Don DeLaMare
Kevin Brown
Brian Faires

Cell Phone Controlled Security System

Project Proposal

Project Description

Introduction

Home security and home automation are wonderful features that everyone would like to enjoy if they weren't so expensive to install and maintain. A less expensive alternative is an automated security system that the owner monitors himself. This is made possible through the use of a cell phone and a home phone line. The home phone line is used to contact the owner at specified phone numbers in the event of security problems. For example, if the user is not home and the window sensor is tripped, the system will call the owner indicating that the window sensor was tripped. The owner can then listen to what is happening inside the home with the help of microphones near each sensor that transmit through the phone line when that sensor is triggered. The owner can then communicate with whoever is inside the house through the use of a speaker. Then they have the options of setting off the siren or calling a neighbor to check on the house. If the system cannot contact the owner, it will use an internal ordered list of other numbers to make backup calls. When there is a security problem the owner should be the first to know and have the ability to be in control of the situation with the use of their cell phone.

Design Overview

The phone line and each of the security devices will be interfaced with a microcontroller. The security devices include a door sensor, window sensor, motion sensor, smoke alarm, security keypad, LCD screen, microphones, speaker, and siren. The keypad, LCD screen and one microphone will be mounted together near the door sensor. The microcontroller will be located in a safe place. If the window sensor is tripped while the alarm is armed, the system immediately calls the owner. The smoke sensor will trigger a call to the owner regardless of the armed/unarmed state. If the door sensor is triggered, the microcontroller prompts the keypad near the door for an access code. If a correct code isn't entered within 30 seconds (or whatever time is specified), the owner is called. Further action is determined by user-specified siren logic. Other user options, which can be set via the keypad or phone line, include: edit primary password, add/delete secondary passwords, edit primary phone number, add/delete secondary phone numbers, and set siren logic.

The system will also interface with any typical answering system the owner may have. The microcontroller will use a relay to control the answering machine. If the owner calls in and wishes to access the system, they can enter a code. Depending on the validity of the code, the microcontroller will use the relay to enable/disable the answering machine.

Home automation features are not currently included in the design. If time permits, a thermostat control and temperature sensor will be added, so the owner can change temperature settings remotely.

Risks

Hardware

The hardware risk we have is if we can not wire up all our devices to the microcontroller. We are trying to eliminate unnecessary wires. We are doing this by sending only the signals we need to the microcontroller, tying other controls high or low, and wiring up the power or ground separately. We are also trying to do serial communication instead of parallel communication wherever possible. We may need to use a serial to parallel converter. We will also use decoders and/or encoders to eliminate wires.

Software

If the program takes up too much space and cannot fit on 32k bytes of EPROM, we will have to cut functionality for some user-specified options. Worst case would be losing some minimal functionality/logic. The priorities for functionality are as follows:

All sensors
LCD Screen
Microphones
Text to Speech chip
Phone line interaction
Relays
-- Cuttable functionality --
Answering Machine interaction
User-defined red state logic
Multiple passwords/users

