Cell Phone Controlled Security and Home Automation System

by Kevin Brown, Don DeLaMare and Brian Faires

Table of Contents

Chapter 1 Introduction

Imagine a high-end home security system with no monthly maintenance fees, made possible using an list of automatically contacted phone numbers, synthetic speech and dual-tone detection to interface with the owner via a phone connection. The implementation of such a system requires a home phone line, a cell phone or touch-tone phone, and a power supply to reliably monitor a home utilizing motion, door, window, and fire sensors.

The home phone line is used to contact trusted parties at specified phone numbers in the event of a fire alarm or security issue. For example, if the window sensor gets tripped and no password is entered at the keypad, the system will call the owner and indicate that the window sensor was tripped. The owner can then listen to what is happening inside the home with microphones near each sensor that transmit through the phone line when that sensor is triggered. The owner can then communicate with whomever is inside the house through the use of a speaker phone built into the system. If the security system cannot contact the owner, it will use an internal ordered list of other numbers to make further backup calls. The contacted party can then take immediate action, such as calling a neighbor or the police. When there is a security problem the owner will be the first to know and have the ability to have control of the situation with the use of their cell phone.

Settings on the security system or home automation system can be changed at home or by cell phone. The base system at the house includes a keypad and LCD screen for on-site use. Internal menus are displayed on the screen and can be navigated once the admin password has been verified. Non-admin passwords can only be used to arm and disarm the system. Options with an admin password include adding, deleting and reordering phone numbers in the call list, adding or deleting passwords, changing the admin password, arming or disarming the system, and adjusting automatic temperature controls.

Home automation is closely related to a security system such as this, and is intended to be added to the base package. With automation features a homeowner can remotely toggle appliances such as air conditioning and heating units, lamps or porch lights, landscape sprinkler timers, snow-melt systems, outdoor property lighting, and safety lighting. Any of these features could be added using the existing relays in this system.

Cell Phone Controlled Security and Home Automation System

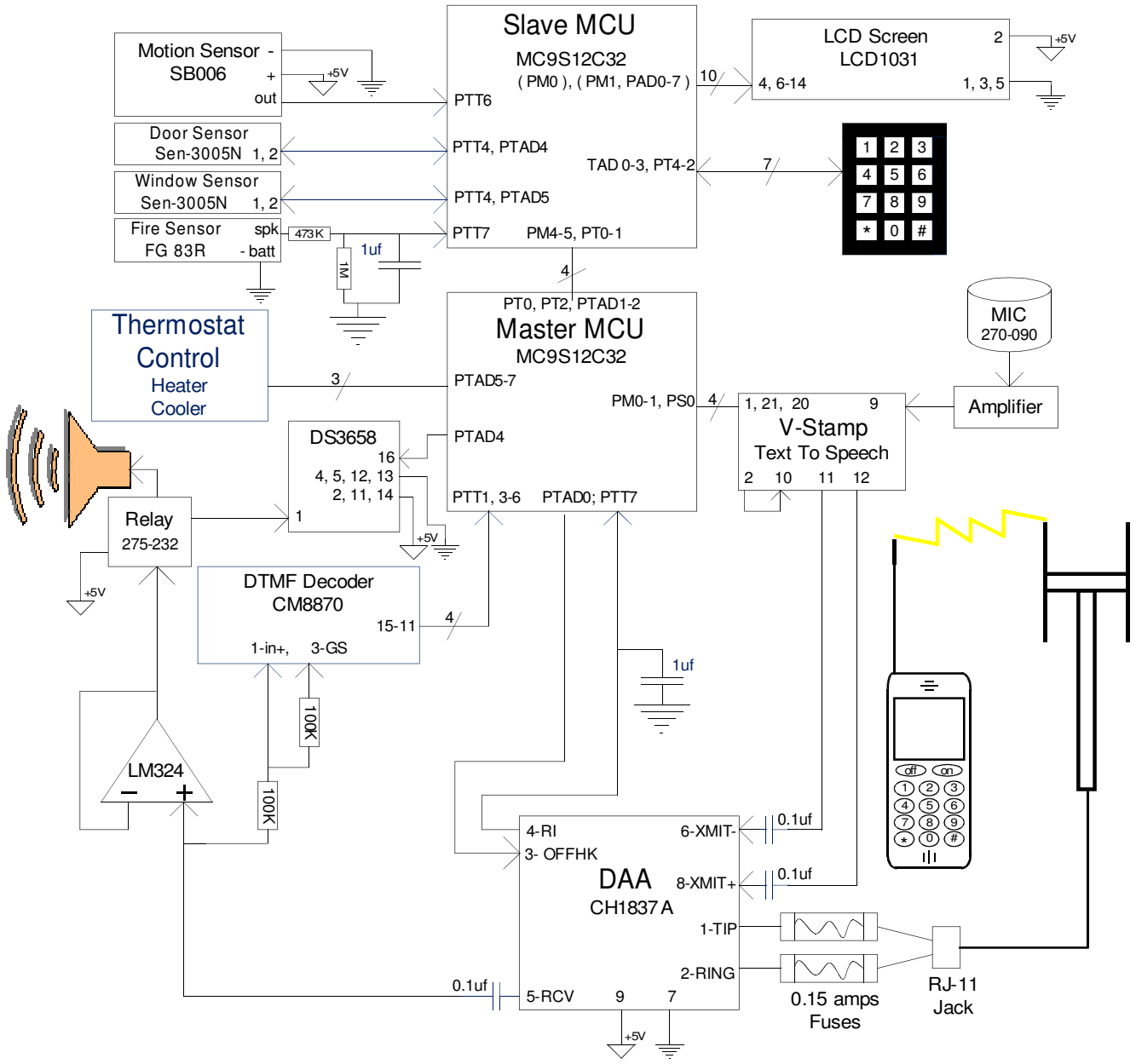


Figure 1.1

Chapter 2 Slave Microcontroller

The slave microcontroller handles the functionality of the LCD screen and security system keypad. It also interfaces with all the security sensors. See Appendix A.2 for a schematic of the slave sub-system.

LCD Screen and Keypad

In Figure 2.1 is an image of the 16x2 LCD screen, 102 Grayhill Keypad, and connectors for the sensors. The LCD screen is controlled via a Hitachi HD44780 Microcontroller, which takes in 8 data lines. Each line

carries one of 8 bits of a command, or a character to the LCD's controller from the Slave microcontroller. All characters are displayed by the slave microcontroller sending the LCD controller the corresponding ASCII code by default. [1]

A 102 Grayhill Keypad is used for user input. Each row of the keypad is polled high one at a time and the columns are read. The keypad has to have pull down resistor configuration to prevent static noise left on the wires; without it, this noise causes undesired button pushes. [2]

The LCD screen and the keypad share output ports. The LCD screen has an enable bit that is low when the microcontroller is getting input from the keypad. Then when the LCD screen is updating, the enable pin is high and the microcontroller is not looking for input from the keypad.

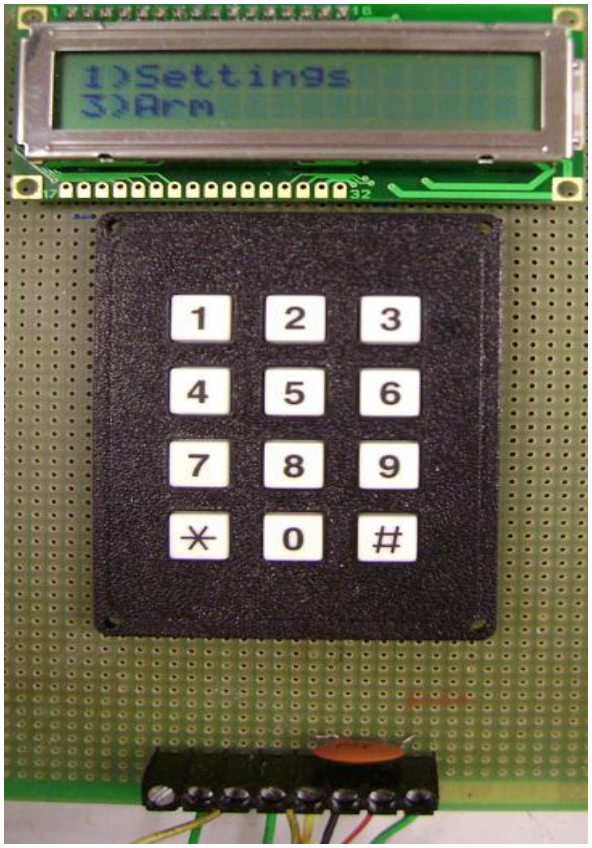


Figure 2.1

At the bottom of Figure 2.1 are the connectors for the sensors. The sensors are interfaced in three different ways.

Door and Window Sensors

The door and window sensor employ polling. A polling signal is sent out from the slave microcontroller on one wire of the sensors, if the signal returns on the 2nd data line, the circuit is closed and the sensor is not triggered. If the signal does not return, the sensor has been tripped and the circuit is open.

Figure 2.2

Motion Sensor

The motion sensor has a signal line, which goes high to 3.3V if motion is detected. It must have power (+5V) and ground furnished by an external power source. Please see the project website (URL in Appendix B), which contains a full data sheet for the motion sensor module. An image of the motion sensor is shown in Figure 2.3

Figure 2.3

Fire/Smoke Sensor

A 1uF capacitor on the grounded line of the smoke detector helps to eliminate noise and false alarms. The signal wire is simply tied to the smoke detector's speaker with a voltage divider. The voltage divider is used to step down the voltage to a safe level for the microcontroller.

Chapter 3 Master Microcontroller

Digital Access Arrangement Module

A Digital Access Arrangement Module (DAA) chip was utilized to interface with the outside phone line. We attached a fuse to both wires of the phone line to limit potential hazards to the security system and the outside phone company equipment. The DAA handles the incoming sound signals through receive port (RCV) and sends outgoing sound signals through the xmit +/- ports. The master microcontroller uses the OFFHK port of the DAA allows to send a signal to pick-up or hang-up the phone call. [3] For a schematic of the master microcontroller subsystem, see Appendix A.1.

The Ring Indicator (RI) on the DAA his a high signal that toggles to low when a call signal is received on the phone-line. The signal is captured by the microcontroller which is configured as a pull-up resistor interrupt. A capacitor coupled to ground was added so the interrupt would not be triggered by noise. After a few rings the microcontroller picks up the phone by a high signal to the OFFHK port of the DAA.

The RCV port of the DAA goes to both a Dual-tone multi-frequency (DTMF) decoder circuit and to a speaker system supported by a voltage follower circuit. The voltage follower was added to eliminate

potential power drops when the speaker was powered on. The voltage follower allows the DTMF decoder to read signals while maintaining sufficient voltage supply to the speaker.

Dual Tone Multi-Frequency Chip

The DTMF decoder is used to detect incoming button presses. This chip detects when a button was pushed. Then a four bit code is passed to the microcontroller indicating which key on the phone is pressed. [7]

Relay and Peripheral Driver

A relay is used to turn the signal on to the speaker when desired by a button press on the cell phone. The relay needs 0.5 amps to trigger it. The microcontroller can only supply 3 milliamps. With the help of a Quad High Current Peripheral Driver the relay can be turned off and on by coupling it with a microcontroller[6]. The relay is always hooked up to a five volt power supply, and the driver grounds the relay when a signal is received from the microcontroller. Chris Strong deserves credit for suggesting the peripheral driver modification.

