode to enter a number using the phone keypad.
12. Name Entry Mode: This a generic mode to enter a string using the phone keypad.

### 3.1.1 Mode 1: Wait for Call

This is the default mode and the mode that comes up on power-up. In this mode, the device waits for incoming and outgoing calls. Its behavior is as follows:

- PLAY pressed: Go to mode 8.
- BROWSE pressed: Go to mode 4.
- CLOCK pressed: Go to mode 6.
- ANNC pressed: Go to mode 7.
- Incoming Call: Go to mode 2.
- Outgoing Call: Go to mode 3.
- LEFT arrow pressed: Go to mode 9.
- RIGHT arrow pressed: Go to mode 10.

### 3.1.2 Mode 2: Display Incoming Number

This mode displays the incoming number, time, and date. The name is also displayed if the number is in the name/number database. No buttons are active in

```
George Washington    IN
617-225-7366 15:13 11/23
```

Figure 3-3: Display for Mode 2

```
George Washington    OUT
617-225-7366 15:13 11/23
```

Figure 3-4: Display for Mode 3

this mode. As soon as the phone is hung up it returns to mode 1. As soon as the fourth ring starts it goes to mode 5.

### 3.1.3 Mode 3: Display Outgoing Number

This mode displays the outgoing number, time and date. The name is also displayed if the number is in the name/number database. No buttons are active in this mode. As soon as the phone is hung up it returns to mode 1.

### 3.1.4 Mode 4: Browse Log

This mode allows the user to browse the call log. As soon as this mode is entered, the last call is displayed. If any incoming calls or outgoing calls are made while in browse mode, they are ignored. The behavior is as follows:

- BROWSE pressed: Go to mode 1.

```
George Washington    OUT
617-225-7366 15:13 11/23
```

Figure 3-5: Display for Mode 4

```
George Washington      #5
617-225-7366 15:13 11/23
```

Figure 3-6: Display for Mode 5

```
Set Time:      15:13 11/23
CLOCK:save    DELETE:abort
```

Figure 3-7: Display for Mode 6

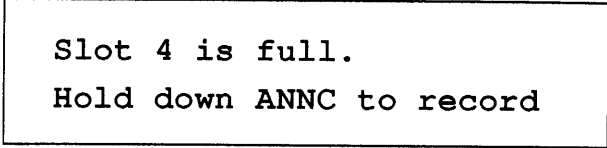
- LEFT arrow pressed: Go back in log.
- RIGHT arrow pressed: Go forward in log.
- All other events ignored.

### 3.1.5 Mode 5: Play Announcement/Record Message

This mode plays the announcement associated with the incoming call. If no announcement is associated with the number in the number/announcement database, then announcement 0, the default announcement, is played. If the phone is hung up or answered before the beep, the device goes to mode 3 without recording a message. Otherwise, the device will record a message and go to mode 3 as soon as the phone is hung up or answered, whichever comes first.

### 3.1.6 Mode 6: Set Time

This mode allows the user to change the internal clock on the device. This time is used for tagging only the outgoing messages and is displayed in mode 1. Incoming messages use the time encoded in the Caller ID information; they don't use this clock. As soon as this mode is entered, the hour flashes. The following sequence is then used to set the time:



Slot 4 is full.  
Hold down ANNC to record

Figure 3-8: Display for Mode 7

1. Left Arrow: decrease hour
2. Right Arrow: increase hour
3. CLOCK: flash minute
4. Left Arrow: decrease minute
5. Right Arrow: increase minute
6. CLOCK: flash month
7. Left Arrow: decrease month
8. Right Arrow: increase month
9. CLOCK: flash day
10. Left Arrow: decrease day
11. Right Arrow: increase day
12. CLOCK: set time and go to mode 1.

If at any time DELETE is pressed, this mode is aborted and it goes back to mode 1, leaving the time as it was before entering the mode. All other buttons and events are ignored.

### **3.1.7 Mode 7: Modify Announcements**

This mode allows the user to record an outgoing message, or an announcement. There are 10 slots available to store into, with slot 0 being the default announcement.

303-538-2826	1 of 5
Daniel Day Lewis	I 12:43

Figure 3-9: Display for Mode 8

Any number not associated with an announcement will be played announcement 0.  
The behavior while in this mode is as follows:

- Left Shift: decrement slot number
- Right Shift: increment slot number
- DELETE: free current slot
- PLAY: listen to current slot, if full
- ANNC: go to mode 1
- ANNC: if held down, record until released, and replace current message if slot is full

All other buttons and events are ignored.

### 3.1.8 Mode 8: Playback Messages

This mode allows the user to play back the messages. As soon as the mode is entered it begins playing, starting with the oldest message. The behavior is as follows:

- Left Shift: rewind one message with each press
- Right Shift: advance one message with each press
- DELETE: delete current message
- PLAY: go to mode 3

All other buttons and events are ignored.

```
505-576-7255   entry #1
Slot #4 is empty
```

Figure 3-10: Display for Mode 9

### 3.1.9 Mode 9: Administer Announcement/Number Database

This mode allows the user to modify the announcement/number database, which contains associations between outgoing messages and a phone number. The behaviour while in this mode is as follows:

- Left Shift: go back in database
- Right Shift: go forward in database
- DELETE: delete entry
- ANNC: listen to current slot, if full
- \* (on phone): switch cursor between number and announcement
- # (on phone): if cursor on number, go to mode 10.
- # (on phone): if cursor on announcement, go to mode 11.
- PLAY: go back to mode 1

All other buttons and events are ignored.

### 3.1.10 Mode 10: Administer Name/Number Database

This mode allows the user to modify the name/number database, which contains associations between names and phone numbers. As soon as this mode is entered, the phone is disconnected from the phone line to allow the keypad to be used to input names and numbers. The device's behavior while in this mode is as follows:

- Left Shift: go back one entry in database

```
505-576-7255   entry #16
Cesar Chavez
```

Figure 3-11: Display for Mode 10

```
Type Number, # to accept
6162257_
```

Figure 3-12: Display for Mode 11

- Right Shift: go forward one entry in database
- DELETE: delete entry
- \* (on phone): switch cursor between number and name
- # (on phone): if cursor is on number, go to mode 10.
- # (on phone): if cursor is on name, go to mode 11.
- PLAY: go back to mode 1

All other buttons and events are ignored.

### 3.1.11 Mode 11: Number Entry Mode

This mode allows the user to enter a number using the telephone keypad. The buttons on the keypad take on the following meaning:

- 2-9: Put number at current cursor position
- 1: Loop through 0 and 1
- 0: Accept number and return to previous mode
- \*: backspace

```
Type Name, # to accept
Sal Lo_
```

Figure 3-13: Display for Mode 12

- #: Accept digit and move cursor forward one position

All other buttons and events are ignored.

### 3.1.12 Mode 12: Name Entry Mode

This mode allows the user to enter a name using the telephone keypad. The buttons on the keypad take on the following meaning:

- Letters: loop through possible letters for that key
- 0: Accept name and return to previous mode
- \*: backspace
- #: Accept letter and move cursor forward one position

All other buttons and events are ignored.

## 3.2 Current Implementation

This section describes the current implementation's deviation from the full specification. Because of time constraints and difficulties dealing with the local AT&T distributor, I was not able to acquire the AT&T TAD chipset. Consequently the answering machine functionality is not available in the current implementation. As a result, the functions which require the answering machine will not work. For example, in the Display Incoming Number mode, the device would normally go to the Play Announcement/Record Message mode. Rather than going there it goes immediately



back to the default mode, mode 1. Similarly, the modes Record Message, Modify Announcements, and Play Messages are all disabled. Everything else works as described in the previous section.

# Chapter 4

## Hardware Specification

This chapter will describe the hardware in the current implementation. Since I didn't actually get my hands on the data sheets for the TAD chipset I am not able to include it here. There are 5 major parts to the hardware: the 68HC16 evaluation board, the Motorola Caller ID Receiver, the Dual-Tone Multiple-Frequency (DTMF) Touch-Tone decoder, the LCD display, and the analog interface to the phone line.

### 4.1 The 68HC16 Evaluation Board

The 68HC16 is a 16-bit microprocessor from Motorola that can run at 16 MHz. This processor is particularly well-suited for this application because it does not require a lot of support chips. It has a serial input/output interface built in, as well as several general purpose input/output port (GPIP) pins. It also has 10 user-configurable chip-selects so that external devices can be mapped into the 68HC16's memory map, eliminating the need for external PALs to perform this function. Eight timers are also built-in to the 68HC16. It also includes other useful features such as an eight channel analog-to-digital converter and support for multiple serial devices, both of which I am currently not using.

The 68HC16 has a 16 bit data bus and a 16 bit address bus which, along with the GPIP pins and serial I/O pins, are used to connect the 68HC16 to the other chips in the system. Sockets for RAM and ROM are already built onto the 68HC16

Evaluation Board so descriptions of the RAM and ROM connections are not included here.

## **4.2 The Motorola Caller ID Receiver**

The Motorola Caller ID receiver has three outputs: the Caller ID serial data output, the ring detection line, and the carrier detection line. The serial data output is connected directly to the serial I/O input (RXD) on the 68HC16. As soon as valid Caller ID information is detected, the resulting data is driven on this line. The carrier and ring detect lines are connected to GPIF pins on the 68HC16. As long as a valid carrier is detected on the phone line the carrier detect line is pulled low. As long as a ring is detected, the ring detect line is pulled low. The schematics for this can be found in Appendix B.

## **4.3 The DTMF Touch-Tone Decoder**

According to the Touch-Tone standard, each button on a phone keypad sends a sound that is actually a combination of two tones. The buttons are separated into columns and rows (see Figure 4-1). Each row and column is associated with a different audible frequency. When a button is pressed two tones are produced: one for the row and one for the column. The combination of these two tones is what is sent out on the phone line. (See [2]).

The DTMF decoder has an input pin, which is connected to the phone line, one status output pin (DV), and 4 data output pins. The DV pin is connected to a GPIF pin on the 68HC16, and the 4 data output pins are connected to lines 8-11 of the 68HC16 data bus. The DTMF chip also has a R/W pin and an active high register enable pin. The R/W pin is connected directly to the 68HC16 R/W pin. Since the chip selects on the 68HC16 are active low, I needed to place an inverter between the chip select on the 68HC16 and the register enable pin on the DTMF receiver. Whenever the DTMF receiver detects a valid Touch-Tone at its input, it asserts DV

		Col 1	Col 2	Col 3	Col 4		
STD DTMF (Hz)	697	1	2	3	A	Row 1	
	770	4	5	6	B	Row 2	
	852	7	8	9	C	Row 3	
	941	*	0	#	D	Row 4	
		1209	1336	1477	1633		
STD DTMF (Hz)							

Figure 4-1: DTMF Keypad Layout

and places a byte corresponding to that tone onto its 4 data output pins. In order to read the data from the chip one needs to wait for the DV pin to be asserted, then read from the address for which the chip select is configured. The schematics for this circuit can be found in Appendix B.