Interface Specifications

Top-level Interface

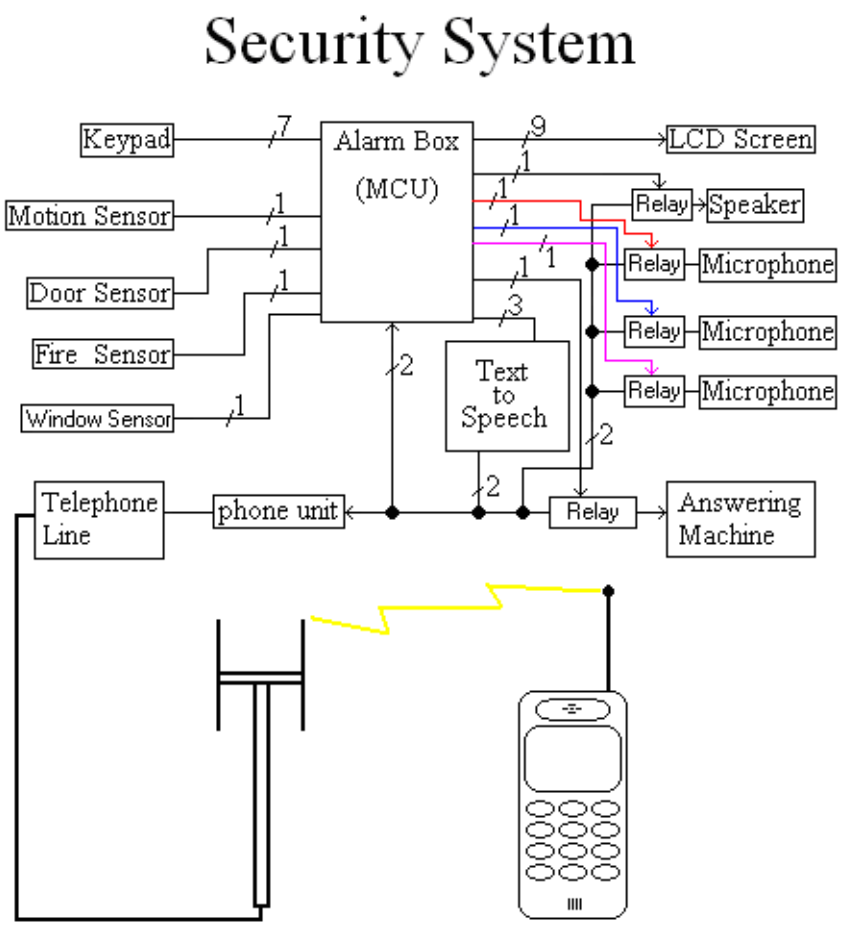

Figure 1: Top-level Interface

The fire, door, motion, and window sensors, microphones, relays, and speaker will be hooked up to power and/or ground and will only send or receive their signal to or from the microcontroller.

Microcontroller to phone unit

The phone unit will have to pick up the phone on incoming calls if there is no answering machine. The microcontroller will constantly listen to the phone unit for a ring (high voltage pulse) and answer the phone (allowing a current to flow back on the phone line) after the desired number of rings. Then it will send out a string to the text to speech chip to say, “enter code.” Then the phone unit will listen for a low and a high frequency to detect a row and a column of a four by four matrix to match a button press. If the correct code was entered, it will go to the menus of the security system, then continue sending the menus' strings to the text to speech chip and listen for button presses to switch between menus.

The Microcontroller->Text to Speech->Phone line diagram shown below.