V-Stamp Text-to-Speech Chip

Data is sent from the master microcontroller to the V-Stamp chip serially. The chip has text-to-speech capabilities, meaning it can take ASCII text and automatically convert it into speech to an audio output. Commands such as voice type, voice volume and speed, and emphasis/accents are sent by first sending hex 0x01, followed by the command number and letter (9S is fastest speed, 0S is slowest). Once configured correctly, the unit can read any text sent to it. Some words must be spelled phonetically to be pronounced correctly.

A DTMF generator is on the V-Stamp chip. Phone numbers are dialed by sending a phone number command to the chip. Each number generates a 100ms number tone followed by 100ms pause before the next number tone. [5]

The V-Stamp chip also can handle an analog input which the circuit utilizes to add the microphone input to the phone line for the speakerphone. The microphone has its own little amplifier circuit (see Appendix A.3) which connects the V-Stamp chip and amplifies the signal. This also allows the microphone to pick up sounds throughout the majority of the house. [4] This allows the home owner to hear what is occurring that has set off the alarm.

Figure 3.1

Thermostat

Thermostat control is done by measuring the temperature of the house and turning the heater or air conditioner on and off automatically. An analog input to the master microcontroller via a 2k-100k thermistor enables the microcontroller to know the temperature of the house. Then two outputs are used to turn the heater and cooler on and off .

Chapter 4 Communication Between the Microcontrollers

The Cell Phone Controlled Security and Home Automation System is controlled by two microcontrollers. The microcontrollers have 32k ROM and 2k RAM. Figure 4.1 shows one of these microcontrollers; this microcontroller is used to control the system using interrupt, analog-to-digital, general in out, and serial ports. All code was written in C computer language using Code Warrior. [8]

Communication between the microcontrollers is done by asynchronous, interrupt-driven message passing. This allows each microcontroller to send messages at its convenience while not having to immediately process messages coming from the other microcontroller. Incoming messages are queued for the receiving microcontroller to handle at its own convenience.

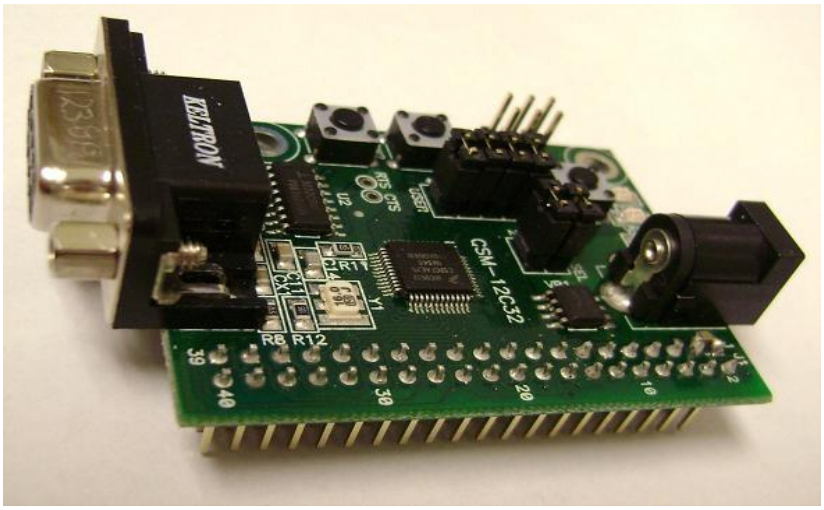


Figure 4.1

For each microcontroller, messages are sent on two pins and received on two interrupt ports. One of the interrupts is designated as the *outer interrupt*, and is pulsed high to signal the start and end of a message packet. The other interrupt, designated the *inner interrupt*, is pulsed high N times between outer interrupt pulses. N corresponds to the message encoding, which is saved on each microcontroller. Figure 4.2 shows how the message corresponding to N==4 would be sent.

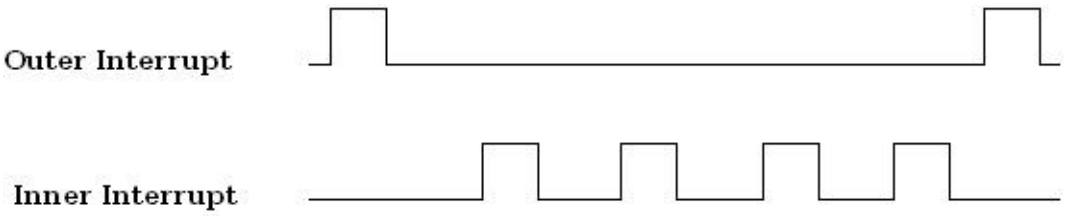


Figure 4.2

The first outer interrupt pulse initializes a counter that is incremented by the inner interrupt pulses. The second outer interrupt pulse stores the message in a first-in-first-out message buffer. From the non-exceptional code's point of view, messages instantly appear in the buffer. The only necessity is to periodically check the message buffer and execute the corresponding logic.

Some messages require multiple packets to be sent and interpreted at the same time. Furthermore, the number of packets in a given message may be of variable length. To handle this, header and footer packets are sent with messages that require them. When a header is initially read, the code on the receiving waits until the footer exists somewhere in the buffer. When this happens, the entire message is interpreted and regular control flow continues.

Chapter 5 Control Flow

The Slave microcontroller, after a one-time initialization, executes a single control loop indefinitely. The first stage of this loop updates the keypad state and then processes any new input. If the current menu, which serves as a unique identifier of the system's current state, is a buffered input menu then new key presses are added to a keypad buffer. Otherwise, an immediate jump to a new menu is made immediately. All logic that is unique to particular menus is executed in the function JumpToMenu(byte newMenu). The next stage of the forever loop is a call to the function ProcessMessages(). This function checks the message buffer described in Chapter 4 and processes the tasks called for by any new messages. Next, the sensors are checked and, if necessary, calls to JumpToMenu() are made. Timers may also be started in this stage. The timers are checked in the last stage of the forever loop. There, active timers are incremented if the real time interrupt has been triggered. This interrupt triggers once per second. Once the timers are incremented, they are checked against their maximum values. If any timers are equal to their maximum value they have expired and unique logic exists for each scenario. The most common scenario is when a sensor has been tripped and no password entered at the keypad. In this case, a JumpToMenu(LOCKDOWN_MENU) is made to disable the keypad, and a message is sent to the Master microcontroller describing the sensor that was activated.

The Master microcontroller also goes from a one-time initialization into an endless control loop. The first stage handles the temperature controls. It just checks the temperature, checks it against the user defined settings, and toggles the heater or air conditioner if necessary. The next stage is a call to ProcessMessages(), which functions the same as the function in the Slave microcontroller. For the last stage, the online status of the phone line is checked. If online, the keypad is checked and any new input is processed as in the Slave. That is to say the key press is buffered or immediately processed depending upon the value of the current menu's boolean value bufferedInput. If the phone is offline, the ring detection is checked and the phone pickup timer started.

Any stage of either control flow can make calls to JumpToMenu() and Send(), which can change the state of either microcontroller at any time. Also, when some messages are sent the control flow will stall until a response is received. This makes it necessary to avoid the possibility of deadlock. This is done by ensuring that all idle waiting is done by calling ProcessMessages(). Since all waiting is preceded by sending a message who's task requires a response, no situation exists where both microcontrollers can be waiting for a response and neither sending one.

Chapter 6 Retrospection

One of the issues that was most formidable was trying to use serial peripheral interface (SPI) between the two microcontrollers. We felt that SPI would be ideal; however, we mistakenly assumed it was vital to our design. We should have considered alternative message passing schemes. Finally, after several weeks, an interrupt driven message passing system was used that utilized four data lines between the slave

and master. An unfortunate side effect was that everything in the project was halted by not being able to interface the master and slave in a timely fashion. We should have paralleled more elements in our design phase.

Initially, the chip for synthesizing speech was a much more inexpensive version that required soldering on a very small scale. After several attempts to solder small lead wires to the needed pins, it was determined a better solution was ordering another chip in a larger form-factor which allowed more workability. The time spent learning to solder on such a small scale was not wasted and the smaller chip should definitely be considered for future products with more experienced developers, especially if cost is an issue; it was roughly 1/4 the cost of the larger form-factor chip, but did not have DTMF generator capability.

The relay in the final circuit did not meet our design needs. A relay with a smaller triggering current and voltage would have been ideal. The most difficult thing was finding a relay with a small enough coil current and voltage tolerance. In the end, a peripheral driver (part DS3658) was used to generate enough of a signal to trip the relay, which worked fine. The design would have been simplified if we had found a better relay. Please see Appendix A.3 for a detailed schematic of the master microcontroller and peripheral driver wiring.

The original design also included doing DTMF detection using just the microcontrollers and pulse-width modulation (PWM). It was infinitely better in hindsight to just use a pre-boxed solution rather than re-engineer our own solution. Simplifying our design by adding pre-packaged solutions gave us time to focus on the "bigger" problems, such as getting the base-system functional, microcontroller communication, phone-line interfacing, and finalizing other important system design details.

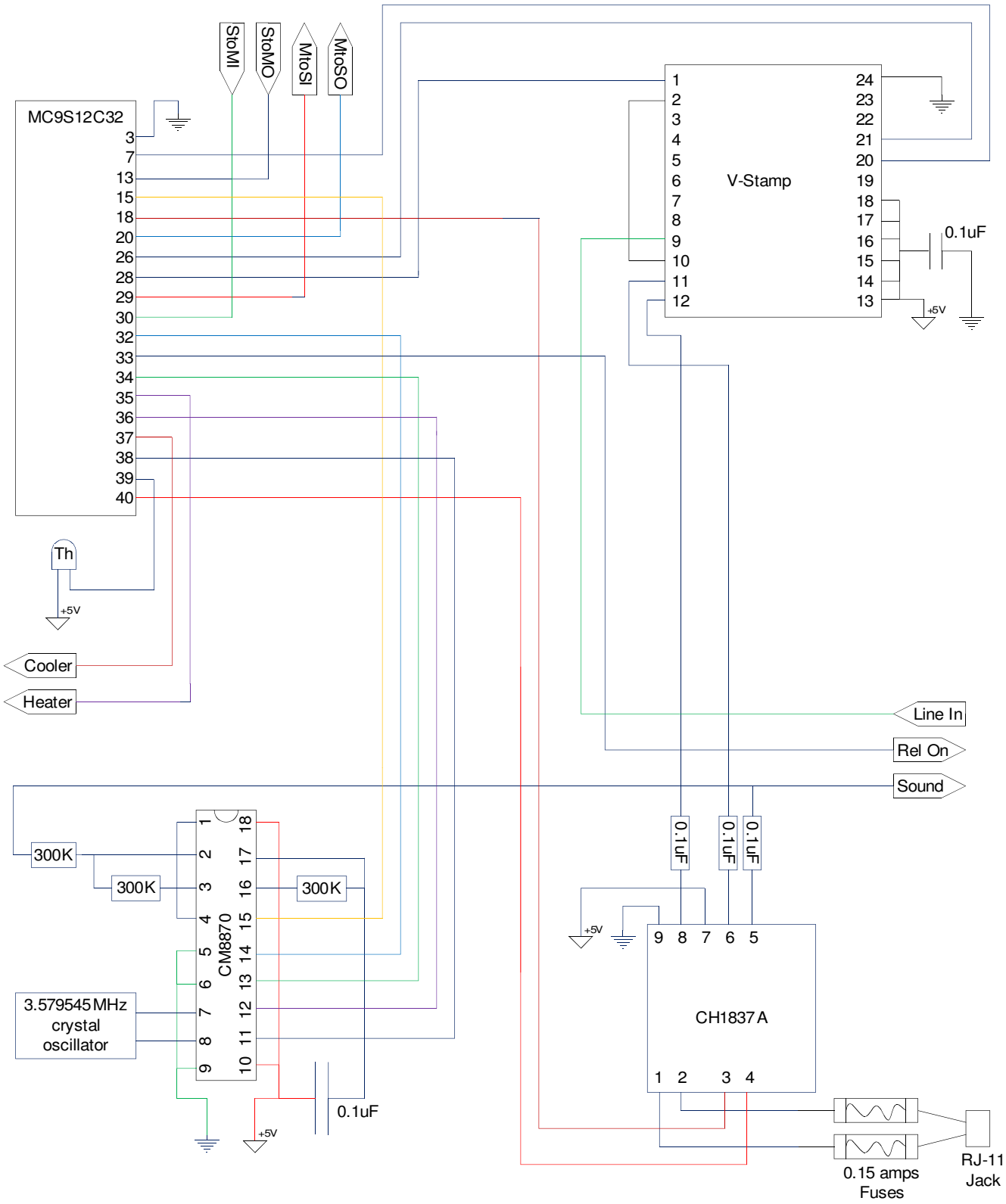
Communication between the microcontrollers failed after moving the project from the breadboards to a solder board (done for aesthetics). This communication failure was fixed by simply hooking up the V-Stamp text-to-speech chip. We didn't realize that not connecting the V-Stamp chip left the master microcontroller in an infinite loop. The time lost fixing this simple problem should have been used to finalize full code coverage. The code shown at the presentation most likely would have been bug free if this problem had been remedied quicker.