## 4.4 The LCD Display

The LCD display has two lines of 24 characters. It contains an ASCII character generator and offers several display options. The LCD display has eight input/output data pins which are connected to lines 8-15 of the 68HC16 data bus. It also contains a register enable (RE) pin, a R/W pin, and a register select (RS) pin. The R/W pin is connected to the R/W pin of the 68HC16 and the RE pin is connected to the 68HC16 chip select corresponding to the LCD display. The RS pin tells the LCD display if the data to be transferred on the data pins is to be data or instructions. This pin is connected to the low bit of the 68HC16 address bus.

After being mapped into the 68HC16 memory map, the LCD can be accessed as two different addresses. One address is for data, and the other for instructions. The method for writing characters on the LCD is described in section 5.3. The schematics for the LCD can be found in Appendix B.

## 4.5 The Analog Phone-line Interface

Because the telephone line carries voltages up to 50 volts, and because the telephone ground voltage is floating relative to the 68HC16 ground, the device needs a reliable analog front-end between the phone-line and the digital circuitry. This front end is implemented using capacitors to isolate the circuitry from the line and also to remove the DC component of the telephone line. Another difficulty I run into is that I need the phone keypad for data entry. However, if I take the phone off-hook so I can send the touch tones to the DTMF receiver, I need some way to keep those tones from also going out onto the line, while still providing power to the phone so that it can generate the tones. This isolation is accomplished by a low-pass filter that can be actuated using a double-pole, double-throw relay. This relay is activated automatically whenever data needs to be entered using the keypad. At all other times the phone is connected to the line as usual.

# Chapter 5

## Software Specification

The software can be broken down into several major sections. These are the DTMF receiver, the input/output pins (GPIP pins), the LCD display, the Caller ID (CID) receiver, the touch tone data input interface, the main dispatch routine, the clock routines, and other miscellaneous routines.

### 5.1 GPIP pins

The 68HC16 has several pins that can be configured as general purpose input/output pins. These will be referred to as GPIP pins. This section contains the routines and macros that initialize the GPIP pins and provide a simple and consistent interface to access these pins. Definitions for the various pins are included in `gpip.h` which is included in each file that uses the pins. In order to check the state of a pin, the macro `GPIP(pin)` is defined. For example, if switch 2 is pressed, `GPIP(SWITCH2)` will be true.

### 5.2 The DTMF Receiver

As described in section 4.3, the DTMF receiver listens for valid Touch-Tones on the phone line and returns a code corresponding to the tone it recognizes. Two main functions are defined to deal with setting up the DTMF decoder and receiving data

from it.

- **DTMF\_Init:** This routine takes and returns no arguments. It maps the DTMF decoder's data output pins into address 0x40001 of the 68HC16s address space. After this mapping is done, the address 0x40001 holds the incoming touch-tone code whenever pin DV of the DTMF receiver is high. The pin DV is connected to a GPIIP pin on the 68HC16 and is referred to as the **DTMF\_BIT**.
- **DTMF\_Get:** This routine takes one argument, **type**, which specifies the format of the output. If **type** is **RAW** then it returns the number (from 0 to 12) that it gets from the DTMF decoder. If **type** is **COOKED** then it returns the ASCII character corresponding to the touch-tone that was pressed.

In order to determine if a valid tone is detected, it is necessary to monitor the **DTMF\_BIT**, which is defined in **gpiip.h**, using the expression **GPIIP(DTMF\_BIT)**. If a tone is detected, this expression becomes true, and the DTMF character can be read from the address 0x40001.

## 5.3 LCD Display

There are several functions defined to initialize LCD and send various characters to the LCD display. They are as follows:

- **LCD\_Init:** This routine takes and returns no arguments. It maps the LCD's instruction (**LCDIR**) register into address 0x50000 of the 68HC16's memory map and the LCD's data (**LCDDR**) register into address 0x50001 of the 68HC16's memory map.
- **LCD\_SendChar:** This routine takes two arguments, **data** and **type**. The **type** can be either **LCD\_INST** or **LCD\_DATA**, depending on whether the data is an instruction to the LCD display, such as turn on cursor, or if the data is an ASCII character. If **type** is **LCD\_INST**, **data** is written into the **LCDIR** register, if **type** is **LCD\_DATA**, **data** is written into the **LCDDR** register.

- **LCD\_SendString:** This routine takes two arguments, a string pointed to by **data** and **type**. The **type** can be either **LCD\_INST** or **LCD\_DATA**. The string pointed to by **data** must end in a null character (0x0).
- **LCD\_ClearScreen:** This routine takes no arguments. It uses **LCD\_SendChar** to send a clear instruction to the LCD display.
- **LCD\_MoveCursor:** This routine takes two arguments, **row** and **column**. It uses **LCD\_SendChar** to position the cursor at the specified **row** and **column**. **row** must be 0 or 1. **column** must be between 0 and 23.

## 5.4 CID Receiver

I wrote three routines to receive and process data from the Caller ID receiver.

- **CID\_get:** This routine takes one arguments, a pointer to string **cid\_buffer**. This routine listens to the CID chip and receives the serial CID data coming from the CID chip. If no recognizable message arrives before either the next ring or when the phone is picked up, or when the ringing stops, then 0 is returned as the length of **cid\_buffer**. If a message is received and placed into **cid\_buffer**, then the size of **cid\_buffer** is returned.
- **CID\_date:** This routine takes three arguments; **length**, a pointer to a string, **cid\_data**, and another pointer to a string, **date**. This routine extracts the date from a buffer containing CID data and puts it into **date**. If **cid\_data** does not contain a valid date, the first byte of **date** will be a null (0x0).
- **CID\_number:** This routine takes three arguments, **length**, a pointer to a string, **cid\_data**, and another pointer to a string, **number**. This routine extracts the phone number from a buffer containing CID data and puts it into **number**. If **cid\_data** does not contain a valid number, the first byte of **number** will be a null (0x0).



## 5.5 Touch-Tone Data Input

In order to use the keypad to enter names and numbers I wrote two routines: `TT_GetName` and `TT_GetNumber`.

- `TT_GetName`: This routine takes two arguments; `size`, and a pointer to a string `string`. The keypad of the phone is mapped so that for the buttons 2-9, each button loops through the letters for that button; first in lowercase, then in uppercase, followed by the digit for that button. Button 1 loops between space, 0, and 1. `#` moves the cursor to the next position, `*` backspaces, and 0 accepts the string as entered. After `size` characters have been entered, the string is terminated with a null character (0x0) and the routine returns. If 0 was pressed before the `size` is reached, the entered string is terminated by a null (0x0) and the routine returns.
- `TT_GetNumber`: This routine takes two arguments; `size`, and a pointer to a string `string`. The keypad of the phone is mapped so that for the buttons 2-9, each button enters the digit for that button. Button 1 loops between 0 and 1. `#` moves the cursor to the next position, `*` backspaces, and 0 accepts the string as entered. After `size` characters have been entered, the string is terminated with a null character (0x0) and the routine returns. If 0 was pressed before the `size` is reached, the entered string is terminated by a null (0x0) and the routine returns.

## 5.6 Clock Maintenance

I wrote several routines to set up and maintain a clock. They are:

- `CLOCK_Init`: This routine initializes a global array of four bytes beginning at `clock_ticks`. It also sets up a timer that calls the interrupt handler `CLOCK_advance` every tenth of a second.

- **CLOCK\_advance**: This routine is an interrupt handler that advances `clock_ticks` by one.
- **CLOCK\_todate**: This routine takes one argument, a pointer to a string `date`. This routine reads the current value of `clock_ticks`, converts it to the CID date format, and stores that in `date`.
- **CLOCK\_fromdate**: This routine takes one argument, a pointer to a string `date`. This routine sets the value of `clock_ticks`, taking the current time and date from `date` which is in the CID date format.
- **CLOCK\_resettime**: This routine calls `TT_GetNumber` to get the current date, and uses that info to set the time using `CLOCK_fromdate`.

## 5.7 Main Dispatch Routine

This routine contains the main event loop that controls all processing in the program. This routine contains the structure for the finite-state machine described in the Functional Specification (section 3).

## 5.8 Utility routines

These routines include miscellaneous routines that are necessary but don't fit in any of the other major sections.

- **SCI\_Init**: This routine initializes the serial I/O module in the 68HC16 to 1200 baud, 8 data bits, no parity, 1 stop bit. It also enables the receiver.
- **complete\_number**: This routine takes one argument, a pointer to a string `number`. It attempts to complete the phone number stored in `number` into a full 10-digit phone number. If it can't do this, it returns `FAIL` and puts a null (0x0) into the first character of `number`. If it succeeds, it returns `SUCCESS` and places the number into `number`.

# Chapter 6

## Conclusion

The purpose of this thesis is to build an answering machine that combines the speed and convenience of a digital answering machine with the advanced features of Caller ID to provide the telephone user with a level of control never before found in the home. On phone lines equipped with the Caller ID service, the phone number of the caller is encoded and sent out by the phone company. This information is sent between the first and second ring and can be retrieved to provide the callee with the caller's phone number. The device described in this thesis contains an internal database of phone number/name pairs which is scanned for a match as soon as the information becomes available. If the number is found in the database, the corresponding name is displayed on an LCD display; otherwise, the number is displayed. This feature saves the user from having to remember the numbers of frequently calling parties. This database can be updated using the telephone keypad. A log is also maintained that contains all incoming and outgoing calls. As a result, the user will know who called throughout the day, even if the caller does not leave a message on the answering machine.

The other component of this device is the integration of a digital answering machine. This makes possible specialized outgoing messages, depending on who is calling. For example, you can assign one greeting to your boss, another to your parents, and yet another to your friend. This combination of a digital answering machine and Caller ID makes possible new and discrete ways of screening calls. There is no longer

any need to wait for the answering machine to pick up in order to determine who is calling. When the answering machine answers, the appropriate greeting will be sent. Since the device knows who is calling, it can also know how to act on that call.