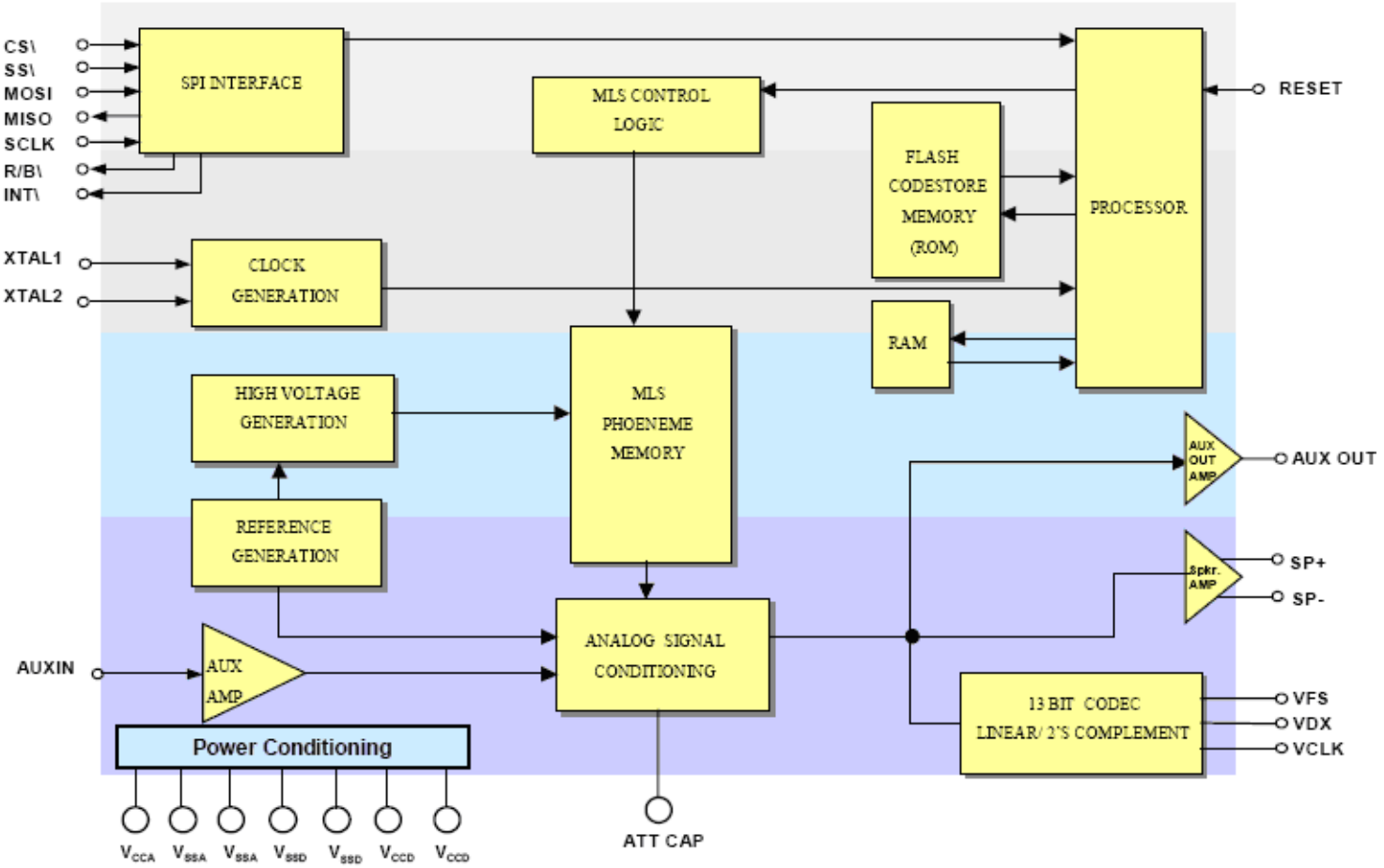

Figure 2: Text to Speech Chip Description

(Text to Speech)

The Text to speech chip has five control lines that we will use in our security system that are SS (Slave Select), MOSI (Master Out, Slave In), MISO (Master In, Slave Out), SCLK (Serial Clock), and R/B (Ready/Busy signal) as shown in figure 1. SP+, SP-, and ATT CAP will also be used on the chip.

SS is an input that is low during communication and high when communication is done.

MOSI is a digital input for the data by SPI as shown in Figure 2.

MISO echoes back what was sent on MOSI to make sure it was received.

SCLK controls the timing of the data over MOSI and MISO lines with a max frequency of 5 MHz.

R/B tells if the FIFO is full or not. Ready for more data or not.

SP+ and SP- to hook up to the phone line to speak the strings sent from the microcontroller.

ATT CAP will have a 4.5 micro farad capacitor hooked up to it to ground to reduce noise.

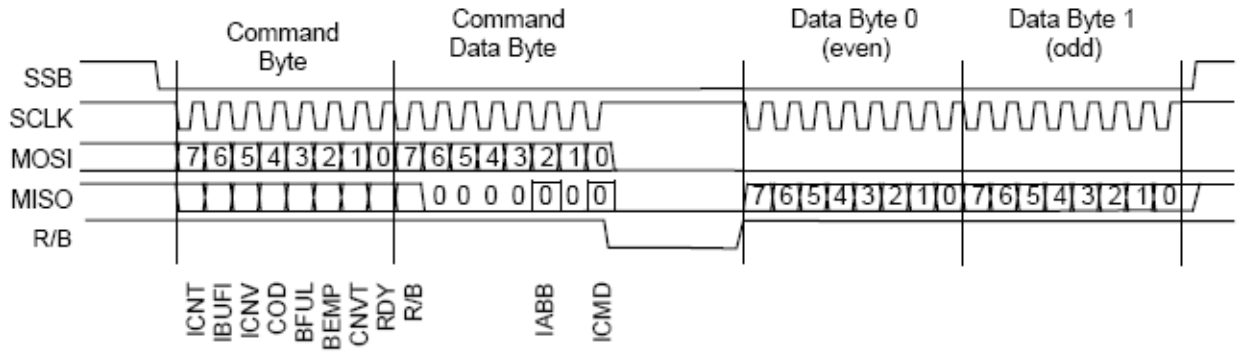

Figure 3: Text to Speech Interface

In Figure 3 MOSI transmits the data to the text to speech chip in 16-bit words. The first word is a command word to set operations, status, and customization of the text to speech chip. Then the following words are the letters in lower case ASCII to words to be read separated by spaces. Upper case ASCII are commands that can later be sent to modify settings. Some of these setting are start, stop, pause, resume, volume up/down, speedup/slowdown talking speed, and set speech pitch.