Chapter 7 Conclusion

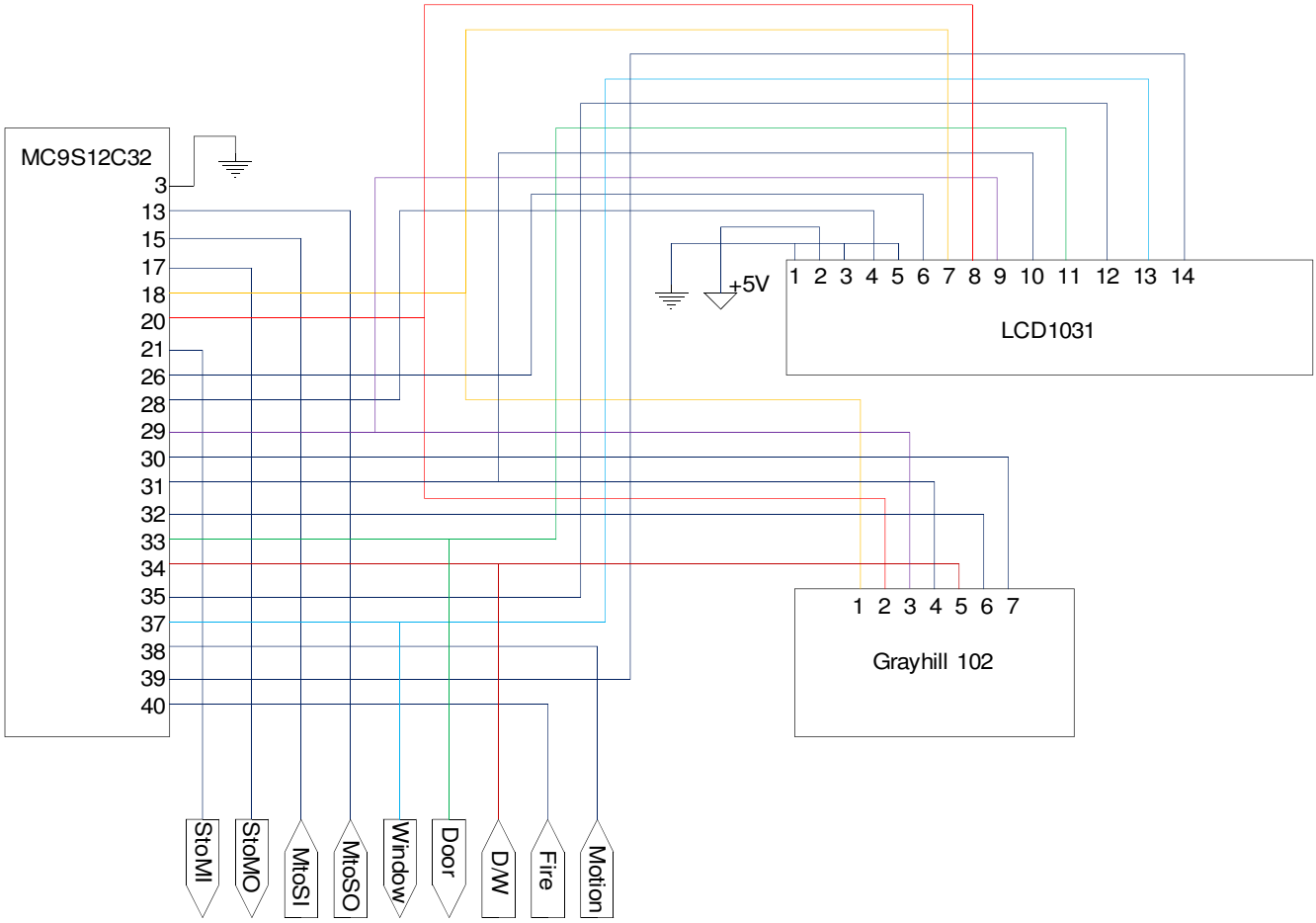
Home security and home automation system controlled by a cell phone is possible for any home owner and cell phone user. One of our key objectives was to engineer a system that would be affordable to every home owner and not incur a monthly subscription cost. We believe that a system without a subscription, that is otherwise equal to a modern security system is a much better value to the average consumer. While we acknowledge that many similar systems exist, the ability to interface with the home security system remotely via a phone-line is the most unique and valuable feature of the system described here.

Our final system successfully interfaced two microcontrollers to give the owner full control of the home security system both on-site and remotely. We did add in the functional thermostat with the thermistor, but failed to add controlling menus and options for much home automation before the project deadline. Home automation was in our initial sales pitch of the item, but not included as a base goal of the original package. We continue to maintain that if viable home automation features are added as luxuries to the base system, it would be even more attractive to consumers. However the goal of making this project was not to have a marketable product, but rather to engineer something that would prove useful and/or improve the quality of life for many.