## 6.1 Current Implementation

The current prototype implements everything described in section 3 except for the digital answering machine. Whenever a call is made, whether outgoing or incoming, the prototype displays the number and name (if it is in the database) of the person being called or the person that is calling. It also keeps a running log of the last thirty calls made, whether incoming or outgoing. The database is modified using an input method based on using the phone's keypad to input names and numbers. The elimination of a separate keypad lessens the cost and complexity involved in building the product. Also, the fewer buttons there are, the less intimidating it will be to the consumer.

## 6.2 Future Directions

The current prototype is essentially a Caller ID decoder, with the next logical step being extending it to include the integrated digital answering machine. The development doesn't have to stop there. One very useful addition would be an RS-232 connection to a PC. The device could be programmed to send the Caller ID data out the RS-232 port as soon as it gets it. Then, if there is a specialized Personal Information Manager (PIM) running on the PC and it understands this Caller ID data, this feature could be integrated in with no extra hardware. A PC interface to the database would also be helpful. This would eliminate the admittedly awkward telephone keypad method for inputting data.

Another variation on this same theme would be to have a Caller ID on a PC expansion card. Then programs on the PC would have direct access to the Caller ID information and there would be no more need for the 68HC16 microprocessor or

the RAM and ROM. The digital answering machine could also be integrated into the PC expansion board. Then, the messages could be accessed on the computer as if they were audio electronic mail. A graphical interface to access all the features of the Caller ID digital answering machine would greatly increase the ease of use and flexibility of the product. It could provide a nice, consistent interface to all the features.

The increased control and flexibility provided by both Caller ID and digital answering machines hold great promise in terms of increasing the control that the telephone users have over their telephones. All it takes is for someone to do it!

# Appendix A

## Software

---

```
/* main.c */
```

```
#include <hc6_sup.h>
```

```
#include "main.h"
```

```
#include "lcd.h"
```

```
#include "dtmf.h"
```

```
#include "database.h"
```

```
#include "strings.h"
```

```
#include "tt.h"
```

```
#include "cid.h" 10
```

```
#include "pins.h"
```

```
isil_static STRINGS, a_iram;
```

```
static uchar STR_waitmsg[]="Waiting for message";
```

```
static uchar STR_outgoing[]="OUT";
```

```
static uchar STR_incoming[]="IN";
```

```
static uchar STR_init[]={ 0x3f, 0xc, 0x2, 0x0};
```

```
static uchar pref_25[]="61725";
```

```
static uchar pref_22[]="61722";
```

```
static uchar area_code[]="617"; 20
```

```
void MAIN(void)
```

```
{
```

```

uchar temp;
uchar cid_data[20];
uchar count;
uchar number[NUMBER_SZ];
uchar date[DATE_SZ];
uchar name[NAME_SZ];
uchar cur_digit;
30

LCD_Init();
DTMF_Init();
SCI_init();
PINS_init();

init_database();
cur_digit=0;

LCD_SendString(STR_init, LCD_INST);
LCD_ClearScreen();
40
/*
LCD_PrNumeric(45);
while(1)
    cur_digit=0;
*/

LCD_SendString(STR_waitmsg, LCD_DATA);

while (1) {
    if (GPIP(RING_BIT) == RINGING) {
50
        count=CID_get((uchar fp*)cid_data);
        CID_number((uchar fp*)cid_data, (uchar fp*)number, count);
        CID_date((uchar fp*)cid_data, (uchar fp*)date, count);
        DB_GetByNumber(NAME_NUM, (uchar fp*)name, (uchar fp*)number);
        LOG_Add(name, number, date, INCOMING);
        LCD_ClearScreen();
        LCD_MoveCursor(0,0);
        LCD_SendString(name, LCD_DATA);
        LCD_MoveCursor(0x0,0x15);

```

```

        LCD_SendString(STR_incoming, LCD_DATA);
        LCD_MoveCursor(1,0);
        LCD_PrPhoneNumber(number);
        LCD_MoveCursor(1,13);
        LCD_PrDate(date);
    }

    if (SWITCH(SWITCH1) != UP) {
        while (SWITCH(SWITCH1) != UP) {}
        LOG_Browse();
        LCD_ClearScreen();
        LCD_MoveCursor(0,0);
        LCD_SendString(STR_waitmsg, LCD_DATA);
    }

    if (SWITCH(SWITCH2) != UP) { /* Admin Name/Number Database */
        while (SWITCH(SWITCH2) != UP) {}
        RELAY_ON;
        DB_AdminNameNum();
        LCD_ClearScreen();
        LCD_MoveCursor(0,0);
        LCD_SendString(STR_waitmsg, LCD_DATA);
        RELAY_OFF;
    }

    if (GPIP(DTMF_BIT)) {
        temp=DTMF_Get(COOKED);
        if (temp != 0) {
            number[cur_digit]=temp;
            cur_digit++;
            if (cur_digit==5) {
                number[cur_digit]=0x0;
                complete_number((uchar fp*)number);
                DB_GetByNumber(NAME_NUM, (uchar fp*)name, (uchar fp*)numbe
                LOG_Add(name, number, date, OUTGOING);
                LCD_ClearScreen();
            }
        }
    }

```



```

        LCD_MoveCursor(0x0,0x0);
        LCD_SendString(name, LCD_DATA);
        LCD_MoveCursor(0x0,0x15);
        LCD_SendString(STR_outgoing, LCD_DATA);
        LCD_MoveCursor(0x1,0x0);
        LCD_PrPhoneNumber(number);
        cur_digit=0;
    }
}
}
}
}

```

```

void SCI_init(void) {
    uchar temp2;
    uint fp* sccr0;
    uint fp* sccr1;
    uint fp* scsr;
    uchar fp* scdr;

    sccr0=(uint fp*)SCCR0;
    sccr1=(uint fp*)SCCR1;
    scsr=(uint fp*)SCSR;
    scdr=(uchar fp*)SCDR+1;
    *sccr0=0x1B5;
    *sccr1=0x0004;
    if ((*scsr&0x40) != 0) temp2=*scdr;
}

```

```

uchar complete_number(uchar fp *number)
{
    uchar size, count;
    uchar temp[NUMBER_SZ];

    for (size=0; size < NUMBER_SZ; size++)
        if (number[size]==0x0) break;
}

```

```

switch(size) {
  case 10:
    return(SUCCESS);
  case 7:
    for (count=0; count < size; count++)
      temp[count]=number[count];
    for (count=0; count < 3; count++)
      number[count]=area_code[count];
    for (count=0; count < 7; count++)
      number[count+3]=temp[count];
    number[10]=0x0;
    return(SUCCESS);
  case 5:
    switch (number[0]) {
      case '3':
      case '8':
        for (count=0; count < size; count++)
          temp[count]=number[count];
        for (count=0; count < 5; count++)
          number[count]=pref_25[count];
        for (count=0; count < 5; count++)
          number[count+5]=temp[count];
        number[10]=0x0;
        return(SUCCESS);
      case '5':
        for (count=0; count < size; count++)
          temp[count]=number[count];
        for (count=0; count < 5; count++)
          number[count]=pref_22[count];
        for (count=0; count < 5; count++)
          number[count+5]=temp[count];
        number[10]=0x0;
        return(SUCCESS);
      default:
        return(FAIL);
    }
}

```

140

150

160

```
    }  
    default:  
        return(FAIL);  
    }  
}
```

170

```
main(int argc, char *argv[]) { MAIN(); }
```

---

---

*/\* main.h \*/*

**typedef unsigned short** ushort;

**typedef unsigned int** uint;

**typedef unsigned char** uchar;

**#define** FAIL 1

**#define** SUCCESS 0

**#define** TRUE 1

**#define** FALSE 0 10

**#define** NULL (void \*) 0x0

**extern void** SCI\_init(void);

**extern main**(int, char \*[]);

**extern void** MAIN(void);

**extern uchar** complete\_number(uchar fp \*);

**#define** SIMMCR 0xFFA00

**#define** SIMTR 0xFFA02

**#define** SYNCR 0xFFA04 20

**#define** RSR 0xFFA07

**#define** SIMTRE 0xFFA08

**#define** PORTE0 0xFFA11

**#define** PORTE1 0xFFA13

**#define** DDRE 0xFFA15

**#define** PEPAR 0xFFA17

**#define** PORTF0 0xFFA19

**#define** PORTF1 0xFFA1B

**#define** DDRF 0xFFA1D

**#define** PFPAR 0xFFA1F 30

**#define** SYPCR 0xFFA21

**#define** PICR 0xFFA22

**#define** PITR 0xFFA24

**#define** SWSR 0xFFA27

**#define** TSTMSRA 0xFFA30

<b>#define</b>	<b>TSTMSRB</b>	<b>0xFFA32</b>	
<b>#define</b>	<b>TSTSC</b>	<b>0xFFA34</b>	
<b>#define</b>	<b>TSTRC</b>	<b>0xFFA36</b>	
<b>#define</b>	<b>CREG</b>	<b>0xFFA38</b>	
<b>#define</b>	<b>DREG</b>	<b>0xFFA3A</b>	<b>40</b>
<b>#define</b>	<b>CSPDR</b>	<b>0xFFA41</b>	
<b>#define</b>	<b>CSPAR0</b>	<b>0xFFA44</b>	
<b>#define</b>	<b>CSPAR1</b>	<b>0xFFA46</b>	
<b>#define</b>	<b>CSBARBT</b>	<b>0xFFA48</b>	
<b>#define</b>	<b>CSORBT</b>	<b>0xFFA4A</b>	
<b>#define</b>	<b>CSBAR0</b>	<b>0xFFA4C</b>	
<b>#define</b>	<b>CSOR0</b>	<b>0xFFA4E</b>	
<b>#define</b>	<b>CSBAR1</b>	<b>0xFFA50</b>	
<b>#define</b>	<b>CSOR1</b>	<b>0xFFA52</b>	
<b>#define</b>	<b>CSBAR2</b>	<b>0xFFA54</b>	<b>50</b>
<b>#define</b>	<b>CSOR2</b>	<b>0xFFA56</b>	
<b>#define</b>	<b>CSBAR3</b>	<b>0xFFA58</b>	
<b>#define</b>	<b>CSOR3</b>	<b>0xFFA5A</b>	
<b>#define</b>	<b>CSBAR4</b>	<b>0xFFA5C</b>	
<b>#define</b>	<b>CSOR4</b>	<b>0xFFA5E</b>	
<b>#define</b>	<b>CSBAR5</b>	<b>0xFFA60</b>	
<b>#define</b>	<b>CSOR5</b>	<b>0xFFA62</b>	
<b>#define</b>	<b>CSBAR6</b>	<b>0xFFA64</b>	
<b>#define</b>	<b>CSOR6</b>	<b>0xFFA66</b>	
<b>#define</b>	<b>CSBAR7</b>	<b>0xFFA68</b>	<b>60</b>
<b>#define</b>	<b>CSOR7</b>	<b>0xFFA6A</b>	
<b>#define</b>	<b>CSBAR8</b>	<b>0xFFA6C</b>	
<b>#define</b>	<b>CSOR8</b>	<b>0xFFA6E</b>	
<b>#define</b>	<b>CSBAR9</b>	<b>0xFFA70</b>	
<b>#define</b>	<b>CSOR9</b>	<b>0xFFA72</b>	
<b>#define</b>	<b>CSBAR10</b>	<b>0xFFA74</b>	
<b>#define</b>	<b>CSOR10</b>	<b>0xFFA76</b>	
<b>#define</b>	<b>RAMMCR</b>	<b>0xFFB00</b>	
<b>#define</b>	<b>RAMTST</b>	<b>0xFFB02</b>	
<b>#define</b>	<b>RAMBAH</b>	<b>0xFFB04</b>	<b>70</b>
<b>#define</b>	<b>RAMBAL</b>	<b>0xFFB06</b>	