Microcontroller to Keypad

The keypad will have 7 parallel wires with four inputs and three outputs. The microcontroller will send a high bit to one of the rows, then will read the columns and save them in an array called "allKeys" to see if there was a button pressed. Then the microcontroller will do the same thing for the next three rows, adding them to the array allKeys. Then allKeys is XOR'd with the “oldAllKeys” to see if there is a difference. If there is a new key pressed the value is sent the function that is executing.

Microcontroller to LCD Screen

Communication to the LCD screen is done through Hitachi's HD 44780 microcontroller, which controls the LCD display. This interface to the Hitachi chip requires at least 4-bits and up to 8-bits if the master micro-controller is capable of output to 8-bits. Below is an example of a 8-bit communication to the HD 44780.

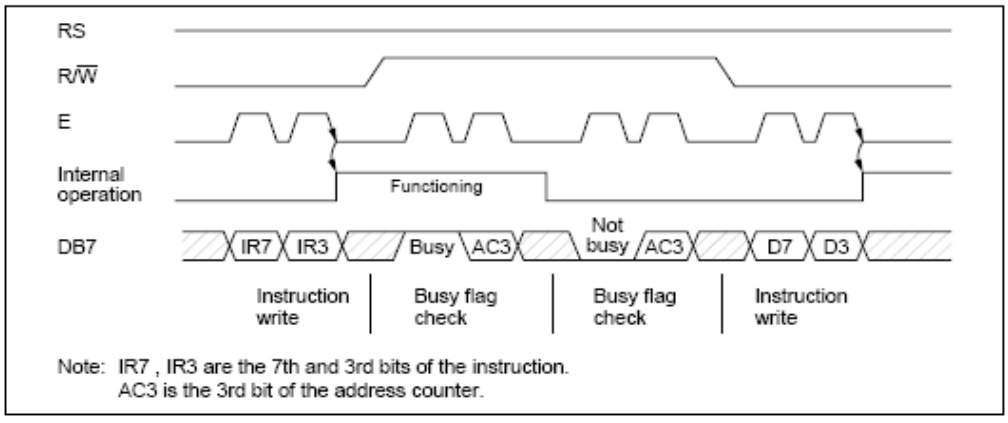

Figure 4: 8-bit Data Transfer Timing Sequence to HD 44780

Data line descriptions are as follows: The E line starts data read/write; RS (Register Select) selects the registers, 0 for instruction register (write) address counter (read), or 1 for Data register; R/W (Read/Write) selects read (0) or write (1).

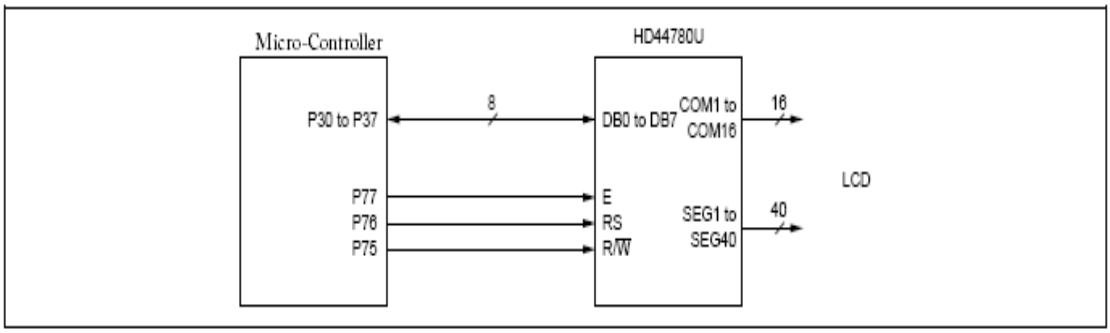

Figure 5: Example of Interface to a Micro-controller

So, as seen in Figure 5, a total of 11 wires are required for 8-bit communication. We may be able to tie R/W to high if only writes to the LCD are necessary, enabling a 10-wire interface.