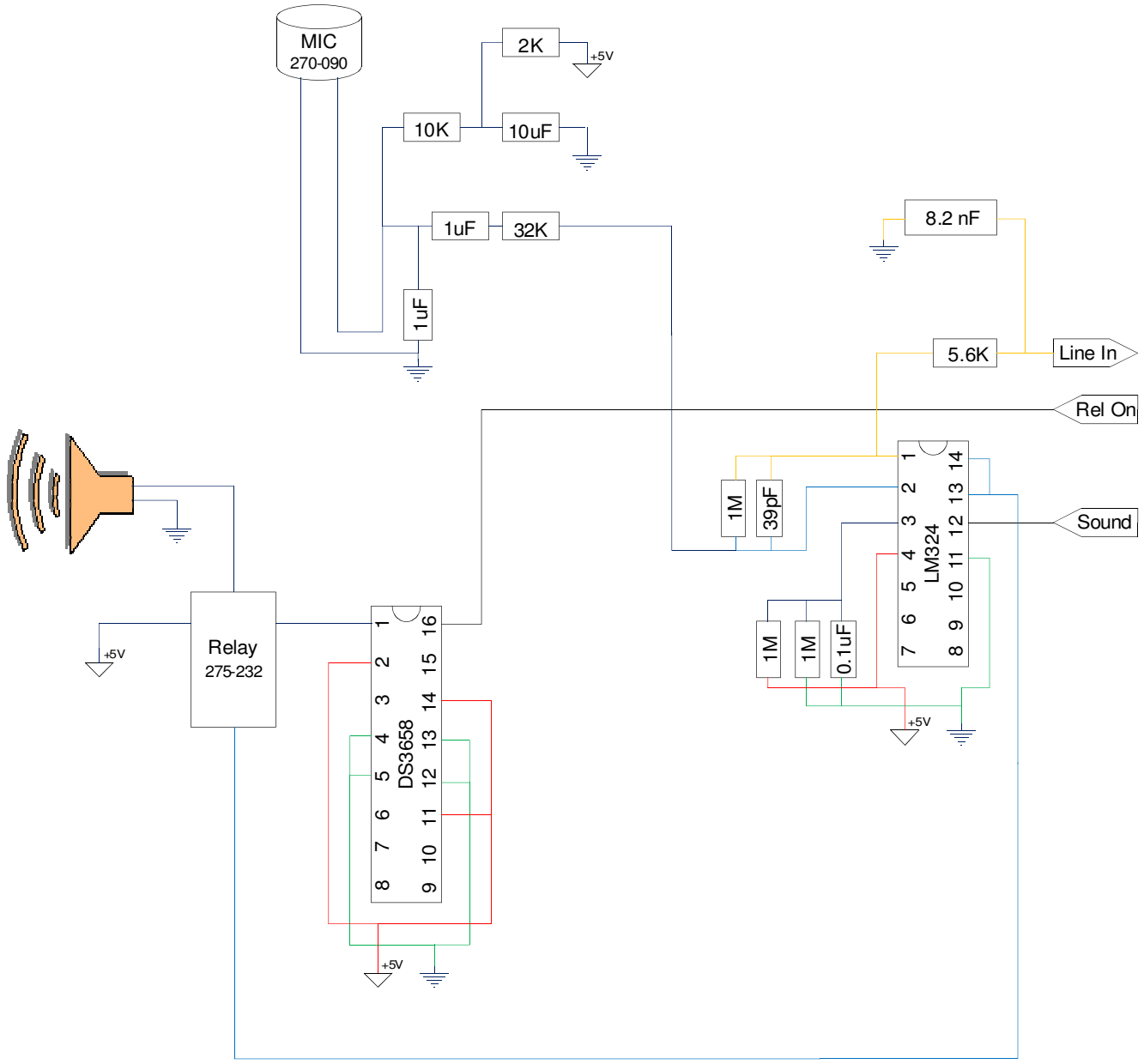
Master Schematic



Slave Schematic



Amplifiers


MIC
270-090
2K
+5V
10K
10uF
1uF
32K
1uF
8.2 nF
5.6K
Line In
Rel On
Sound
1M
39pF
LM324
1M
1M
0.1uF
+5V
Relay
275-232
+5V
DS3658
+5V

```
//------------------------------- main.c ------------------------------------------
  // Master
  #include <hidef.h>      /* common defines and macros */
  #include <mc9s12c32.h>     /* derivative information */
  #pragma LINK_INFO DERIVATIVE “mc9s12c32”
  #include “stdafx.h”


  extern bool isOnline;
  extern bool isRinging;
  extern bool replyArrived;
  extern byte curMenu;
  extern byte lastMenu;
  extern byte lastKeypadState;
  extern byte menuInteractionTable[TOTAL_MENUS][6];
  extern Menu menus[TOTAL_MENUS];
  extern byte keypadBuffer[KEYPAD_BUFFER_SIZE];
  extern byte backupKeypadBuffer[KEYPAD_BUFFER_SIZE];


  // Saved messages
  bool keypadAlert = false;
  bool doorAlert = false;
  bool windowAlert = false;
  bool motionAlert = false;
  bool smokeAlert = false;
  bool adminLocked = false;


  // User controls
  bool heaterState = false;
  bool coolerState = false;
  bool sirenState = false;
  bool armedState = false;
  bool speakerState = false;
  byte totalPhoneNumbers = 2;


  // State variables
  bool sentRingMsg = false;
  byte lastNumberCalled = EMPTY;


  // Global array indexs
  byte phoneNumberArrayIndex = 0;


  // Temperature variables
  byte temperature = 70;
  byte minTemperature = 60;
  byte maxTemperature = 80;
```

```
//---------------------------------------------------------------------
   void main(void) {
   EnableInterrupts;
   InitializePorts();
   InitializeMenus();
   ADC_Init();


   while(true){
   ControlTemp();
   ProcessMessages();
   // Do state effects
   if(curMenu != OFFLINE_MENU) // phone is on
         {
   UpdateKeypad();
   ProcessKeypadInput();
         }
   else // phone is off
         {
   if(isRinging && !sentRingMsg)
                  {
   sentRingMsg = true;
   Send(MO_PHONE_RANG); // Phone will be picked up via MI_PICKUP in ProcessMessages() after
   timer expires
                  }

         }

  }

}
//---------------------------------------------------------------------
   void interrupt 8 OuterInterrupt(void){  // PT0
   Outer();
   TFLG1 |= 0x01; // ack, clear C0F: This makes interrrupt 8 on PT0 ready to interrupt again.
}
   void interrupt 10 InnerInterrupt(void){ // PT2
   Inner();
   TFLG1 |= 0x04; // ack, clear C1F: This makes interrrupt 10  on PT2 ready to interrupt again.
}
   void interrupt 15 RingIndication(void){
   isRinging = true;
   TFLG1 |= 0x80; // re-enable the phone ringing interrupt
}
```

```
//-------------------------------------------------------------------
   // Tempature stuff
   void ADC_Init(void){
   ATDCTL2 = 0x80; // enable ADC
   ATDCTL3 = 0; // no freeze
   ATDCTL4 = 0x1f; // sample time slowest
}


   unsigned char GetTemp(void){
   volatile unsigned char ADCInput = 23;
   ATDCTL5 ^= 0x87; // right justification channel 7
   while((ATDSTAT1 & 0x80) == 0){}; // wait for CCF7
   ADCInput = ATDDR7;
   ADCInput = 103-(ADCInput/4);//100 - (ADCInput/4);//(200 – ADCInput);  // 67
   if (ADCInput > 102);//temperature = 2;
   else if (ADCInput < 50);//temperature = 1;
   else
   temperature = ADCInput;
   return temperature-11;
}


   void TurnHeater(bool value){
   if(value){
   PTAD |= 0x20;
  }
   else{
   PTAD &= 0xDF;
   TurnCooler(false);
  }
}


void TurnCooler(bool value){
   if(value){
   PTAD |= 0x40;
  }
   else{
   PTAD &= 0xBF;
   TurnHeater(false);
  }
}
```

```
void ControlTemp(void){
    GetTemp();
    if (GetTemp() >= maxTemperature && coolerState){
    TurnCooler(true);
   }
    else if (GetTemp() <= minTemperature && heaterState){
    TurnHeater(true);
   }
    else{
    TurnCooler(false);
    TurnHeater(false);
   }
 }
 //-----------------------------------------------------------------
void ProcessMessages(void){
    // Use break if you're done executing the message
    // Use return to preserve what's in the input buffer. Using
    // break will guarantee an EMPTY message before returning.

    byte msg;

    while(true)
   {
    // Preempt the Pull() for a couple cases
    if(Peek(0)==MI_NO){
    replyArrived = true;
    return;
     }
    if(Peek(0)==MI_YES){

    replyArrived = true;
    return;
     }



    msg = Pull(); // Get next msg
    switch(msg){
    case(EMPTY):
    return;

    case(MI_PICKUP):
    sentRingMsg = false;
```

```
JumpToExactMenu(5);
break;

case(MI_HERE_COMES_A_NUMBER):
while(Peek(9)==EMPTY); // wait for 10 digit phone number to follow
replyArrived = true; // the number is here
return;

case(MI_HERE_COMES_PICKUP):
while(Peek(1) != MI_DONE && Peek(2) != MI_DONE);
replyArrived = true;
return;

case(MI_HERE_COMES_A_SENSOR):
while(Peek(1) != MI_DONE && Peek(2) != MI_DONE);
replyArrived = true;
return;

// All these alerts do the same logic
case(MI_KEYPAD_ALERT):
keypadAlert = true;
JumpToExactMenu(0);
break;

case(MI_SMOKE):
smokeAlert = true;
JumpToExactMenu(0);
break;

case(MI_DOOR):
doorAlert = true;
JumpToExactMenu(0);
break;

case(MI_MOTION):
motionAlert = true;
JumpToExactMenu(0);
break;

case(MI_WINDOW):
windowAlert = true;
JumpToExactMenu(0);
break;

case(MI_ADMIN_LOCKOUT):
adminLocked = true;
```

```
break;

case(MI_ADMIN_UNLOCK):
adminLocked = false;
break;

case(MI_CANCEL_SMOKE):
smokeAlert = false;
JumpToExactMenu(4);// the goodbye menu, falls through to offline
break;

case(MI_GET_TEMP):
Send(MO_HERE_COMES_A_TEMP);
GetTemp();
Send(GetTemp());
break;

case(MI_TOGGLE_HEATER):
TurnHeater(!heaterState);
break;

case(MI_TOGGLE_COOLER):
TurnCooler(!coolerState);
break;

case(MI_GET_HEATER_STATE):
Send(MO_HERE_COMES_A_TEMP);
Send(heaterState);
break;

case(MI_GET_COOLER_STATE):
Send(MO_HERE_COMES_A_TEMP);
Send(coolerState);
break;

case(MI_GET_MIN_TEMP):
Send(MO_HERE_COMES_A_TEMP);
Send(minTemperature);
break;

case(MI_GET_MAX_TEMP):
Send(MO_HERE_COMES_A_TEMP);
Send(maxTemperature);
break;

case(MI_HERE_COMES_MIN_TEMP):
```

```
   while(Peek(0)==EMPTY);
   minTemperature = Pull();
   break;


   case(MI_HERE_COMES_MAX_TEMP):
   while(Peek(0)==EMPTY);
   maxTemperature = Pull();
   break;
    }

  }

}

//-----------------------------------------------------------------
   void JumpToExactMenu(byte nextMenu){
   lastMenu = curMenu;
   curMenu = nextMenu;
   JumpToMenu(EFFECTS_ONLY);
}


   void JumpToMenu(byte nextMenu){
   byte I;
   byte tempBuff[KEYPAD_BUFFER_SIZE] = {EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY,
   EMPTY, EMPTY, EMPTY, EMPTY};


   if(nextMenu!=EFFECTS_ONLY)
   lastMenu = curMenu;


   // First process the buffer to determine next menu
   if(nextMenu==PROCESS_KEYPAD_BUFFER)
  {
   if(curMenu==5 || curMenu==6 || curMenu==7){
   if(!IsValidPassword()) nextMenu = INVALID_MENU;
   else{
   Send(MO_IS_ANY_PASSWORD);
   I=0;
   while(keypadBuffer[I++] != EMPTY) Send(keypadBuffer[I]);
   Send(MO_DONE);
   WaitReply();
   if(Pull()==MI_YES)
   nextMenu = CORRECT_MENU;
   else
   nextMenu = INCORRECT_MENU;
     }

    }
   else if(curMenu==12 || curMenu==13 || curMenu==14){
```

```
if(!IsValidPassword())
nextMenu = INVALID_MENU;
else{
Send(MO_IS_ADMIN_PASSWORD);
I=0;
while(keypadBuffer[I++] != EMPTY) Send(keypadBuffer[I]);
Send(MO_DONE);
WaitReply();
if(Pull()==MI_YES)
nextMenu = CORRECT_MENU;
else
nextMenu = INCORRECT_MENU;
  }
 }
else if(curMenu==17){
if(totalPhoneNumbers == PHONE_NUMBER_BUFFER_SIZE)
nextMenu = INCORRECT_MENU;
else if(!IsValidPassword())
nextMenu = INVALID_MENU;
else{
nextMenu = CORRECT_MENU;
  }
 }
else if(curMenu==23){
if(!IsValidPassword())
nextMenu = INVALID_MENU;
else{
nextMenu = CORRECT_MENU;
BackupKeypadBuffer();
  }
 }
else if(curMenu==25){
if(!Equals(keypadBuffer, backupKeypadBuffer))
nextMenu = INCORRECT_MENU;
else{
Send(MO_NEW_ADMIN_PASSWORD);
I=0;
while(keypadBuffer[I++] != EMPTY) Send(keypadBuffer[I]);
Send(MO_DONE);
nextMenu = CORRECT_MENU;
  }
 }
else if(curMenu==28){
if(keypadBuffer[6]==EMPTY || keypadBuffer[0]==0)
nextMenu = INVALID_MENU;
```

```
 }
else if(curMenu==43){
if(keypadBuffer[0]==EMPTY || keypadBuffer[2]!=EMPTY)
nextMenu = INVALID_MENU;
else{
minTemperature = keypadBuffer[0];
if(keypadBuffer[1] != EMPTY)
minTemperature += keypadBuffer[1] * 10;
  }


 }
else if(curMenu==44){
if(keypadBuffer[0]==EMPTY || keypadBuffer[2]!=EMPTY)
nextMenu = INVALID_MENU;
else{
maxTemperature = keypadBuffer[0];
if(keypadBuffer[1] != EMPTY)
maxTemperature += keypadBuffer[1] * 10;
  }

 }
else if(curMenu==47){
if(keypadBuffer[0]==EMPTY || keypadBuffer[2]!=EMPTY)
nextMenu = INVALID_MENU;
else{
I = keypadBuffer[0];
if(keypadBuffer[1] != EMPTY)
I += keypadBuffer[1] * 10;
Send(MO_NEW_PICKUP_DELAY);
Send(I);
  }

 }
else if(curMenu==48){
if(keypadBuffer[0]==EMPTY || keypadBuffer[2]!=EMPTY)
nextMenu = INVALID_MENU;
else{
I = keypadBuffer[0];
if(keypadBuffer[1] != EMPTY)
I += keypadBuffer[1] * 10;
Send(MO_NEW_SENSOR_DELAY);
Send(I);
  }

 }

}
```

```
    // All menus now have a straightforward lookup
    if(nextMenu != EFFECTS_ONLY)
    curMenu = menuInteractionTable[curMenu][nextMenu];

    // All special case effects are handled here
  //
    // First handle simple offline/online interaction and general case functionality like speaking the message
  //
    if(curMenu==OFFLINE_MENU && lastMenu!=OFFLINE_MENU)
   {
    Hangup();
   }
    else if(curMenu!=OFFLINE_MENU && lastMenu==OFFLINE_MENU)
   {
    Pickup();
    if(curMenu == 0) // Picked up to make a call