<b>#define</b>	<b>QMCR</b>	<b>0xFFC00</b>	
<b>#define</b>	<b>QTEST</b>	<b>0xFFC02</b>	
<b>#define</b>	<b>QILR</b>	<b>0xFFC04</b>	
<b>#define</b>	<b>QIVR</b>	<b>0xFFC05</b>	
<b>#define</b>	<b>SCCR0</b>	<b>0xFFC08</b>	
<b>#define</b>	<b>SCCR1</b>	<b>0xFFC0A</b>	
<b>#define</b>	<b>SCSR</b>	<b>0xFFC0C</b>	
<b>#define</b>	<b>SCDR</b>	<b>0xFFC0E</b>	
<b>#define</b>	<b>QPDR</b>	<b>0xFFC15</b>	<b>80</b>
<b>#define</b>	<b>QPAR</b>	<b>0xFFC16</b>	
<b>#define</b>	<b>QDDR</b>	<b>0xFFC17</b>	
<b>#define</b>	<b>SPCR0</b>	<b>0xFFC18</b>	
<b>#define</b>	<b>SPCR1</b>	<b>0xFFC1A</b>	
<b>#define</b>	<b>SPCR2</b>	<b>0xFFC1C</b>	
<b>#define</b>	<b>SPCR3</b>	<b>0xFFC1E</b>	
<b>#define</b>	<b>SPSR</b>	<b>0xFFC1F</b>	
<b>#define</b>	<b>RR0</b>	<b>0xFFD00</b>	
<b>#define</b>	<b>RR1</b>	<b>0xFFD02</b>	
<b>#define</b>	<b>RR2</b>	<b>0xFFD04</b>	<b>90</b>
<b>#define</b>	<b>RR3</b>	<b>0xFFD06</b>	
<b>#define</b>	<b>RR4</b>	<b>0xFFD08</b>	
<b>#define</b>	<b>RR5</b>	<b>0xFFD0A</b>	
<b>#define</b>	<b>RR6</b>	<b>0xFFD0C</b>	
<b>#define</b>	<b>RR7</b>	<b>0xFFD0E</b>	
<b>#define</b>	<b>RR8</b>	<b>0xFFD00</b>	
<b>#define</b>	<b>RR9</b>	<b>0xFFD02</b>	
<b>#define</b>	<b>RRA</b>	<b>0xFFD04</b>	
<b>#define</b>	<b>RRB</b>	<b>0xFFD06</b>	
<b>#define</b>	<b>RRC</b>	<b>0xFFD08</b>	<b>100</b>
<b>#define</b>	<b>RRD</b>	<b>0xFFD0A</b>	
<b>#define</b>	<b>RRE</b>	<b>0xFFD0C</b>	
<b>#define</b>	<b>RRF</b>	<b>0xFFD0E</b>	
<b>#define</b>	<b>TR0</b>	<b>0xFFD20</b>	
<b>#define</b>	<b>TR1</b>	<b>0xFFD22</b>	
<b>#define</b>	<b>TR2</b>	<b>0xFFD24</b>	
<b>#define</b>	<b>TR3</b>	<b>0xFFD26</b>	

<b>#define</b>	<b>TR4</b>	<b>0xFFD28</b>	
<b>#define</b>	<b>TR5</b>	<b>0xFFD2A</b>	
<b>#define</b>	<b>TR6</b>	<b>0xFFD2C</b>	<b>110</b>
<b>#define</b>	<b>TR7</b>	<b>0xFFD2E</b>	
<b>#define</b>	<b>TR8</b>	<b>0xFFD30</b>	
<b>#define</b>	<b>TR9</b>	<b>0xFFD32</b>	
<b>#define</b>	<b>TRA</b>	<b>0xFFD34</b>	
<b>#define</b>	<b>TRB</b>	<b>0xFFD36</b>	
<b>#define</b>	<b>TRC</b>	<b>0xFFD38</b>	
<b>#define</b>	<b>TRD</b>	<b>0xFFD3A</b>	
<b>#define</b>	<b>TRE</b>	<b>0xFFD3C</b>	
<b>#define</b>	<b>TRF</b>	<b>0xFFD3E</b>	
<b>#define</b>	<b>CR0</b>	<b>0xFFD40</b>	<b>120</b>
<b>#define</b>	<b>CR1</b>	<b>0xFFD41</b>	
<b>#define</b>	<b>CR2</b>	<b>0xFFD42</b>	
<b>#define</b>	<b>CR3</b>	<b>0xFFD43</b>	
<b>#define</b>	<b>CR4</b>	<b>0xFFD44</b>	
<b>#define</b>	<b>CR5</b>	<b>0xFFD45</b>	
<b>#define</b>	<b>CR6</b>	<b>0xFFD46</b>	
<b>#define</b>	<b>CR7</b>	<b>0xFFD47</b>	
<b>#define</b>	<b>CR8</b>	<b>0xFFD48</b>	
<b>#define</b>	<b>CR9</b>	<b>0xFFD49</b>	
<b>#define</b>	<b>CRA</b>	<b>0xFFD4A</b>	<b>130</b>
<b>#define</b>	<b>CRB</b>	<b>0xFFD4B</b>	
<b>#define</b>	<b>CRC</b>	<b>0xFFD4C</b>	
<b>#define</b>	<b>CRD</b>	<b>0xFFD4D</b>	
<b>#define</b>	<b>CRE</b>	<b>0xFFD4E</b>	
<b>#define</b>	<b>CRF</b>	<b>0xFFD4F</b>	
<b>#define</b>	<b>GPTMCR</b>	<b>0xFF900</b>	
<b>#define</b>	<b>GPTMTR</b>	<b>0xFF902</b>	
<b>#define</b>	<b>ICR</b>	<b>0xFF904</b>	
<b>#define</b>	<b>PDDR</b>	<b>0xFF906</b>	
<b>#define</b>	<b>GPTPDR</b>	<b>0xFF907</b>	<b>140</b>
<b>#define</b>	<b>OC1M</b>	<b>0xFF908</b>	
<b>#define</b>	<b>OC1D</b>	<b>0xFF909</b>	
<b>#define</b>	<b>TCNT</b>	<b>0xFF90A</b>	

<b>#define</b>	<b>PACTL</b>	<b>0xFF90C</b>	
<b>#define</b>	<b>PACNT</b>	<b>0xFF90D</b>	
<b>#define</b>	<b>TIC1</b>	<b>0xFF90E</b>	
<b>#define</b>	<b>TIC2</b>	<b>0xFF910</b>	
<b>#define</b>	<b>TIC3</b>	<b>0xFF912</b>	
<b>#define</b>	<b>TOC1</b>	<b>0xFF914</b>	
<b>#define</b>	<b>TOC2</b>	<b>0xFF916</b>	150
<b>#define</b>	<b>TOC3</b>	<b>0xFF918</b>	
<b>#define</b>	<b>TOC4</b>	<b>0xFF91A</b>	
<b>#define</b>	<b>TI4O5</b>	<b>0xFF91C</b>	
<b>#define</b>	<b>TCTL1</b>	<b>0xFF91E</b>	
<b>#define</b>	<b>TCTL2</b>	<b>0xFF91F</b>	
<b>#define</b>	<b>TMSK2</b>	<b>0xFF921</b>	
<b>#define</b>	<b>TFLG1</b>	<b>0xFF922</b>	
<b>#define</b>	<b>TFLG2</b>	<b>0xFF923</b>	
<b>#define</b>	<b>CFORC</b>	<b>0xFF924</b>	
<b>#define</b>	<b>PWMC</b>	<b>0xFF924</b>	160
<b>#define</b>	<b>PWMA</b>	<b>0xFF926</b>	
<b>#define</b>	<b>PWMB</b>	<b>0xFF927</b>	
<b>#define</b>	<b>PWMCNT</b>	<b>0xFF928</b>	
<b>#define</b>	<b>PWMBUFA</b>	<b>0xFF92A</b>	
<b>#define</b>	<b>PWMBUFB</b>	<b>0xFF92B</b>	
<b>#define</b>	<b>PRESCL</b>	<b>0xFF92C</b>	
<b>#define</b>	<b>ADCMCR</b>	<b>0xFF700</b>	
<b>#define</b>	<b>ADTEST</b>	<b>0xFF702</b>	
<b>#define</b>	<b>ADCPDR</b>	<b>0xFF706</b>	
<b>#define</b>	<b>ADCTL0</b>	<b>0xFF70A</b>	170
<b>#define</b>	<b>ADCTL1</b>	<b>0xFF70C</b>	
<b>#define</b>	<b>ADSTAT</b>	<b>0xFF70E</b>	
<b>#define</b>	<b>RJURR0</b>	<b>0xFF710</b>	
<b>#define</b>	<b>RJURR1</b>	<b>0xFF712</b>	
<b>#define</b>	<b>RJURR2</b>	<b>0xFF714</b>	
<b>#define</b>	<b>RJURR3</b>	<b>0xFF716</b>	
<b>#define</b>	<b>RJURR4</b>	<b>0xFF718</b>	
<b>#define</b>	<b>RJURR5</b>	<b>0xFF71A</b>	
<b>#define</b>	<b>RJURR6</b>	<b>0xFF71C</b>	