Software

The code included in Appendix A is meant to clarify how the software will use global variables and functions in C to resemble an object oriented design. The objective is to maintain three separate state machines that indirectly interact. Some of the objects have input buffers that must be regularly checked, regardless of the state of the other state machines. Recognizing that the size of this software program is inherently limited, we feel that global variables and functions are appropriate here. This is because they allow more efficient control of all objects by all objects by ignoring encapsulation. They will help minimize both instruction memory requirements and the demand on our stack. Ignoring encapsulation will require anyone changing the code to have a thorough understanding of what should communicate with what. Since our number of programmers will be at most three, we feel that this will not be a difficult task.

The three state machines are represented by three global variables called currState, currPhoneState and currKeypadMenu. Our main control loop uses functions UpdatePhone() and UpdateKeypad() to update the last two states, and also supplies the logic based on currState, which represents our primary state machine. The Update functions will convert the keypad and phone input into convenient data structures such as allKeys[], lastKeypress, and currPhoneInput. The Update functions also will process this input to determine the states of all machines. After this is done, the sensors are checked and actions the primary state may be changed. If necessary, currPhoneState will be set to EXITING to signal to the phone object that it should end the current phone call the next UpdatePhone(). Lastly, output for the LCD screen and phone line will be determined and sent.

The enumerations at the top of Appendix A provide a clear means of handling our state machines. The first enumeration, titled States, is for our main state currState. Here is a brief description of its values:

UNARMED: The system is unarmed and the keypad available for arming the system or changing user options. The fire sensor is active and the phone line available to handle incoming calls.

GREEN: The common armed state. All sensors are active but none have been triggered. The phone line is

available to incoming calls.

YELLOW: A sensor is triggered from the GREEN state. A countdown begins and unless a password is entered at the keypad, the ORANGE state will be triggered. If a correct code is entered, the system will go to the UNARMED state.

ORANGE: There is a problem at the home. The phone object now rejects incoming calls and disconnects if there is a call active. Then it calls the primary phone.

RED: The primary phone is either unavailable, or has signalled an emergency once they were contacted in the ORANGE state. As per user specified options, the system will contact secondary numbers and/or trigger the siren.

Schedule Flow

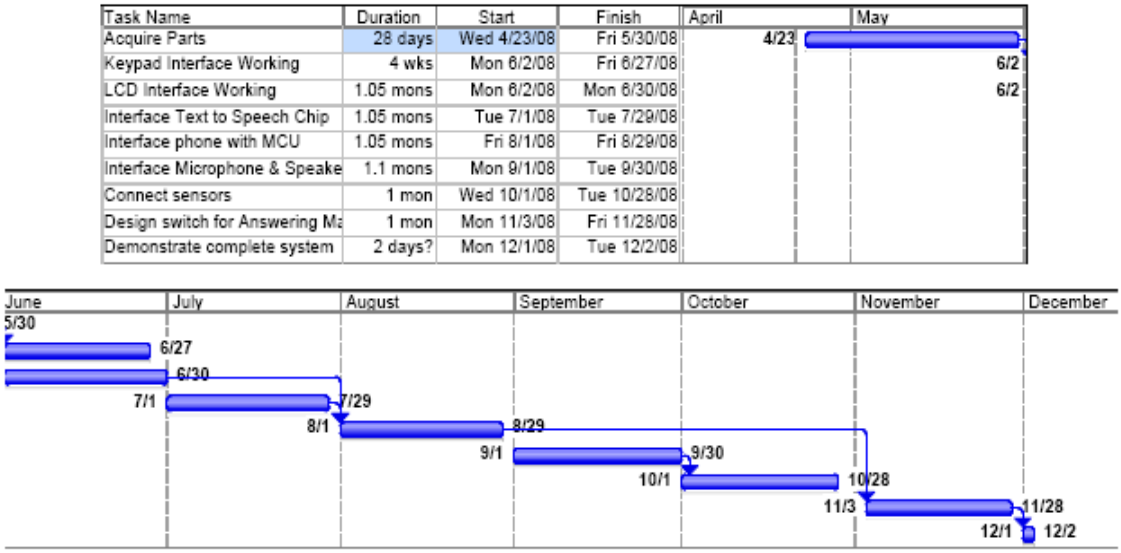

Figure 6: Schedule Flow

March
Everyone will Research where to buy sensors, microcontroller, microphones, relay, speaker, text to speech chip, and LCD screen.