     {
    if(lastNumberCalled == EMPTY)
       {
    lastNumberCalled = 0;
    CallNumber(0);
       }
    else
       {
    lastNumberCalled++;
    if(lastNumberCalled < totalPhoneNumbers)
    CallNumber(lastNumberCalled);
    else
    lastNumberCalled = EMPTY;
       }
     }
   }
    if(curMenu!=OFFLINE_MENU)
   {
    Speak(menus[curMenu].message, menus[curMenu].length);
   }

  //
 // Begin menu specific special cases
```

```
//
        //--1--
  if(curMenu==1){
  if(keypadAlert) Speak(”kee pad”, 7);
  if(doorAlert) Speak(”frunt dor sensor”, 15);
  if(windowAlert) Speak(”wind o sensor”, 13);
  if(motionAlert) Speak(”mo shun sensor”, 16);
  if(smokeAlert) Speak(”smoke sensor”, 14);


  keypadAlert = false;
  doorAlert = false;
  windowAlert = false;
  motionAlert = false;
  smokeAlert = false;
 }
 //--2--
  else if(curMenu==lastMenu && curMenu==2){
  speakerState = !speakerState;
  if(speakerState) TurnSpeaker(false);
  else TurnSpeaker(true);
 }
        //--3--
  else if(curMenu==3){
  sirenState = !sirenState;
  if(sirenState) Speak(”on”, 2);
  else Speak(”awf”, 3);


 }
        //--8--
  else if(curMenu==8){
  if(armedState) Speak(”armed”, 2);
  else Speak(”uhn armed”, 5);
 }
        //--9--
  else if(curMenu==9){
  if(lastMenu==10){
  armedState = !armedState;
  if(armedState) Send(MO_ARM);
  else Send(MO_DISARM);
   }


  if(armedState) Speak(”armed”, 2);
  else Speak(”uhn armed”, 5);
```

```
}
//--15--
else if(curMenu==15){
if(adminLocked){
JumpToMenu(HIGHER_MENU);
 }
}
//--11--
else if(curMenu==11){
speakerState = !speakerState;
TurnSpeaker(speakerState);
if(speakerState) Speak("on", 2);
else Speak("awf", 3);
}
//--20--
else if(curMenu==20){
Send(MO_NEW_REGULAR_PASSWORD);
I=0;
while(keypadBuffer[I++] != EMPTY) Send(keypadBuffer[I]);
Send(MO_DONE);
ClearKeypadBuffer();

}
//--22--
else if(curMenu==22){
Send(MO_DELETE_PASSWORD);
I=0;
while(keypadBuffer[I++] != EMPTY) Send(keypadBuffer[I]);
Send(MO_DONE);
ClearKeypadBuffer();

}
//--25--
else if(curMenu==25){
BackupKeypadBuffer();
}
//--26--
else if(curMenu==26){
Send(MO_NEW_ADMIN_PASSWORD);
for(I=0; keypadBuffer[I] != EMPTY; I++)
        {
SpeakInt(backupKeypadBuffer[I]);
Speak(" ",1);
Send(backupKeypadBuffer[I]);
        }
```

```
 Send(MO_DONE);
 ClearKeypadBuffer();
}
 //--30--
 else if(curMenu==30){
 Send(MO_NEW_PHONE_NUMBER);
 I=0;
 while(keypadBuffer[I++] != EMPTY) Send(keypadBuffer[I]);
 Send(MO_DONE);
 ClearKeypadBuffer();
}
 //--31--
 else if(curMenu==31){
 Send(MO_GET_PHONE_NUMBER);
 Send(phoneNumberArrayIndex);
 WaitReply();
 while(Peek(0) != MI_DONE) SpeakInt(Pull());
 Pull(); // Burn the MI_DONE msg


}
 //--33--
 else if(curMenu==33){
 Send(MO_DELETE_PHONE_NUMBER);
 Send(phoneNumberArrayIndex);
}
 //--34--
 else if(curMenu==34){
 if(lastKeypadState == 4){
 if(phoneNumberArrayIndex > 0)
 phoneNumberArrayIndex--;
  }
 else{
 if(phoneNumberArrayIndex < totalPhoneNumbers-1)
 phoneNumberArrayIndex++;
  }
 SpeakInt(phoneNumberArrayIndex);
}
 //--35--
 else if(curMenu==35){
 GetTemp();     // Must be called twice to be accurate
 SpeakInt(GetTemp());
}
 //--36--
 else if(curMenu==36){
 heaterState = !heaterState;
```

```
  if(heaterState) Speak(”on”, 2);
  else Speak(”awf”, 3);
 }
  //--37--
  else if(curMenu==37){
  if(lastMenu==43){
  minTemperature = keypadBuffer[0];
  if(keypadBuffer[1] != EMPTY)
  minTemperature += keypadBuffer[1] * 10;
   }
  SpeakInt(minTemperature);
 }
  //--39--
  else if(curMenu==39){
  sirenState = !sirenState;
  if(sirenState) Speak(”on”, 2);
  else Speak(”awf”, 3);
 }
  //--40--
  else if(curMenu==40){
  if(lastMenu==44){
  maxTemperature = keypadBuffer[0];
  if(keypadBuffer[1] != EMPTY)
  maxTemperature += keypadBuffer[1] * 10;
   }
  SpeakInt(maxTemperature);
 }
  //--41--
  else if(curMenu==41){
  SpeakInt(minTemperature);
 }
  //--42--
  else if(curMenu==42){
  SpeakInt(maxTemperature);
 }
  //--45--
  else if(curMenu==45){
  Send(MO_GET_PICKUP_DELAY);
  WaitReply();
  SpeakInt(Pull());
 }
  //--46--
  else if(curMenu==46){
  Send(MO_GET_SENSOR_DELAY);
  WaitReply();
```

```
  SpeakInt(Pull());
 }
 //--49--
 else if(curMenu==49){
 Send(MO_NEW_PICKUP_DELAY);
 Send(keypadBuffer[0] + (keypadBuffer[1]==EMPTY) ? 0: (keypadBuffer[1]*10));
 ClearKeypadBuffer();
 Send(MO_GET_PICKUP_DELAY);
 WaitReply();
 SpeakInt(Pull());
}
 //--50--
 else if(curMenu==50){
 Send(MO_NEW_SENSOR_DELAY);
 Send(keypadBuffer[0] + (keypadBuffer[1]==EMPTY) ? 0: (keypadBuffer[1]*10));
 ClearKeypadBuffer();
 Send(MO_GET_SENSOR_DELAY);
 WaitReply();
 SpeakInt(Pull());
}
 //--51--
 else if(curMenu==51){
 if(heaterState){
 TurnCooler(true);
  }
 else if(coolerState){
 TurnCooler(false);
  }
 else{
 TurnHeater(true);
  }
 if(heaterState || coolerState) Speak("on", 2);
 else Speak("awf", 3);
}
//
 // Handle fall through menus
//
 if(menus[curMenu].fallThrough == true){
 JumpToMenu(NEXT_MENU);
}
}
//------------------------------ menu.c ------------------------------------------
 #include "stdafx.h"
```

```
Menu menus[TOTAL_MENUS];
byte curMenu = OFFLINE_MENU;
byte lastMenu = -1;
// menus map as follows:
// for non-buffered input menus: {MENU_1, MENU_2, MENU_3, MENU_4, MENU_5,
HIGHER_MENU}
// for fall-through menus:       {NEXT_MENU, ---, ---, ---, ---, ---}
// for buffered input menus:     {CORRECT, INCORRECT, INVALID, ---, ---, HIGHER_MENU}
byte menuInteractionTable[TOTAL_MENUS][6] =
{{1 ,0 ,0 ,0 ,0 ,0 },//0
{2 ,2 ,2 ,2 ,2 ,2 },//1
{2 ,3 ,4 ,2 ,2 ,2 },//2
{2 ,2 ,2 ,2 ,2 ,2 },//3
{38,4 ,4 ,4 ,4 ,4 },//4
{8 ,6 ,5 ,5 ,5 ,5 },//5
{8 ,7 ,6 ,6 ,6 ,6 },//6
{8 ,4 ,7 ,7 ,7 ,7 },//7
{10,10,10,10,10,10},//8
{10, 9, 9, 9, 9, 9},//9
{9 ,35,11,39,12,10},//10
{10,10,10,10,10,10},//11
{40,13,12,12,12,10},//12
{40,14,13,13,13,13},//13
{40, 4,14,14,14,14},//14
{16,27,38,15,15,15},//15
{17,21,23,16,16,16},//16
{20,19,17,17,17,17},//17
{23,23,23,23,23,23},//18
{16,16,16,16,16,16},//19
{16,16,16,16,16,16},//20
{22,21,21,21,21,21},//21
{16,16,16,16,16,16},//22
{25,23,23,23,23,23},//23
{23,23,23,23,23,23},//24
{26,18,18,25,25,25},//25
{16,16,16,16,16,16},//26
{28,31,27,27,27,27},//27
{30,29,29,28,28,28},//28
{27,27,27,27,27,27},//29
{27,27,27,27,27,27},//30
{32,32,32,32,32,32},//31
{33,32,32,32,34,34},//32
{27,27,27,27,27,27},//33
{27,27,27,27,27,27},//34
{36,36,36,36,36,36},//35
{51,37,40,41,42,10},//36
{35,35,35,35,35,35},//37
{45,46,38,38,38,15},//38
{10,39,39,39,39,39},//39
```

```
{35,35,35,35,35,35},//40
{43,43,43,43,43,43},//41
{44,44,44,44,44,44},//42
{37,43,43,43,43,36},//43
{40,44,44,44,44,36},//44
{47,47,47,47,47,47},//45
{48,48,48,48,48,48},//46
{49,47,47,47,47,38},//47
{50,48,48,48,48,38},//48
{38,38,38,38,38,38},//49
{38,38,38,38,38,38},//50
{36,36,36,36,36,36}};//51


void InitializeMenus(void){
// Outgoing calls
PopulateMenu(0, “this is the ottomated home security sihstem press one to accept this caal”, 73, false, false);
PopulateMenu(1, “alarm was triggered by the following sensors”, 44, true, false);
PopulateMenu(2, “to toggle the speeker press won to toggle the siren press too to disconnect and call the next number press three “, 113, false, false);
PopulateMenu(3, “the siren is “, 13, true, false);
PopulateMenu(4, “goodbye”, 7, true, false);
// Incoming calls
PopulateMenu(5, “hello this is the ottomated home security sihstem pleese enter your code”, 72, false, true);
PopulateMenu(6, “code not recognized pleese try agin”, 35, false, true);
PopulateMenu(7, “code not recognized pleese try agin”, 35, false, true);
PopulateMenu(8, “code aksepted the sihstim is “, 18, true, false);
PopulateMenu(9, “the sihstem is “, 15, true, false);
PopulateMenu(10,”mayn menu to arm or disarm press one to check temperature press two to activate speeker press three to toggle siren press for to login as admin press five”, 154, false, false);
PopulateMenu(11,”the speeker is”, 14, true, false);
PopulateMenu(12,”enter admin password”, 20, false, true);
PopulateMenu(13,”incorrect pleese try again”, 26, false, false);
PopulateMenu(14,”incorrect pleese try again”, 26, false, false);
PopulateMenu(15,”code aksepted for passwords press one for fone numbers press too for timers press three”, 87, false, false);
// Passwords
PopulateMenu(16,”to add a password press one to delete a password press too to change the admin password press three”, 99, false, false);
PopulateMenu(17,”enter the password”, 18, false, true);
PopulateMenu(19,”sorry the password buffer is full”, 33, true, false);
PopulateMenu(20,”password added”, 14, true, false);
PopulateMenu(21,”to delete this password press one for next password press to for previous password press three”, 94, false, false);
PopulateMenu(22,”password deleted”, 16, true, false);
PopulateMenu(23,”enter the new admin password”, 28, false, true);
PopulateMenu(24,”invalid password”, 16, true, false);
PopulateMenu(25,”pleese confirm the new admin password “, 38, false, true);
PopulateMenu(26,”the admin password has been changed to “, 39, true, false);
```

```
  PopulateMenu(18,”those passwords do not match”, 28, true, false);
  // Phone Numbers
  PopulateMenu(27,”to add a number press one to edit the number list press to “, 59, false, false);
  PopulateMenu(28,”enter the number to add”, 23, false, true);
  PopulateMenu(29,”that number is invalid “, 23, true, false);
  PopulateMenu(30,”added the number at lowest priority”, 35, true, false);
  PopulateMenu(31,”current number is “, 18, false, false);
  PopulateMenu(32,”to delete press one to hear next number press to for previous number press three to