<b>#define</b>	<b>RJURR7</b>	<b>0xFF71E</b>	<b>180</b>
<b>#define</b>	<b>LJSRR0</b>	<b>0xFF720</b>	
<b>#define</b>	<b>LJSRR1</b>	<b>0xFF722</b>	
<b>#define</b>	<b>LJSRR2</b>	<b>0xFF724</b>	
<b>#define</b>	<b>LJSRR3</b>	<b>0xFF726</b>	
<b>#define</b>	<b>LJSRR4</b>	<b>0xFF728</b>	
<b>#define</b>	<b>LJSRR5</b>	<b>0xFF72A</b>	
<b>#define</b>	<b>LJSRR6</b>	<b>0xFF72C</b>	
<b>#define</b>	<b>LJSRR7</b>	<b>0xFF72E</b>	
<b>#define</b>	<b>LJURR0</b>	<b>0xFF730</b>	
<b>#define</b>	<b>LJURR1</b>	<b>0xFF732</b>	<b>190</b>
<b>#define</b>	<b>LJURR2</b>	<b>0xFF734</b>	
<b>#define</b>	<b>LJURR3</b>	<b>0xFF736</b>	
<b>#define</b>	<b>LJURR4</b>	<b>0xFF738</b>	
<b>#define</b>	<b>LJURR5</b>	<b>0xFF73A</b>	
<b>#define</b>	<b>LJURR6</b>	<b>0xFF73C</b>	
<b>#define</b>	<b>LJURR7</b>	<b>0xFF73E</b>	

---

---

```
/* cid.c */;
```

```
#include <hc6_sup.h>
```

```
#include "main.h"
```

```
#include "cid.h"
```

```
#include "pins.h"
```

```
uchar CID_get(uchar fp *cid_buffer)
```

```
{
```

```
    uint temp1;                                /* Stores incoming characters */    10
```

```
    uchar temp2;                                /* Stores status flag */
```

```
    uchar count;                                /* cid buffer counter */
```

```
    uchar Uflag;                                /* says if char is an initial U */
```

```
    uchar length;
```

```
    uchar overflow;
```

```
    uint fp* scsr;
```

```
    uchar fp* scdr;
```

```
    scsr=(uint fp*)SCSR;                        20
```

```
    scdr=(uchar fp*)SCDR+1;                    /* only read lower bit of RXD buffer */
```

```
    if ((*scsr&0x40) != 0) temp2=*scdr; /* clear any waiting data */
```

```
    count=0;                                    /* reset cid buffer counter */
```

```
    Uflag=1;                                    /* Set Uflag */
```

```
    overflow=0;
```

```
    while (1) {                                /* Look for message type */
```

```
        while(((temp1=*scsr)&0x40) == 0x0) {}
```

```
        if ((temp1&0x8) != 0) overflow=1;      30
```

```
                                                /* Spin until receive buffer is full */
```

```
        temp2=*scdr;                            /* store incoming char */
```

```
        if ((temp1 & 0x2) == 0)                /* only keep char if no errors */
```

```
            if (temp2==0x4)                    /* If messagetype 0x4 */
```

```
                break;                          /* then move on */
```

```

}

while (1) {
    /* This is the message length */
    while(((temp1=*scsr)&0x40) == 0x0) {}
    if ((temp1&0x8) != 0) overflow=1;
    /* Spin until receive buffer is full */
    temp2=*scdr;
    /* store incoming char */
    if ((temp1 & 0x2) == 0) {
        /* only keep char if no errors */
        length=temp2;
        /* This is the length */
        break;
        /* then move on */
    }
}

while (1) {
    /* this is the data */
    if (count==length) break;
    /* break if received length chars */
    while(((temp1=*scsr)&0x40) == 0x0) {}
    if ((temp1&0x8) != 0) overflow=1;
    /* Spin until receive buffer is full */
    temp2=*scdr;
    /* store incoming char */
    if ((temp1 & 0x2) == 0) {
        /* only keep char if no errors */
        cid_buffer[count]=temp2; /* put char in cid buffer */
        count++;
        /* increment cid buffer counter */
    }
}

cid_buffer[count]=0x0;
return(count);
/* return length of cid buffer */
}

```

```

void CID_date(uchar fp* cid_data, uchar fp* date, uchar length)

```

```

{
    uchar count;
    if (length==0) {
        date[0]=0x0;
    } else {
        for (count=0; count < 8; count ++)
            date[count]=cid_data[count];
        date[count]=0x0;
    }
}

```

```
    }  
}
```

```
void CID_number(uchar fp* cid_data, uchar fp* number, uchar length)
```

```
{
```

```
    uchar count;
```

```
    length-=8;
```

```
    switch (length) {
```

```
        case 10:
```

80

```
            for (count=0; count < 10; count ++)
```

```
                number[count]=cid_data[count+8];
```

```
            number[count]=0x0;
```

```
            break;
```

```
        case 7:
```

```
            number[0]='6';
```

```
            number[1]='1';
```

```
            number[2]='7';
```

```
            for (count=0; count < 7; count ++)
```

```
                number[count+3]=cid_data[count+8];
```

90

```
            number[count+3]=0x0;
```

```
            break;
```

```
        default:
```

```
            number[0]=0x0;
```

```
            break;
```

```
    }
```

```
}
```

---

---

*/\* cid.h \*/*

**#define**       CID\_ERROR     0

**#define**       CID\_NORM      1

**#define**       CID\_EXT       2

**extern** **uchar** CID\_get(**uchar** fp \*);

**extern** **void** CID\_date(**uchar** fp\*, **uchar** fp\*, **uchar**);

**extern** **void** CID\_number(**uchar** fp\*, **uchar** fp\*, **uchar**);

10

---

---

```
/* database.c */
```

```
#include <hc6_sup.h>
#include "main.h"
#include "lcd.h"
#include "database.h"
#include "dtmf.h"
#include "strings.h"
#include "tt.h"
#include "pins.h"
```

10

```
isil_static STRINGS, a_iram;
static uchar browse[]="Browser";
static uchar log_empty[]="The log is empty.";
static uchar in[]="IN";
static uchar out[]="OUT";
static uchar STR_empty[]="empty";
static uchar STR_entry[]="entry #";
static uchar enternumber[]="Enter Valid Phone Number";
static uchar entername[]="Enter Name";
```

20

```
isil_static DBASE, a_full;
NAME_NUM_FLD name_num_db[NAME_NUMBER_SZ];
NUM_MSG_FLD num_msg_db[NUM_MSG_SZ];
LOG_ENTRY phonelog_db[PHONELOG_SZ];
uchar phonelog_cur;
uchar phonelog_full;
```

```
uchar DB_Init(uchar db)
```

```
{
```

30

```
    uchar i;
```

```
    switch (db) {
```

```
        case NAME_NUM:
```

```
            for (i=0; i < NAME_NUMBER_SZ; i++)
```

```

        name_num_db[i].valid=FALSE;
        return(SUCCESS);
    case NUM_MSG:
        for (i=0; i < NUM_MSG_SZ; i++)
            num_msg_db[i].valid=FALSE;
            return(SUCCESS);
    default:
        return(FAIL);
}
}

```

```

uchar DB_Delete(uchar db, uchar entry)
{
    switch (db) {
        case NAME_NUM:
            if ((entry >= 0) && (entry < NAME_NUMBER_SZ)) {
                name_num_db[entry].valid=FALSE;
                return(SUCCESS);
            }
            return(FAIL);
        case NUM_MSG:
            if ((entry >= 0) && (entry < NUM_MSG_SZ)) {
                num_msg_db[entry].valid=FALSE;
                return(SUCCESS);
            }
            return(FAIL);
        default:
            return(FAIL);
    }
}

```

```

uchar DB_GetByEntry(uchar db, uchar entry, uchar fp *data1, uchar fp *data2)
{
    switch (db) {
        case NAME_NUM:

```

```

    if (name_num_db[entry].valid==FALSE) return(FAIL);
    if ((entry >= 0) && (entry < NAME_NUMBER_SZ)) {
        strcpy(data1, name_num_db[entry].name);
        strcpy(data2, name_num_db[entry].number);
        return(SUCCESS);
    } else {
        return(FAIL);
    }
}
case NUM_MSG:
    if (num_msg_db[entry].valid==FALSE) return(FAIL);
    if ((entry >= 0) && (entry < NUM_MSG_SZ)) {
        strcpy(data1, num_msg_db[entry].number);
        data2[0]=num_msg_db[entry].message;
        return(SUCCESS);
    } else {
        return(FAIL);
    }
}
}

```

80

90

```

uchar DB_GetByNumber(uchar db, uchar fp *data1, uchar fp *data2)
{
    uchar count;

    switch (db) {
        case NAME_NUM:
            for (count=0; count < NAME_NUMBER_SZ; count++) {
                if (name_num_db[count].valid==TRUE) {
                    if (strcmp(data2, name_num_db[count].number)==0) {
                        strcpy(data1, name_num_db[count].name);
                        return(SUCCESS);
                    }
                }
            }
            data1[0]=0x0;
            return(FAIL);
    }
}

```



```

    case NUM_MSG:
        for (count=0; count < NUM_MSG_SZ; count++) {
            if (num_msg_db[count].valid==TRUE) {
                if (strcmp(data1, num_msg_db[count].number)==0) {
                    data2[0]=num_msg_db[count].message;
                    return(SUCCESS);
                }
            }
        }
        data2[0]=0x0;
        return(FAIL);
    }
}

```

120

```

uchar DB_Add(uchar db, uchar entry, uchar fp *data1, uchar fp *data2)

```

```

{
    switch (db) {
        case NAME_NUM:
            if ((entry >= 0) && (entry < NAME_NUMBER_SZ)) {
                strcpy(name_num_db[entry].name, data1);
                strcpy(name_num_db[entry].number, data2);
                name_num_db[entry].valid=TRUE;
                return(SUCCESS);
            } else {
                return(FAIL);
            }
        case NUM_MSG:
            if ((entry >= 0) && (entry < NUM_MSG_SZ)) {
                strcpy(num_msg_db[entry].number, data1);
                num_msg_db[entry].message=data2[0];
                num_msg_db[entry].valid=TRUE;
                return(SUCCESS);
            } else {
                return(FAIL);
            }
    }
}

```

130

140

```
}
```

```
void LOG_Init(void)
```

```
{
```

```
    int i;
```

```
    for (i=0; i < PHONELOG_SZ; i++) phonelog_db[i].valid=FALSE;
```

```
    phonelog_cur=0;
```

150

```
    phonelog_full=FALSE;
```

```
}
```

```
void init_database(void)
```

```
{
```

```
    DB_Init(NAME_NUM);
```

```
    DB_Init(NUM_MSG);
```

```
    LOG_Init();
```

```
}
```