April/May
Purchase microcontroller, LCD screen, keypad, text to speech chip.

June
Step 1 - Keypad and LCD Screen interfaces working
Kevin will wire up the keypad and the LCD Screen to microcontroller.
Brian will design the software for the keypad and the LCD.
Don will test the software on the micro-controller with the keypad and LCD.

July
Step 2 -Interface text to speech chip and speaker with the microcontroller.

Kevin will wire up the text to speech and the microcontroller.
Brian will design the software to send strings to the unit.
Don will test the text to speech unit with the speeker.

August

Step 3 - Interface the telephone line with the microcontroller.
Kevin will wire up the telephone line and the microcontroller.
Brian will design the software for the telephone line control.
Don will test the phone line answer / call / number input / microphone.

September

Step 4 - Interface the microphones to the microcontroller.
Kevin will wire up the microphones to microcontroller.
Brian will design the software for the relays for themicrophones.
Don will test the speaker output with the microphones.

October

Step 5 - Interface the motion, door, and fire sensors with appropriate microphone to the microcontroller.
Kevin will wire up the sensors to the microcontroller.
Brian will design the software for the sensor input.
Don will test all the sensors to set off alarm in the right state.

November

Step 6 - Interface the relay to the microcontroller for the answering machine.
Kevin will wire up the relay for the answering machine.
Brian will design the software for the relay control.
Don will test the relay to see if it will cut off the answering machine.

December

Test by Achieving full code coverage and hardware signal coverage for the Homeowner Controlled Security System.
Kevin will make sure all signals logic going to and from the hardware is tested.
Brian will make sure all software works all together on the microcontroller.
Don will test to make sure the right signals are sent from the microcontroller to the hardware.

Bill of Materials

Item	Company	Model	Cost	Contact
Microcontroller	Freescale	APS12C32SLK	donated	andy.mastronardi@freescale.com
Keypad	Grayhill	96-12	$9	UofU Lab parts counter
Smoke Detector		any	$10	Don has an extra
Text to Speech Chip	Winbond	WTS-701	$38	www.textspeak.com, sales@textspeak.com, 203-557-0222
1st Choice Siren	RadioShack	108dB Piezo	$10.49	www.radioshack.com/home/index.jsp
2nd Choice Siren	RadioShack	102dB	$5.49	www.radioshack.com/home/index.jsp
Relay	RadioShack	275-232	$2.99	www.radioshack.com/home/index.jsp
Door Sensor	Home Security and Automation	recessed roller plunger	$4.50	www.homesecurityandautomation.com
Microphone	RadioShack	270-090	$2.79	www.radioshack.com/home/index.jsp
LCD Display 16x2	BG Micro	LCD1031	$5.95	http://www.bgmicro.com
LCD controller	BG Micro	HD44780	with LCD	http://www.bgmicro.com
Motion Sensor	RadioShack	276-033	$9.99	www.radioshack.com/home/index.jsp