  raze current numbers priority press for to lower priority press five”, 152, false, false);
  PopulateMenu(33,”number deleted”, 14, true, false);
  PopulateMenu(34,”number is now in slot “, 22, true, false);
  // Thermostat
  PopulateMenu(35,”the current temperature is “, 27, true, false);
  PopulateMenu(36,”to toggle ottomatic controls press won to turn on heeter press to to turn on air
  condishahner press three to adjust mimimum temperature press for to adjust maximum temperature press
  five”, 186, false, false);
  PopulateMenu(37,”heeter will turn on at “, 23, true, false);
  PopulateMenu(40,”air condishahner will turn on at “, 33, true, false);
  PopulateMenu(41,”minimum is currently “, 21, true, false);
  PopulateMenu(42,”maximum is currently “, 21, true, false);
  PopulateMenu(43,”enter new minimum “, 18, false, true);
  PopulateMenu(44,”enter new maximum “, 18, false, true);
  // Timers
  PopulateMenu(38,”for fone timer press one for sensor timer press to “, 51, false, false);
  PopulateMenu(39,”the siren is “, 13, true, false);
  PopulateMenu(45,”fone pickup delay is “, 21, true, false);
  PopulateMenu(46,”senser trigger delay is “, 24, true, false);
  PopulateMenu(47,”pleez enter new fone pickup delay”, 33, false, true);
  PopulateMenu(48,”pleez enter new senser trigger delay”, 36, false, true);
  PopulateMenu(49,”new fone pickup delay is “, 25, true, false);
  PopulateMenu(50,”new senser trigger delay is “, 28, true, false);
  PopulateMenu(51,”ottomatic controls are now “, 27, true, false);
}


  void PopulateMenu(byte menuIndex, char* message, byte length, bool fallThrough, bool bufferedInput){
  menus[menuIndex].message = message;
  menus[menuIndex].length = length;
  menus[menuIndex].fallThrough = fallThrough;
  menus[menuIndex].bufferedInput = bufferedInput;
}
//------------------------------ stdafx.c ----------------------------------------


  #include “stdafx.h”


  extern byte keypadBuffer[KEYPAD_BUFFER_SIZE];


  // misc functions
```

```
   void delay(unsigned long ms) {
   unsigned long I;
   for (I=0;I < ms; I++) {
   asm {
   PSHX
   LDX #$640
   Loop:
   NOP
   NOP
   DBNE X, Loop
   PULX
    }
 }
}
//---------------------------------------------------------------
   void InitializePorts(void){
   asm sei
   DDRAD = DDRAD_Button_Mask_Master;
   DDRT = DDRT_Button_Mask_Master;
   ATDDIEN = 0x7F; // PTAD7 anolog input

   PERT = 0x80;
   PPST = 0x00;

   TIE |= 0x85; // Arm OC0  on PT0, Arm OC2  on PT2, &  Arm OC7  on PT7   pg 279
   TCTL4 = 0x11; // rising edge on PT0 and PT2        pg 279
   TCTL3 = 0x80; // rising edge on PT7
   TFLG1 = 0x85; // clears interrupt PT0, PT1, and PT7 ready for the next interrupt
   TSCR1 = 0x80; // enable TEN

   DDRM |= 0x05; // 1-output, 0-input
   SCIBD = 26;  // 9600 Bits per Sec
   SCICR1 = 0x00; // No parity
   SCICR2 = 0x2C; // enable, arm RDRF

   TtS_Init();
   asm cli
}
//---------------------------------------------------------------
   void TurnMic(bool value){
   if(value) PTM |= 0x04; // turn on microphone
   else PTM &= 0xFB; // turn off microphone
}
//---------------------------------------------------------------
```

```
  void TurnSpeaker(bool value){
  if(value) PTAD |= 0x10;
  else PTAD &= 0xEF;
}

//-----------------------------------------------------------------

  bool IsValidPassword(void){
  if(keypadBuffer[3]==EMPTY) // not >3 characters
  return false;
  if(keypadBuffer[PASSWORD_LENGTH] != EMPTY) // not <max password length
  return false;
  return true;
}

//-----------------------------------------------------------------

  bool Equals(byte* a, byte* b){
  // Compares password buffers
  byte I=0;
  for(;I<KEYPAD_BUFFER_SIZE;I++)
  if(a[I]!=b[I]) return false;
  return true;
}



//------------------------------ stdafx.h ----------------------------------------

  #include <hidef.h>      /* common defines and macros */
  #include <mc9s12c32.h>     // derivative information


  // Incoming Message Codes
  #define MI_PICKUP 1
  #define MI_NO 2 //Should be followed by the Question that was asked with Incoming Message encoding
  #define MI_YES 3//Should be followed by the Question that was asked with Incoming Message encoding
  #define MI_KEYPAD_ALERT 4
  #define MI_HERE_COMES_A_NUMBER 5
  #define MI_HERE_COMES_PICKUP 6
  #define MI_DONE 7
  #define MI_SMOKE 8
  #define MI_DOOR 9
  #define MI_MOTION 10
  #define MI_WINDOW 11
  #define MI_ADMIN_LOCKOUT 12
  #define MI_ADMIN_UNLOCK 13
  #define MI_HERE_COMES_A_SENSOR 14
  #define MI_CANCEL_SMOKE 15
  #define MI_GET_TEMP 16
  #define MI_TOGGLE_HEATER 17
  #define MI_TOGGLE_COOLER 18
  #define MI_GET_HEATER_STATE 19
  #define MI_GET_COOLER_STATE 20
```

```
#define MI_GET_MIN_TEMP 21
#define MI_GET_MAX_TEMP 22
#define MI_HERE_COMES_MIN_TEMP 23
#define MI_HERE_COMES_MAX_TEMP 24


// Outgoing Message Codes
#define MO_NO 0
#define MO_YES 1
#define MO_ARM 2
#define MO_DISARM 3
#define MO_LOCKDOWN 4
#define MO_IS_ADMIN_PASSWORD 5
#define MO_IS_ANY_PASSWORD 6
#define MO_GET_PICKUP_DELAY 7
#define MO_NEW_ADMIN_PASSWORD 8
#define MO_NEW_REGULAR_PASSWORD 9
#define MO_NEW_PICKUP_DELAY 10
#define MO_PHONE_RANG 11
#define MO_GET_PHONE_NUMBER 12  // THIS VALUE MUST STAY ABOVE 9
#define MO_GET_SENSOR_DELAY 13
#define MO_NEW_SENSOR_DELAY 14
#define MO_DONE 15
#define MO_HERE_COMES_A_TEMP 16
#define MO_DELETE_PASSWORD 17
#define MO_NEW_PHONE_NUMBER 18
#define MO_DELETE_PHONE_NUMBER 19


// Master State Codes
#define DEFAULT_STATE 0
#define KEYPAD_ALERT_CALL 1
#define KEYPAD_ALERT_SPEAK 2
#define LISTENING_FOR_INPUT 3
#define SPEAK_MAIN_MENU 4


// Magic Numbers
#define INPUT_BUFFER_SIZE 40
#define KEYPAD_BUFFER_SIZE 10
#define TOTAL_MENUS 52
#define PASSWORD_LENGTH 8
#define PASSWORD_BUFFER_SIZE 10 // from Slave's Buffer, only for reference
#define PHONE_NUMBER_BUFFER_SIZE 10


// Menu Indices
#define EMPTY 0xFF
#define OFFLINE_MENU 38
#define CORRECT_MENU 0
#define INCORRECT_MENU 1
#define INVALID_MENU 2
#define PROCESS_KEYPAD_BUFFER TOTAL_MENUS
```

```
#define EFFECTS_ONLY PROCESS_KEYPAD_BUFFER+1
#define NEXT_MENU 0
#define HIGHER_MENU 5
#define MENU_0 EMPTY
#define MENU_1 0
#define MENU_2 1
#define MENU_3 2
#define MENU_4 3
#define MENU_5 4
#define MENU_6 5
#define MENU_7 EMPTY
#define MENU_8 EMPTY
#define MENU_9 EMPTY
#define MENU_POUND EMPTY

//Keypad buttons
#define KP_0 10
#define KP_STAR 11
#define KP_POUND 12

// Port masks
#define DDRM_Button_Mask_Master 0xFF // outputs = 1, inputs = 0
#define DDRAD_Button_Mask_Master 0x7F // outputs = 1, inputs = 0
#define DDRT_Button_Mask_Master 0x00 // outputs = 1, inputs = 0

// bool definition
typedef unsigned char bool;
#define true (0==0)
#define false !true

// Menu definition
typedef struct{
char* message;
byte length;
bool fallThrough;
bool bufferedInput;
} Menu;

// Tempature
extern void ADC_Init(void);
extern unsigned char GetTemp(void);
extern void TurnHeater(bool);
extern void TurnCooler(bool);
extern void ControlTemp(void);
// all functions
extern void delay(unsigned long ms);
extern void InitializePorts(void);
extern void InitializeMenus(void);
```

```
    extern void ProcessMessages(void);
    extern void TurnMic(bool);
    extern void TurnSpeaker(bool);
    extern bool IsValidPassword(void);
    extern bool Equals(byte* a, byte* b);
    // Menus
    extern void JumpToExactMenu(byte nextMenu);
    extern void JumpToMenu(byte nextMenu);
    extern void PopulateMenu(byte menuIndex, char* message, byte length, bool fallThrough, bool
    bufferedInput);
    // Keypad
    extern void BufferKeypress(byte keypress);
    extern void BackupKeypadBuffer(void);
    extern void UpdateKeypad(void);
    extern void ProcessKeypadInput(void);
    extern void ClearKeypadBuffer(void);
    // Messages
    extern void Push(byte);
    extern byte Pull(void);
    extern void Send(byte);
    extern void Inner(void);
    extern void Outer(void);
    extern byte Peek(byte);
    extern void WaitReply(void);
    extern byte GetMessageInt(void);
    // DAA
    extern void Pickup(void);
    extern void Hangup(void);
    extern void DialNumber(byte number);
    extern void CallNumber(byte phoneNumberBufferIndex);
    // Vstamp
    extern void TtS_Init(void);
    extern void TtS_Out(char letter);
    extern void Speak(char* sentence, byte length);
    extern void SpeakInt(byte sayMe);
 //------------------------------ messages.c ------------------------------------------------------
#include “stdafx.h”
#include “Buffer.h”

byte outputBuffer[OUTPUT_BUFFER_SIZE] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

void Push(byte addMe)
 {
    byte I;
    for(I=0;I<OUTPUT_BUFFER_SIZE;I++)
      {
        if(outputBuffer[I] == 0)
```

```
        {
            outputBuffer[I] = addMe;
            return;
        }
      }
  }

byte Pull(void)
 {
    byte I;
    byte returnValue = outputBuffer[0];
    // pull messages off output buffer
    if(returnValue != 0)
   {
    // Shift buffer up a slot
    for(I=1;I<OUTPUT_BUFFER_SIZE;I++)
      {
        outputBuffer[I-1] = outputBuffer[I];
      }
    outputBuffer[OUTPUT_BUFFER_SIZE-1] = 0;
   }
    return returnValue;
 }
 //------------------------------ keypad.c ------------------------------------------------------------
    #include “stdafx.h”

    byte keypadBuffer[KEYPAD_BUFFER_SIZE] = {EMPTY, EMPTY, EMPTY, EMPTY, EMPTY,
    EMPTY, EMPTY, EMPTY, EMPTY, EMPTY};
    byte backupKeypadBuffer[KEYPAD_BUFFER_SIZE] = {EMPTY, EMPTY, EMPTY, EMPTY,
    EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY};