160

```
void LOG_Add(uchar fp*name, uchar fp*number, uchar fp*date, uchar type)
```

```
{
```

```
    strcpy(phonelog_db[phonelog_cur].name, name);
```

```
    strcpy(phonelog_db[phonelog_cur].number, number);
```

```
/*
```

```
    strcpy(phonelog_db[phonelog_cur].date, cur_date());
```

```
*/
```

```
    if (type==INCOMING)
```

```
        strcpy(phonelog_db[phonelog_cur].date, date);
```

```
    else
```

170

```
        phonelog_db[phonelog_cur].date[0]=0x0;
```

```
    phonelog_db[phonelog_cur].type= type;
```

```
    phonelog_db[phonelog_cur].valid=TRUE;
```

```
    phonelog_cur++;
```

```
    if (phonelog_cur==PHONELOG_SZ) {
```

```
        phonelog_cur=0;
```

```
        phonelog_full=TRUE;
```

```
    }
```

```
}
```

180

```
void LOG_Browse(void)
```

```
{
```

```
    uchar cur, start, stop;
```

```
    LCD_ClearScreen();
```

```
    LCD_MoveCursor(0,0);
```

```
    LCD_SendString(browse, LCD_DATA);
```

```
    start=phonelog_cur;
```

190

```
    stop=phonelog_cur-1;
```

```
    if (start==0)
```

```
        stop=PHONELOG_SZ;
```

```
    if (phonelog_full==FALSE) {
```

```
        if (phonelog_cur==0) {
```

```
            LCD_SendString(log_empty, LCD_DATA);
```

```
            while (SWITCH(SWITCH1) == UP) {}
```

```
            while (SWITCH(SWITCH1) != UP) {}
```

```
            return;
```

200

```
        }
```

```
        start=0;
```

```
        stop=phonelog_cur-1;
```

```
    }
```

```
    cur=stop;
```

```
    LCD_ClearScreen();
```

```
    LCD_MoveCursor(0,0);
```

```
    LCD_SendString((uchar fp*) phonelog_db[cur].name, LCD_DATA);
```

```
    LCD_MoveCursor(0,20);
```

```
    if (phonelog_db[cur].type==INCOMING)
```

210

```
        LCD_SendString((uchar fp*) in, LCD_DATA);
```

```
    else
```

```
        LCD_SendString((uchar fp*) out, LCD_DATA);
```

```
    LCD_MoveCursor(1,0);
```

```
    LCD_PrPhoneNumber(phonelog_db[cur].number);
```

```

LCD_MoveCursor(1,13);
LCD_PrDate(phonelog_db[cur].date);

while (SWITCH(SWITCH1) == UP) {
    if (SWITCH(SWITCH2) != UP) {
        if (cur!=start)
            if (cur==0) cur=PHONELOG_SZ;
            else cur--;
        LCD_ClearScreen();
        LCD_MoveCursor(0,0);
        LCD_SendString((uchar fp*) phonelog_db[cur].name, LCD_DATA);
        LCD_MoveCursor(0,20);
        if (phonelog_db[cur].type==INCOMING)
            LCD_SendString((uchar fp*) in, LCD_DATA);
        else
            LCD_SendString((uchar fp*) out, LCD_DATA);
        LCD_MoveCursor(1,0);
        LCD_PrPhoneNumber(phonelog_db[cur].number);
        LCD_MoveCursor(1,13);
        LCD_PrDate(phonelog_db[cur].date);
        while (SWITCH(SWITCH2) != UP) {}
    }
    if (SWITCH(SWITCH3) != UP) {
        if (cur!=stop)
            if (cur==PHONELOG_SZ) cur=0;
            else cur++;
        LCD_ClearScreen();
        LCD_MoveCursor(0,0);
        LCD_SendString((uchar fp*) phonelog_db[cur].name, LCD_DATA);
        LCD_MoveCursor(0,20);
        if (phonelog_db[cur].type==INCOMING)
            LCD_SendString((uchar fp*) in, LCD_DATA);
        else
            LCD_SendString((uchar fp*) out, LCD_DATA);
        LCD_MoveCursor(1,0);
        LCD_PrPhoneNumber(phonelog_db[cur].number);
    }
}

```

```

        LCD_MoveCursor(1,13);
        LCD_PrDate(phonelog_db[cur].date);
        while (SWITCH(SWITCH3) != UP) {}
    }
}
while (SWITCH(SWITCH1) != UP) {}
}

```

```
void DB_AdminNameNum(void)
```

260

```

{
    uchar cur_item;
    uchar cur, done;
    uchar name[NAME_SZ];
    uchar number[NUMBER_SZ];
    uchar key;
    uchar valid;
    uchar temp;
    uchar reset;

```

270

```

reset=1;
cur_item=0;
done=0;
cur=0;

```

*/\* Start at beginning of database*

```
while (done != 1) {
```

```
    if (reset==1) {
```

```
        reset=0;
```

```
        if ((valid=DB_GetByEntry(NAME_NUM, cur, name, number))==SUCCESS) {
```

```
            LCD_ClearScreen();
```

```
            LCD_MoveCursor(cur_item,0);
```

```
            LCD_SendChar('>', LCD_DATA);
```

```
        LCD_MoveCursor(0,1);
```

```
            LCD_PrPhoneNumber(number);
```

```
            LCD_MoveCursor(0,15);
```

```
            LCD_SendString(STR_entry, LCD_DATA);
```

```

        LCD_MoveCursor(0,22);
        LCD_PrNumeric(cur);
        LCD_MoveCursor(1,1);
        LCD_SendString(name, LCD_DATA);
    } else {
        LCD_ClearScreen();
        LCD_MoveCursor(0,0);
        LCD_SendString(STR_empty, LCD_DATA);
        LCD_MoveCursor(0,15);
        LCD_SendString(STR_entry, LCD_DATA);
        LCD_MoveCursor(0,22);
        LCD_PrNumeric(cur);
    }
}

if (SWITCH(SWITCH2) != UP) {
    /* Left Arrow */
    if (cur==0) {
        cur=NAME_NUMBER_SZ-1;
    } else {
        cur--;
    }
    if ((valid=DB_GetByEntry(NAME_NUM, cur, name, number))==SUCCESS) {
        LCD_ClearScreen();
        LCD_MoveCursor(cur_item,0);
        LCD_SendChar('>', LCD_DATA);
        LCD_MoveCursor(0,1);
        LCD_PrPhoneNumber(number);
        LCD_MoveCursor(0,15);
        LCD_SendString(STR_entry, LCD_DATA);
        LCD_MoveCursor(0,22);
        LCD_PrNumeric(cur);
        LCD_MoveCursor(1,1);
        LCD_SendString(name, LCD_DATA);
    } else {
        LCD_ClearScreen();
        LCD_MoveCursor(0,0);

```

```

        LCD_SendString(STR_empty, LCD_DATA);
        LCD_MoveCursor(0,15);
        LCD_SendString(STR_entry, LCD_DATA);
        LCD_MoveCursor(0,22);
        LCD_PrNumeric(cur);
    }
    while (SWITCH(SWITCH2) != UP) {}
}

if (SWITCH(SWITCH3) != UP) {           /* Right Arrow */
    while (SWITCH(SWITCH3) != UP) {}
    if (cur==NAME_NUMBER_SZ-1) {
        cur=0;
    } else {
        cur++;
    }
    if ((valid=DB_GetByEntry(NAME_NUM, cur, name, number))==SUCCESS){
        LCD_ClearScreen();
        LCD_MoveCursor(cur_item,0);
        LCD_SendChar('>', LCD_DATA);
        LCD_MoveCursor(0,1);
        LCD_PrPhoneNumber(number);
        LCD_MoveCursor(0,15);
        LCD_SendString(STR_entry, LCD_DATA);
        LCD_MoveCursor(0,22);
        LCD_PrNumeric(cur);
        LCD_MoveCursor(1,1);
        LCD_SendString(name, LCD_DATA);
    } else {
        LCD_ClearScreen();
        LCD_MoveCursor(0,0);
        LCD_SendString(STR_empty, LCD_DATA);
        LCD_MoveCursor(0,15);
        LCD_SendString(STR_entry, LCD_DATA);
        LCD_MoveCursor(0,22);
        LCD_PrNumeric(cur);
    }
}

```

```
}  
}
```

360

```
if (GPIP(DTMF_BIT)) {  
    key=DTMF_Get(COOKED);  
    switch (key) {  
        case '*':  
            if (valid==SUCCESS) {  
                if (cur_item==0) cur_item=1;  
                else cur_item=0;  
            }  
            break; 370  
        case '#':  
            if ((cur_item==0) && (valid==SUCCESS)) {  
                do {  
                    LCD_ClearScreen();  
                    LCD_MoveCursor(0,0);  
                    LCD_SendString(enternumber, LCD_DATA);  
                    temp=TT_GetNumber((uchar fp*)number,NUMBER_SZ);  
                    if (temp==0) break;  
                } while (complete_number(number)==FAIL); 380  
                if (temp != 0)  
                    DB_Add(NAME_NUM, cur, name, number);  
            }  
            if ((cur_item==1) && (valid==SUCCESS)) {  
                LCD_ClearScreen();  
                LCD_MoveCursor(0,0);  
                LCD_SendString(enternumber, LCD_DATA);  
                temp=TT_GetName((uchar fp*)name,NAME_SZ);  
                if (temp!=0)  
                    DB_Add(NAME_NUM, cur, name, number);  
            }  
  
            if (valid==FAIL) {  
                LCD_ClearScreen();  
                LCD_MoveCursor(0,0);  
            }  
        }  
    }  
}
```



```

LCD_SendString(entername, LCD_DATA);
temp=TT_GetName((uchar fp*)name,NAME_SZ);
if (temp!=0) {
    do {
        LCD_ClearScreen();           400
        LCD_MoveCursor(0,0);
        LCD_SendString(enternumber, LCD_DATA);
        temp=TT_GetNumber(number,NUMBER_SZ);
        if (temp==0) break;
    } while (complete_number(number)==FAIL);
    if (temp!=0)
        DB_Add(NAME_NUM, cur, name, number);
    }
}
}
reset=1;
}
if (SWITCH(SWITCH4) != UP) {
    while(SWITCH(SWITCH4) != UP) {}
    done=1;
}
}
}

```

410

420

---

*/\* database.h \*/*

```
#define NAME_SZ 20
#define NUMBER_SZ 11
#define DATE_SZ 11
#define NAME_NUMBER_SZ 30
#define NUM_MSG_SZ 30
#define PHONELOG_SZ 30
#define INCOMING 0
#define OUTGOING 1
```