Appendix A

(The pseudo-code below excludes 'break's in switch statements and some other excessive code when the intended meaning is obvious.)

```
// Enumerations
enum States{UNARMED, GREEN, YELLOW, ORANGE, RED, FIRE};
enum KeypadMenus{ARMED, COUNTDOWN, ENTER_CODE, MENU0, MENU1, ... , MENU15}
enum PhoneStates{OFFLINE, RINGING, ANSWERING, EXITING, DIALING, MENU0, MENU1, ... ,
MENU15}

enum Sensors{FIRE, WINDOW, DOOR, KEYPAD};
enum Keys{NONE, ZERO, ONE, TWO, ... , NINE, POUND, STAR};

// Permanent/Nearly static Data
Keys[] primaryPassword;
Keys[][] secondaryPasswords;
Keys[] primaryNumber;
Keys[][] secondaryNumbers;



// Keypad Data
Keys keypadBuffer[BUFFER_SIZE] = Keys.NONE;
KeypadMenus currKeypadMenu = MENU0;
bool allKeys[12] = false;
bool oldAllKeys[12] = false;
bool newKeys[12] = false;
Keys lastKeypress = Keys.NONE;



// Phone Data
Keys phoneBuffer[BUFFER_SIZE] = Keys.NONE;
PhoneStates currPhoneState = OFFLINE;
Keys currPhoneInput = Keys.NONE;
Keys prevPhoneInput = Keys.NONE;



// Other Dynamic Data
States currState;
Sensors lastSensor;
float countdown;


// Functions
Main()
```

```
{
   while(true)
   {
       // Check for new input
      UpdateKeypad();
      UpdatePhone();

// Do state logic (sensor checking)
      if(IsOn(FIRE))
           currState = FIRE;



      switch(currState)
      {

          case UNARMED:

          case GREEN:
             if(IsOn(DOOR))
             {
                    currState = YELLOW;
                lastSensor = DOOR;
             }
             if(IsOn(WINDOW))
             {
                currState = YELLOW;
                lastSensor = WINDOW;
             }
             if(IsOn(KEYPAD))
             {
                currState = YELLOW;
                lastSensor = KEYPAD;
             }
          case YELLOW:
             if(timer < 0)
                currState = ORANGE;
          case ORANGE:
             if(currPhoneState == OFFLINE)
                Call primary number;
             else if(currPhoneState == DIALING)
                ;
             else
                currPhoneState = EXITING;

          case RED:
               Call backup numbers and/or trigger siren. Defined by user.
          case FIRE:
               if(currPhoneState == OFFLINE)

                  Call primary number;

              else
```

```
                currPhoneState = EXITING;

            Trigger siren;
        }



        // Process Output
        OuputLCD();
        if(currPhoneState != OFFLINE)
            OuputPhoneMenu();
    }
}



void UpdateKeypad()
{
    UpdateKeypadInput();
    for(i=1 to 12)
        if(newKeys[i])
            BufferKeypadInput(i);
    if(keypadBuffer[0])
        ProcessKeypadInput();
}



void UpdateKeypadInput()
{
    oldAllKeys = allKeys;
    allKeys=0;
    for(i=1 to 3)
    {
        send high bit to ith keypad column;
        mask 4 inputs from keypad into 12bit value;
        allKeys = allKeys XOR mask;
    }

    for(i=1 to 12)
    {
        if(allKeys[i] && !oldAllKeys[i])
            newKeys[i] = true;
    }
}



void BufferKeypadInput(int keypress)
{
    for(i=0 to BUFFER_SIZE-1)
    {
        if(keypadBuffer[i] == NONE)
```

```
            keypadBuffer[i] = keypress;
    }
    lastKeypress = keypress;
}



void ProcessKeypadInput()
{
    switch(currKeypadMenu)
    {

     case ARMED:
          lastSensor = KEYPAD;
          currState = YELLOW;
     case (COUNTDOWN || ENTER_CODE):
         if(lastKeypress == ENTER)
         {
             if(keypadBuffer == any password)
             {
                 currKeypadMenu = MENU0;
                 currState = UNARMED;
             }
             else if(incorrectPWs < 3)
                 repeat;
             else if(incorrectPWs == 3)
                 currState = ORANGE;
         }
     case MENU0:
        if(lastKeypress == a menu option)
           currKeypadMenu = MENU1;
      case MENU1:
       ...
    }
    lastKeypress = NONE; // Since its been processed
}



bool IsOn(Sensors sensor)
{
    switch(sensor)
    {
        case FIRE:
            return fire sensor input;
        case WINDOW:
            return window sensor input;

         case DOOR:
            return door sensor input;
        case KEYPAD:
            for(i=1 to 12)
               if(allKeys[i])
```

```
                return true;
    }

    return false;

}

// Analagous to Keypad logic
void UpdatePhone();
void UpdatePhoneInput();
void BufferPhoneInput();



// To be implemented when hardware is in hand

void OutputLCD();
void OutputPhoneMenu();
```

Reference Page

Microcontroller documentation	http://www.cs.utah.edu/classes/cs5780/doc/MC9S12C128V1.pdf
LCD Screen Controller Chip Documentation HD44780	http://www.eio.com/hd44780.pdf
Text to Speech Chip documentation	http://www.datasheetcatalog.org/datasheets/560/371357_DS.pdf
Phone Line Interfacing	http://www.epanorama.net/circuits/teleinterface.html