    byte lastKeypadState=EMPTY;
    byte curKeypadState=EMPTY;
    extern Menu menus[TOTAL_MENUS];
    extern byte curMenu;

    void UpdateKeypad(void){
    lastKeypadState = curKeypadState;
    if(PTT & 0x02){
    // Enable on PTAD[3]
    curKeypadState = PTT & 0x78;
    curKeypadState>>=3;
   }
```

```
    else
    curKeypadState = EMPTY;
  }


void ProcessKeypadInput(void){
    if(curKeypadState == EMPTY && lastKeypadState != EMPTY)
   {
    if(menus[curMenu].bufferedInput && !menus[curMenu].fallThrough)//second condition should never be
    checked
     {
    if(lastKeypadState==KP_STAR)
      {
            if(keypadBuffer[0]==EMPTY)
               JumpToMenu(HIGHER_MENU);
            else
               ClearKeypadBuffer();
      }
    else if(lastKeypadState==KP_POUND)
      {
            JumpToMenu(PROCESS_KEYPAD_BUFFER);
      }
    else
            BufferKeypress(lastKeypadState);
     }
    else if(!menus[curMenu].fallThrough)// not Buffered Input or a fall through menu
     {
            TtS_Init(); // Stop Vstamp from finishing old menus
            if(lastKeypadState==11) JumpToMenu(HIGHER_MENU);
            else if(lastKeypadState==1) JumpToMenu(MENU_1);
            else if(lastKeypadState==2) JumpToMenu(MENU_2);
            else if(lastKeypadState==3) JumpToMenu(MENU_3);
            else if(lastKeypadState==4) JumpToMenu(MENU_4);
            else if(lastKeypadState==5) JumpToMenu(MENU_5);
          }
    }  // Ignore keypad input on fall through menus
  }


void BufferKeypress(byte keypress){
    int I=0;
    for(I; I<KEYPAD_BUFFER_SIZE && keypadBuffer[I]!=EMPTY; I++);
    if(I != KEYPAD_BUFFER_SIZE)
    keypadBuffer[I] = keypress;
```

```
}

void ClearKeypadBuffer(void){
    int I=0;
    for(;I<KEYPAD_BUFFER_SIZE;I++)
    keypadBuffer[I] = EMPTY;
}

void BackupKeypadBuffer(void){
    int I;
    for(I=0; I<KEYPAD_BUFFER_SIZE; I++)
    backupKeypadBuffer[I] = keypadBuffer[I];
}
//------------------------------- daa.c ---------------------------------------------------------
    #include “stdafx.h”

    extern byte curMenu;
    // Message passing globals
    extern bool waitingForReply;
    extern bool replyArrived;
    extern byte reply[10];

    bool isOnline = false;
    bool isRinging = false;
//-------------------------------------------------------------------------
    void Pickup(void){
    PTAD |= 0x01; // Causes the phone ringing interrupt bit to get set
    delay(500);
    isRinging = false;
    isOnline = true;
    if(curMenu != 5) // if not taking an incoming call
    TurnMic(true);
}
//-------------------------------------------------------------------------
    void Hangup(void){
    isRinging = false;
    delay(100);
    PTAD &= 0xFE; // Causes the phone ringing interrupt bit to get set
    delay(1);
    TurnMic(false);
    TurnSpeaker(false);
    isOnline = false;
}
//-------------------------------------------------------------------------
```

```
    void DialNumber(byte number){
    if(number==EMPTY) return;
    switch(number){
    case(0):
    TtS_Out(0x01);
    Speak("0*",2);
    break;
    case(1):
    TtS_Out(0x01);
    Speak("1*",2);
    break;
    case(2):
    TtS_Out(0x01);
    Speak("2*",2);
    break;
    case(3):
    TtS_Out(0x01);
    Speak("3*",2);
    break;
    case(4):
    TtS_Out(0x01);
    Speak("4*",2);
    break;
    case(5):
    TtS_Out(0x01);
    Speak("5*",2);
    break;
    case(6):
    TtS_Out(0x01);
    Speak("6*",2);
    break;
    case(7):
    TtS_Out(0x01);
    Speak("7*",2);
    break;
    case(8):
    TtS_Out(0x01);
    Speak("8*",2);
    break;
    case(9):
    TtS_Out(0x01);
    Speak("9*",2);
    break;
   }
 }
 //---------------------------------------------------------------------
void CallNumber(byte phoneNumberBufferIndex){
    byte I;
    replyArrived = false;
```

```
    Send(MO_GET_PHONE_NUMBER);
    Send(phoneNumberBufferIndex);
    while(!replyArrived) ProcessMessages(); // Wait for entire reply, 10 more packets
    // buffer[9:0] now holds the phoneNumber we want
    while(Peek(0)==EMPTY);
    if(Peek(0)==MO_GET_PHONE_NUMBER){ // The last message must have been “NO”, followed by the
    question we asked
    Pull();
   }
    else{
    for(I=0;I<10;I++) {
    DialNumber(Pull());
    TtS_Out(0x80);
    TurnMic(true);
     }
   }
    replyArrived = false;
 }
 //------------------------------ vstamp.c ------------------------------------------------------------
    #include “stdafx.h”
    /* PTM Text to Speech V-Stamp
    0-RES# Reset, needs to be low for atleast 1ms after power up.
    1-CTS# Low is ready to accept data
    RSD  data in chars is received
    */
    // Initailizes PTM for V-Stamp Text to speech
    void TtS_Init(void){// take out port M init
    PTM &= 0xFE; // Make sure reset is low
    delay(4); // delay after power up for RES#
    PTM |= 0x01; // RES# goes high ready for input
    delay(4); // delay for data to be sent to V-Stamp
    TtS_Out(’U’); // Send to set up clock rate in V-Stamp incoming data U = 01010101
    delay(4);
    TtS_Out(0x01);
    Speak(”10O”,3); // grechen voice
    TtS_Out(0x01);
    Speak(”3T”,2);  // delay between words
    delay(4);
    TtS_Out(0x01);
    Speak(”6S”,2);  // reading speed
    delay(4);
}

void TtS_Out(char letter){
    while (PTM & 0x02); // CTS# wait until ready for more to be sent as letters or comands
    while(!(SCISR1&0x80));
```

```
    SCIDRL = letter;
  }
void Speak(char* sentence, byte length){
    byte I;
    PTM |= 0x04;
    for(I=0; I<length; I++) {
    TtS_Out(sentence[I]);
   }
    TtS_Out(0x80);
  }
void SpeakInt(byte sayMe){
    if(sayMe > 10)
    TtS_Out(48 + sayMe/10);
    TtS_Out(48 + sayMe%10);
    TtS_Out(0x80);
  }
```

```
//---------------------------------- main.c -------------------------------------------------------
// Slave
#include <hidef.h>      /* common defines and macros */
#include <mc9s12c32.h>     // derivative information
#pragma LINK_INFO DERIVATIVE “mc9s12c32”
#include “stdafx.h”

// External variables
extern short newKeypadState; // For detecting new keypresses
extern byte menuInteractionTable[TOTAL_MENUS][4];
extern Menu menus[TOTAL_MENUS];
// Timer variables
extern bool countingSensorTimer;
extern bool countingPhoneTimer;
extern bool countingSmokeTimer;
extern bool countingKeypadTimer;
extern byte sensorTimer;
extern byte phoneTimer;
extern byte smokeTimer;
extern byte keypadTimer;
extern byte intCount;
// Message passing values
bool replyArrived = false;

// Menu indices
byte passwordIndex = 1;
byte phoneNumberIndex = 0;
byte curMenu = 0; // index into menus[]
byte lastMenu = 0;

// User defined settings
byte pickupDelay = 1;
byte smokeDelay = 5;
byte sensorDelay = 2;
byte Passwords[PASSWORD_BUFFER_SIZE][KEYPAD_BUFFER_SIZE] = {{ 1, 2, 3, 4,
5,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY},
{EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY},
{EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY},
{EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY},
{EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY},
{EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY},
{EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY},
{EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY}};
byte PhoneNumbers[PHONE_NUMBER_BUFFER_SIZE][10] = {{9,5,1,2,9,3,5,5,EMPTY,EMPTY},
{9,6,9,4,1,4,0,2,EMPTY,EMPTY},
{EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY},
```

```
    {EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY},
    {EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY},
    {EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY},
    {EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY},
    {EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY}};

    byte keypadBuffer[KEYPAD_BUFFER_SIZE] =
    {EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY};


//-------------------------------------------------------------------------------
    void main(void){
    EnableInterrupts;
    InitializePorts();
    InitializeMenus();
    InitializeLCD();
    DisplayMenu(curMenu); // Show initial menu

    while(true){
    UpdateKeypad();// Update Keystate
    DoKeypadActions(newKeypadState); // For newly pressed buttons only
    ProcessMessages();// Check interrupt-driven message buffer
    CheckSensors();// Check Sensor states and handle actions if any are active
    UpdateTimers();// Handles timers going off
     }
}
    //--------------   Interrupts/Message Passing   -------------------------
    void interrupt 8 OuterInterrupt(void){  // PT0
    Outer();
    TFLG1 |= 0x01; // ack, clear C0F: This makes interrrupt 0 on PT0 ready to interrupt again.
}
    void interrupt 9 InnerInterrupt(void){ // PT1
    Inner();
    TFLG1 |= 0x02; // ack, clear C1F: This makes interrrupt 9 on PT1 ready to interrupt again.
}


void interrupt 13 TC5handler(void){
    intCount++;
    TFLG1= 0x20; // Acknowledge
    TC5 = TC5+RATE;
    //MoveCursorTo(0);
    //LCDOutInt(intCount);
}
//-------------------------------------------------------------------------
    // Check the message buffer and handle logic. Some messages are questions that
```

```
    // need answers sent, and some messages are >1 packet, requiring a wait.
void ProcessMessages(void){
    byte msg;
    byte I, j;
    byte temp;
    byte tempBuff[KEYPAD_BUFFER_SIZE];

    // init temporary buffer
    for(I=0;I<KEYPAD_BUFFER_SIZE;I++)
    tempBuff[I] = EMPTY;

    while(true)
   {