10

```
typedef struct {
    uchar name[NAME_SZ];
    uchar number[NUMBER_SZ];
    uchar valid; /* TRUE or FALSE */
} NAME_NUM_FLD;
```

```
typedef struct {
    uchar number[NUMBER_SZ];
    uchar message; /* Which outgoing message? */
    uchar valid; /* TRUE or FALSE */
} NUM_MSG_FLD;
```

```
typedef struct {
    uchar name[NAME_SZ];
    uchar number[NUMBER_SZ];
    uchar date[DATE_SZ];
    uchar type; /* INCOMING or OUTGOING */
    uchar valid; /* TRUE or FALSE */
} LOG_ENTRY;
```

30

*/\* valid Database values \*/*

```
#define NAME_NUM 1
#define NUM_MSG 2
```

*/\* Prototypes \*/*

**extern uchar DB\_Add(uchar, uchar, uchar fp \*, uchar fp \*);**

**extern uchar DB\_GetByNumber(uchar, uchar fp \*, uchar fp \*);**

**extern uchar DB\_GetByEntry(uchar, uchar, uchar fp \*, uchar fp \*);**

**extern uchar DB\_Delete(uchar, uchar);**

40

**extern uchar DB\_Init(uchar);**

**extern void init\_database(void);**

**extern void LOG\_Add(uchar fp\*, uchar fp\*, uchar fp\*, uchar);**

**extern void LOG\_Browse(void);**

**extern void DB\_AdminNameNum(void);**



---

```
/* dtmf.c */
```

```
#include <hc6_sup.h>
```

```
#include "main.h"
```

```
#include "lcd.h"
```

```
#include "dtmf.h"
```

```
#include "pins.h"
```

```
isil_static STRINGS, a_iram;
```

```
static uchar DTMF_PHONE[]=" 1234567890*#"; 10
```

```
void DTMF_Init(void)
```

```
{
```

```
    uint fp* cspar1;
```

```
    uint fp* csor6;
```

```
    uint fp* csbar6;
```

```
    cspar1=(uint fp*)CSPAR1;
```

```
    csor6=(uint fp*)CSOR6;
```

```
    csbar6=(uint fp*)CSBAR6; 20
```

```
    *cspar1>(*cspar1 & 0xfffc) | 0x2;
```

```
    *csbar6=0x400;
```

```
    *csor6=0xb820;
```

```
}
```

```
uchar DTMF_Get(uchar type)
```

```
{
```

```
    uchar fp* dtmfr;
```

```
    uchar temp; 30
```

```
    dtmfr=(uchar fp*)DTMFR;
```

```
    if (GPIP(DTMF_BIT)) {
```

```
        if (type==COOKED) temp=DTMF_PHONE[*dtmfr & 0xf];
```

```
        else temp=*dtmfr & 0xf;
    while(GPIP(DTMF_BIT)) {}
    return(temp);
} else {
    return (0);
}
}
```

40



---

*/\* dtmf.h \*/*

**#define** DTMF\_BASE 0x40000  
**#define** DTMF\_R DTMF\_BASE+1  
**#define** RAW 0  
**#define** COOKED 1

**extern void** DTMF\_Init(**void**);  
**extern uchar** DTMF\_Get(**uchar**);

---

---

```
/* lcd.c */
```

```
#include <hc6_sup.h>
#include "main.h"
#include "lcd.h"
#include "database.h"
```

```
void LCD_Init(void)
```

```
{
    uint fp* cspar1;
    uint fp* csor7;
    uint fp* csbar7;
    uint temp;

    cspar1=(uint fp*)CSPAR1;
    csor7=(uint fp*)CSOR7;
    csbar7=(uint fp*)CSBAR7;

    *cspar1=(*cspar1 & 0xff3) | 0x8;
    *csbar7=0x500;
    *csor7=0x3f60;
}
```

```
void LCD_ClearScreen(void)
```

```
{
    LCD_SendChar((uchar) 0x1, LCD_INST);
}
```

```
void LCD_MoveCursor(uchar column, uchar row)
```

```
{
    LCD_SendChar(0x80 + row + column*0x40, LCD_INST);
}
```

```
uchar LCD_SendChar(uchar data, uchar type)
```

```
{
```

```

uchar fp* ldir;
uchar fp* lcddr;
int i;

ldir=(uchar fp*)LCDIR;
lcddr=(uchar fp*)LCDDR;

for(i=0; i < 100; i++) {}          /* Give LCD time to get ready */

while((*ldir & 0x80) != 0) {}     /* Loop until LCD is ready */

switch (type) {
    case LCD_DATA:
        *lcddr=data;
        break;
    case LCD_INST:
        *ldir=data;
        break;
    default:
        return FAIL;
}
return SUCCESS;
}

uchar LCD_SendString(uchar fp*data, uchar type)
{
    uchar counter, temp;

    if(!((type == LCD_DATA) || (type == LCD_INST)))
        return FAIL;              /* If the data isn't a valid type */
                                    /* then return with a FAIL */

    for(counter=0; data[counter] != 0x0; counter++) {
        temp=data[counter];
        LCD_SendChar(data[counter], type);
    }
}

```



```

    return SUCCESS;
}

uchar LCD_PrDate(uchar fp*data)
{
    uchar size;
    uchar counter;

    for (size=0; size < NUMBER_SZ; size++)
        if (data[size]==0x0) break;
    if (size != 8) return (FAIL);

    for(counter=0; counter < 10; counter++) {
        if (counter==2) LCD_SendChar('/', LCD_DATA);
        if (counter==4) LCD_SendChar(' ', LCD_DATA);
        if (counter==6) LCD_SendChar(':', LCD_DATA);
        LCD_SendChar(data[counter], LCD_DATA);
    }
    return SUCCESS;
}

uchar LCD_PrPhoneNumber(uchar fp*data)
{
    uchar size;
    uchar counter;

    for (size=0; size < NUMBER_SZ; size++)
        if (data[size]==0x0) break;
    if (size != 10) return (FAIL);

    for(counter=0; counter < 10; counter++) {
        if ((counter==3) || (counter==6))
            LCD_SendChar('-', LCD_DATA);
        LCD_SendChar(data[counter], LCD_DATA);
    }
    return SUCCESS;
}

```

```
}
```

```
uchar LCD_PrNumeric(uchar data)
```

110

```
{
```

```
    uchar counter;
```

```
    uchar place0;
```

```
    uchar place1;
```

```
    uchar place2;
```

```
    place0=0;
```

```
    place1=0;
```

```
    place2=0;
```

120

```
    while (data != 0) {
```

```
        if (data >= 100) {
```

```
            data-=100;
```

```
            place2++;
```

```
        } else {
```

```
            if (data >= 10) {
```

```
                data-=10;
```

```
                place1++;
```

```
            } else {
```

```
                if (data > 0) {
```

```
                    data-=1;
```

```
                    place0++;
```

```
                }
```

```
            }
```

```
        }
```

```
    }
```

130

```
    place2+='0';
```

```
    place1+='0';
```

```
    place0+='0';
```

140

```
    if (place2=='0') {
```

```
        place2=place1;
```

```
    place1=place0;
    place0=0x0;
    if (place2=='0') {
        place2=place1;
        place1=0x0;
    }
}
```

150

```
LCD_SendChar(place2, LCD_DATA);
if (place1 != 0x0) LCD_SendChar(place1, LCD_DATA);
if (place0 != 0x0) LCD_SendChar(place0, LCD_DATA);
}
```

---

---

*/\* lcd.h \*/*

```
#define LCD_BASE    0x50000
#define LCDIR      LCD_BASE+0
#define LCDDR      LCD_BASE+1
#define LCD_INST   0
#define LCD_DATA   1
```

```
extern void LCD_Init(void);
extern void LCD_ClearScreen(void);
extern void LCD_MoveCursor(uchar, uchar);
extern uchar LCD_SendChar(uchar, uchar);
extern uchar LCD_SendString(uchar fp*, uchar);
extern uchar LCD_PrPhoneNumber(uchar fp*);
extern uchar LCD_PrNumeric(uchar);
extern uchar LCD_PrDate(uchar fp*);
```

10

---

```
/* pins.c */
```

```
#include <hc6_sup.h>
```

```
#include "main.h"
```

```
#include "pins.h"
```

```
uchar fp* gtpdr;
```

```
uchar fp* qpdr;
```

```
void PINS_init(void) {
```

10

```
    uchar fp* tctl1;
```

```
    uchar fp* tctl2;
```

```
    uchar fp* pddr;
```

```
    uchar fp* qpar;
```

```
    uchar fp* qddr;
```

```
    tctl1=(uchar fp*)TCTL1;
```

```
    tctl2=(uchar fp*)TCTL2;
```

20

```
    pddr=(uchar fp*)PDDR;
```

```
    qpar=(uchar fp*)QPAR;
```

```
    qddr=(uchar fp*)QDDR;
```

```
    *tctl1=0;
```

```
    *tctl2=0;
```

```
    *pddr=0x8;
```

```
    /* All pins configured as inputs */
```

```
    *qpar=0;
```

30

```
    *qddr=0;
```

```
    gtpdr=0xFF907;
```

```
    qpdr=0xFFC15;
```

}



---

```
/* pins.h */
```

```
#define GPIIP(x)      (*gptpdr&x)
#define SWITCH(x)    (*qpdr&x)
#define RELAY_ON     *gptpdr=*gptpdr|RELAY
#define RELAY_OFF    *gptpdr=*gptpdr&RELAY_MASK
#define DTMF_BIT     0x1
#define RING_BIT     0x2
#define CARRIER_BIT 0x4
#define RELAY        0x8
#define RELAY_MASK   0xF7
```

10

```
#define SWITCH1     0x1
#define SWITCH2     0x2
#define SWITCH3     0x4
#define SWITCH4     0x8
```

```
#define UP          0
#define RINGING     0
```

20

```
/*
```

```
isil_static STRINGS, a_iram;
static uchar fp* gptpdr=0xFF907;
static uchar fp* qpdr=0xFFC15;
*/
```

```
extern uchar fp* gptpdr;
```

```
extern uchar fp* qpdr;
```

```
extern void PINS_init(void);
```

30

---

---

```
/* strings.c */
```

```
#include <hc6_sup.h>
```

```
#include "main.h"
```

```
#include "strings.h"
```

```
uchar strcpy(uchar fp*a, uchar fp*b)
```

```
{
```

```
    while ((*a = *b) != '\0') {
```

```
        a++;
```

10

```
        b++;
```

```
    }
```

```
}
```

```
uchar strcmp(uchar fp*a, uchar fp*b)
```

```
{
```

```
    for( ; *a==*b; a++, b++)
```

```
        if (*a=='\0') return 0;
```

```
    return (*a-*b);
```

```
}
```

20



---

```
/* strings.h */
```

```
extern uchar strcpy(uchar fp*, uchar fp*);
```

```
extern uchar strcmp(uchar fp*, uchar fp*);
```

---

---

```
/* tt.c */
```

```
#include <hc6_sup.h>
```

```
#include "main.h"
```

```
#include "lcd.h"
```

```
#include "dtmf.h"
```

```
#include "database.h"
```

```
#include "strings.h"
```

```
#include "tt.h"
```

```
isil_static STRINGS, a_iram;
```

10

```
static uchar name_keypad[10][9]={
```

```
{ ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' }, { ' ', '0', '1', ' ', ' ', ' ', ' ', ' ', ' ', ' ' },  
{ 'a', 'b', 'c', 'A', 'B', 'C', '2', ' ', ' ', ' ' }, { 'd', 'e', 'f', 'D', 'E', 'F', '3', ' ', ' ', ' ' },  
{ 'g', 'h', 'i', 'G', 'H', 'I', '4', ' ', ' ', ' ' }, { 'j', 'k', 'l', 'J', 'K', 'L', '5', ' ', ' ', ' ' },  
{ 'm', 'n', 'o', 'M', 'N', 'O', '6', ' ', ' ', ' ' }, { 'p', 'q', 'r', 's', 'P', 'Q', 'R', 'S', '7', ' ' },  
{ 't', 'u', 'v', 'T', 'U', 'V', '8', ' ', ' ', ' ' }, { 'w', 'x', 'y', 'z', 'W', 'X', 'Y', 'Z', '9', ' ' };
```

```
static uchar name_info[]={0, 3, 7, 7, 7, 7, 7, 9, 7, 9};
```

```
static uchar number_keypad[10][9]={
```

```
{ ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' }, { '0', '1', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' },  
{ '2', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' }, { '3', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' },  
{ '4', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' }, { '5', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' },  
{ '6', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' }, { '7', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' },  
{ '8', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' }, { '9', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' };
```

20

```
static uchar number_info[]={0, 2, 1, 1, 1, 1, 1, 1, 1, 1};
```

```
uchar TT_GetName(uchar fp *string, uchar size)
```

```
{
```

```
    uchar cur_key;
```

```
    uchar cur_letter;
```

```
    uchar cur_char;
```

```
    uchar last_key;
```

```
    uchar done;
```

```
    uchar temp;
```

```
    uchar foo;
```

30

```

LCD_SendChar(0xe,LCD_INST); /* Turn cursor on */
LCD_MoveCursor(0x1,0x0);
cur_letter=0;
cur_key=0;
last_key=0;
done=0;
while(done==0) {
    temp=DTMF_Get(RAW);
    if (temp != 0) {
        switch (temp) {
            case 1:
            case 2:
            case 3:
            case 4:
            case 5:
            case 6:
            case 7:
            case 8:
            case 9:
                if (last_key!=temp) cur_key=0;
                last_key=temp;
                cur_char=name_keypad[temp][cur_key];
                LCD_SendChar(cur_char, LCD_DATA);
                LCD_MoveCursor(0x1, cur_letter);
                cur_key++;
                if (cur_key==name_info[temp]) cur_key=0;
                break;
            case 10:
                string[cur_letter]=0x0;
                done=1;
                break;
            case 11:
                LCD_SendChar(' ', LCD_DATA);
                if (cur_letter != 0) cur_letter--;
                cur_char=string[cur_letter];
                LCD_MoveCursor(0x1, cur_letter);

```

```

        break;
    case 12:
        LCD_SendChar(cur_char, LCD_DATA);
        string[cur_letter]=cur_char;
        cur_letter++;
        cur_key=0;
        if (cur_letter==size) {
            string[size]=0x0;
            done=1;
        }
        break;
    default:
        break;
    }
}
}
LCD_SendChar(0xC,LCD_INST); /* Turn cursor off */
return(cur_letter);
}

```

80

90

```

uchar TT_GetNumber(uchar fp *string, uchar size)
{
    uchar cur_key;
    uchar cur_letter;
    uchar cur_char;
    uchar last_key;
    uchar done;
    uchar temp;
    uchar foo;

    LCD_SendChar(0xe,LCD_INST); /* Turn cursor on */
    LCD_MoveCursor(0x1,0x0);
    cur_letter=0;
    cur_key=0;
    last_key=0;
    done=0;
}

```

100

```

while(done==0) {
    temp=DTMF_Get(RAW);
    if (temp != 0) {
        switch (temp) {
            case 1:
            case 2:
            case 3:
            case 4:
            case 5:
            case 6:
            case 7:
            case 8:
            case 9:
                if (last_key!=temp) cur_key=0;
                last_key=temp;
                cur_char=number_keypad[temp][cur_key];
                LCD_SendChar(cur_char, LCD_DATA);
                LCD_MoveCursor(0x1, cur_letter);
                cur_key++;
                if (cur_key==number_info[temp]) cur_key=0;
                break;
            case 10:
                string[cur_letter]=0x0;
                done=1;
                break;
            case 11:
                LCD_SendChar(' ', LCD_DATA);
                if (cur_letter != 0) cur_letter--;
                cur_char=string[cur_letter];
                LCD_MoveCursor(0x1, cur_letter);
                break;
            case 12:
                LCD_SendChar(cur_char, LCD_DATA);
                string[cur_letter]=cur_char;
                cur_letter++;
                cur_key=0;

```

```
        if (cur_letter==size) {
            string[size]=0x0;
            done=1;
        }
        break;
    default:
        break;
}
}
}
LCD_SendChar(0xC,LCD_INST); /* Turn cursor off */
return(cur_letter);
}
```

150

---

*/\* tt.h \*/*

**extern** uchar TT\_GetName(uchar fp\*, uchar);

**extern** uchar TT\_GetNumber(uchar fp\*, uchar);

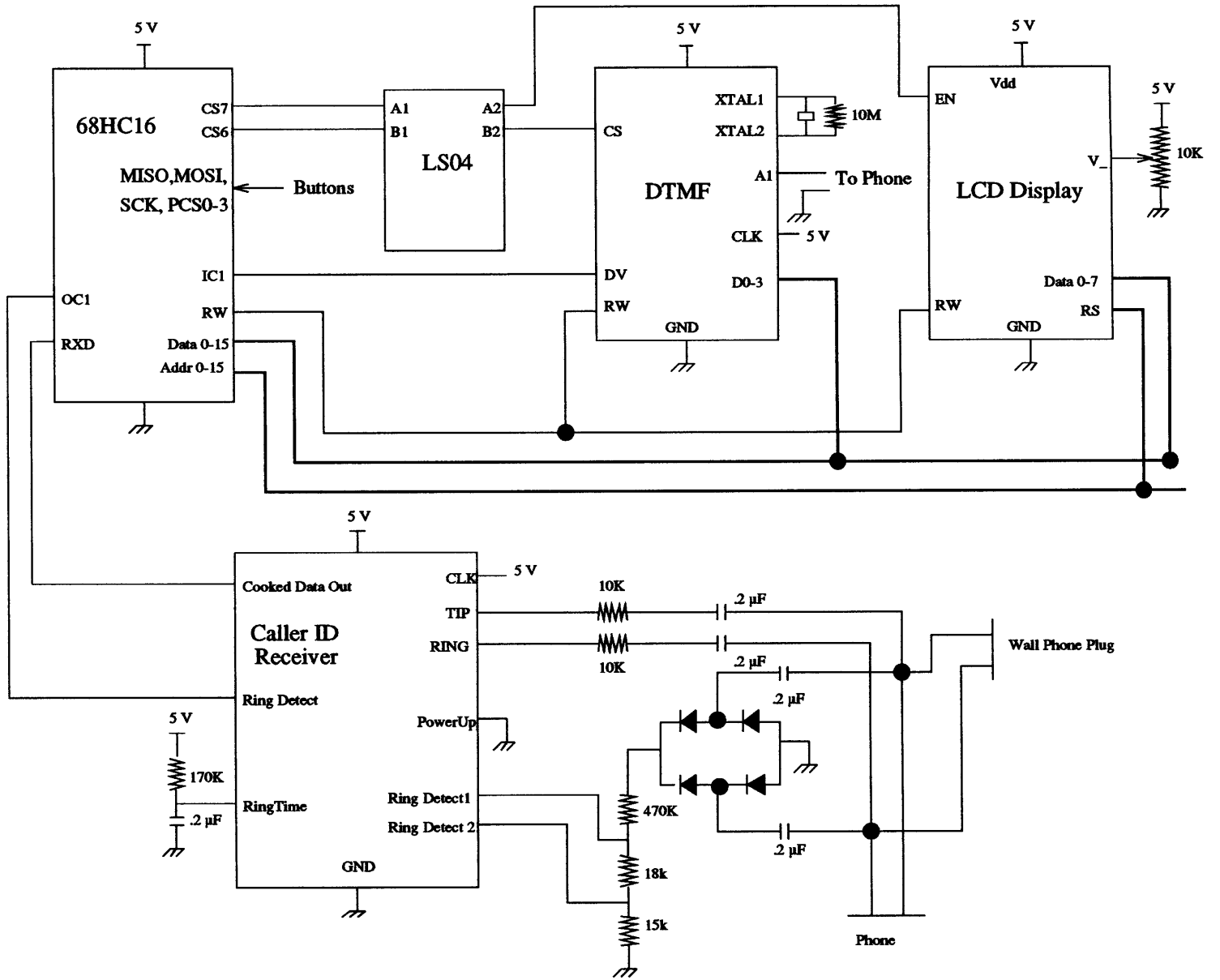
---

# **Appendix B**

## **Schematics**



Figure B-1: Schematic of Hardware



# Bibliography

- [1] Rochelle Communications. Caller id telephone line simulator. *Rochelle Communications Product Literature*, 1991.
- [2] Motorola Inc. Dual tone multiple frequency receiver. *Motorola Semiconductor Technical Data*, 1989.
- [3] Motorola Inc. Calling line identification receiver with ring detector. *Motorola Semiconductor Technical Data*, 1991.
- [4] Motorola Inc. M68hc16z1evb user's manual. *Motorola Semiconductor Technical Data*, 1992.
- [5] AT&T Microelectronics. At&t telephone answering device (tad) chipset datasheet. *AT&T Microelectronics Product Note*, 1991.
- [6] Eric Schneider. Isil 4.0 compiler. *Eris Systems Product Literature*, 1991.