    msg = Pull(); // Get next msg

    switch(msg){
    case(EMPTY):
    return;// empty msg means end of message buffer

    case(MI_ARM):
    JumpToExactMenu(ARMED_MENU);
    break;

    case(MI_DISARM):
    if(IS_ARMED)
    JumpToExactMenu(UNARMED_MENU);
    break;

    case(MI_LOCKDOWN):
    JumpToExactMenu(LOCKDOWN_MENU);
    break;

    case(MI_IS_ADMIN_PASSWORD):
    while(Peek(0)==EMPTY); // wait for length
    temp = Pull();
    while(Peek(temp-1)==EMPTY); // wait for password
    for(I=0;I<temp;I++)
    tempBuff[I] = Pull();
    if(Equals(tempBuff, Passwords[0])) Send(MO_YES);
    else Send(MO_NO);
    Send(MI_IS_ADMIN_PASSWORD);
    break;

    case(MI_IS_ANY_PASSWORD):
    while(Peek(0)==EMPTY); // wait for length
    temp = Pull();
```

```
while(Peek(temp-1)==EMPTY); // wait for password
for(I=0;I<temp;I++)
tempBuff[I] = Pull();
temp = false;
for(I=0;I<PASSWORD_BUFFER_SIZE;I++)
if(Equals(tempBuff, Passwords[0]))
temp = true;
if(temp==true) Send(MO_YES);
else Send(MO_NO);
break;

case(MI_NEW_ADMIN_PASSWORD):
while(Peek(0)==EMPTY); // wait for length
temp = Pull();
while(Peek(temp-1)==EMPTY); // wait for password
for(I=0;I<temp;I++)
Passwords[0][I] = Pull();
break;

case(MI_NEW_REGULAR_PASSWORD):
for(j=0;j<PASSWORD_BUFFER_SIZE && Passwords[j][0]!=EMPTY;j++);//Find first available slot
if(j==PASSWORD_BUFFER_SIZE) break; // Password buffer is full
while(Peek(0)==EMPTY); // wait for length
temp = Pull();
while(Peek(temp-1)==EMPTY); // wait for password
for(I=0;I<temp;I++)
Passwords[j][I] = Pull();
break;

case(MI_GET_PHONE_NUMBER):
while(Peek(0)==EMPTY);// waiting for index #
temp = Pull();
if(PhoneNumbers[temp][0]==EMPTY){
Send(MO_NO);
Send(MI_GET_PHONE_NUMBER);
    }
else{
Send(MO_HERE_COMES_A_NUMBER);
for(I=0;I<10;I++){
Send(PhoneNumbers[temp][I]==EMPTY ? 10 : PhoneNumbers[temp][I]);
     }
    }
break;

case(MI_PHONE_RANG):
countingPhoneTimer = true;
break;
```

```
case(MI_GET_PICKUP_DELAY):
Send(MO_HERE_COMES_PICKUP);
if(pickupDelay > 10) Send((pickupDelay/10)%10);
Send(pickupDelay%10);
Send(MO_DONE);
break;

case(MI_NEW_PICKUP_DELAY):
while(Peek(1) != MI_DONE && Peek(2) != MI_DONE);
if(Peek(1)==MI_DONE){
// 1 digit pickup delay time
pickupDelay = Pull();
    }
else{
// 2 digit pickup delay time
pickupDelay = 10*Pull();
pickupDelay += Pull();
    }
Pull();// Pull MO_DONE msg
break;

case(MI_GET_SENSOR_DELAY):
Send(MO_HERE_COMES_A_SENSOR);
if(sensorDelay > 10) Send((sensorDelay/10)%10);
Send(sensorDelay%10);
Send(MO_DONE);
break;

case(MI_NEW_SENSOR_DELAY):
while(Peek(1) != MI_DONE && Peek(2) != MI_DONE);
if(Peek(1)==MI_DONE){
// 1 digit sensor delay time
sensorDelay = Pull();
    }
else{
// 2 digit sensor delay time
sensorDelay = 10*Pull();
sensorDelay += Pull();
    }
Pull();// Pull MO_DONE msg
break;

case(MI_HERE_COMES_A_TEMP): // Applies to all Temp related incoming messages, since all carry 1
packet answers
while(Peek(0)==EMPTY); // wait for second packet
```

```
    replyArrived = true;
    return;
     }
   }
 }
//-------------------------------------------------------------------------------------------------
    // Handles all special cases for all menus. Whenever a menu change(state change) occurs, a
    //JumpToMenu() is
    // called to handle the one time effects that need to occur.
void JumpToExactMenu(byte newMenuIndex){
    lastMenu = curMenu;
    curMenu = newMenuIndex;
    JumpToMenu(EFFECTS_ONLY);
 }

void JumpToMenu(byte newMenuIndex){
    byte I=0;
    byte j=0;
    byte swap=0;//multi-use variables
    byte tempBuff[KEYPAD_BUFFER_SIZE];
    bool wasAdmin = IS_ADMIN; // for comparison later


    if(newMenuIndex != EFFECTS_ONLY)
    lastMenu = curMenu;


    if(newMenuIndex==PROCESS_KEYPAD_BUFFER){
    // Determine which menu to go to based on contents of keypadBuffer and curMenu
    // Admin Password Menus
    if(curMenu==1 || curMenu==2 || curMenu==3){
    if(IsValidPassword()){
    if(Equals(keypadBuffer, Passwords[0])) newMenuIndex = CORRECT_MENU;
    else newMenuIndex = INCORRECT_MENU;
      }
    else newMenuIndex = INVALID_MENU;
     }
    // Any Password Menus
    else if(curMenu==6 || curMenu==7 || curMenu==8){
    if(IsValidPassword()){
    for(I=0;I<PASSWORD_BUFFER_SIZE;I++){
    if(Equals(keypadBuffer, Passwords[I])) newMenuIndex = CORRECT_MENU;
       }
    if(newMenuIndex==PROCESS_KEYPAD_BUFFER) //no correct password triggered a
    JumpToMenu(correct), so password is wrong
    newMenuIndex = INCORRECT_MENU;
```

```
  }
else newMenuIndex = INVALID_MENU;
 }
// Entering a phone number to add to bottom of red list
else if(curMenu==11 || curMenu==12){
if(keypadBuffer[6]!=EMPTY){// Must be at least 7 numbers
newMenuIndex = CORRECT_MENU;
  }
else newMenuIndex = INVALID_MENU;
 }
// Entering new admin password
else if(curMenu==20){
if(IsValidPassword()){
newMenuIndex = CORRECT_MENU;
for(I=0;I<KEYPAD_BUFFER_SIZE;I++)
tempBuff[I] = keypadBuffer[I];
  }
else newMenuIndex = INVALID_MENU;
 }
else if(curMenu==29){
if(IsValidPassword()){
if(Equals(keypadBuffer, tempBuff)) newMenuIndex = CORRECT_MENU;
else newMenuIndex = INCORRECT_MENU;
  }
else newMenuIndex = INVALID_MENU;
 }
// Entering new regular password
else if(curMenu==21){
if(IsValidPassword()){
if(Passwords[PASSWORD_BUFFER_SIZE-1][0]!=EMPTY) newMenuIndex = HIGHER_MENU; // if
buffer is full, go up a menu level
else newMenuIndex = CORRECT_MENU;
  }
else newMenuIndex = INVALID_MENU;
 }
// Entering a new Pickup Delay, [0-15]sec
else if(curMenu==26){
if(keypadBuffer[2]==EMPTY && keypadBuffer[0]!=0 && (10 * keypadBuffer[0] + keypadBuffer[1]) <
16)
newMenuIndex = CORRECT_MENU;
else if(keypadBuffer[1]==EMPTY) newMenuIndex = CORRECT_MENU; // single digit number
else newMenuIndex = INVALID_MENU;
 }
// Display Countdown while buffering input
```

```
  else if(curMenu==31){



   }
  // Fire Alarm
  else if(curMenu==32 && lastMenu!=32){
  Send(MO_SMOKE);
   }
 }


 //-------------------------------------------------------------------------------------------------
 // All menus now have a straightforward lookup
 //-------------------------------------------------------------------------------------------------


  if(newMenuIndex != EFFECTS_ONLY)
     curMenu = menuInteractionTable[curMenu][newMenuIndex];
  DisplayMenu(curMenu);


  // Let Master MCU know if the Admin is logged in here or not
  if(IS_ADMIN && !wasAdmin) Send(MO_ADMIN_LOCKOUT);
  if(!IS_ADMIN && wasAdmin) Send(MO_ADMIN_UNLOCK);


  // Do effects specific to certain menus and circumstances
  if(curMenu==0 || curMenu==9) { // Successful login menus. Cancel and clear timers
  countingKeypadTimer = false;
  keypadTimer = 0;
  countingSensorTimer = false;
  sensorTimer = 0;
  countingSmokeTimer = false;
  smokeTimer = 0;
 }


 // --0--
  if(curMenu==0 && lastMenu==8){// Canceled a fire alarm after the phone call was placed
  Send(MO_CANCEL_SMOKE);
 }
 // --2&3--
  else if(curMenu==2 || curMenu==3){// Someone is trying to login as admin. Start a timer to call owner if
  no correct PW
  countingKeypadTimer = true;
 }
```

```
// --4--
 else if(curMenu==4 && (lastMenu==3 || lastMenu==2 || lastMenu==3)){// Incorrect password entered.
 Call owner
 Send(MO_KEYPAD_ALERT);// Send Alert to Phone Unit
 }
 else if(curMenu==4 && lastMenu==6) // came from a timed out sensor
 {




 }
// --6--
 else if(curMenu==6)
 {
 MoveCursorTo(11);
 if(countingKeypadTimer) LCDOutInt(sensorDelay-keypadTimer);
 else LCDOutInt(sensorDelay-sensorTimer);// Must be counting a sensor timer
 MoveCursorTo(LCD_LINE_LENGTH+5);
 }
// --7--
 else if(curMenu==7){
 if(!countingSmokeTimer){
 countingSmokeTimer = true;
  }
 }
// --8--
 else if(curMenu==8 && lastMenu!=8){
 Send(MO_SMOKE); // Sends fire alarm to owner's phone when received
  }
 // --10--
 else if(curMenu==10 && lastMenu==17){
 if(newKeypadState & KP_2){// Move phone number up the list
 if(phoneNumberIndex!=0){
 for(I=0;I<10;I++){
 swap = PhoneNumbers[phoneNumberIndex][I];
 PhoneNumbers[phoneNumberIndex][I] = PhoneNumbers[phoneNumberIndex-1][I];
 PhoneNumbers[phoneNumberIndex-1][I] = swap;
     }
   }
  }
 else if(newKeypadState & KP_3){// Move phone number down the list
 if(phoneNumberIndex!=PHONE_NUMBER_BUFFER_SIZE-1 &&
```

```
PhoneNumbers[phoneNumberIndex+1][0]!=EMPTY){
for(I=0;I<10;I++){
swap = PhoneNumbers[phoneNumberIndex][I];
PhoneNumbers[phoneNumberIndex][I] = PhoneNumbers[phoneNumberIndex+1][I];
PhoneNumbers[phoneNumberIndex+1][I] = swap;
    }
  }
 }
}
// --13--
else if(curMenu==13){//Add phone number to list
AddPhoneNumber(keypadBuffer);
}
// --14--
else if(curMenu==14){// Phone number list
if(lastMenu==14){// Pressed 2or8 for up or down the list
if((phoneNumberIndex > 0) && (newKeypadState & KP_2))
phoneNumberIndex--;
else if((phoneNumberIndex < PHONE_NUMBER_BUFFER_SIZE-1 &&
PhoneNumbers[phoneNumberIndex+1][0]!=EMPTY) && (newKeypadState & KP_8))
phoneNumberIndex++;
 }
else{
phoneNumberIndex=0; // Coming from a diff menu
 }


//Display current number in phone number list
MoveCursorTo(19);
if(PhoneNumbers[phoneNumberIndex][0] == EMPTY){
LCDOutString("No Phone #s", 10);
 }
else{
for(I=0;I<10;I++) LCDOutChar(48+PhoneNumbers[phoneNumberIndex][I]);
 }
MoveCursorTo(LCD_LINE_LENGTH*2);
}
// --17--
else if(curMenu==17 && lastMenu==14 && PhoneNumbers[phoneNumberIndex][0]==EMPTY){
// User is trying to delete a non-existant number, go back a menu.
CurMenu = lastMenu;
DisplayMenu(curMenu);
MoveCursorTo(19);
LCDOutString("No Phone #s", 10);
MoveCursorTo(LCD_LINE_LENGTH*2);
```

```
}
// --18--
else if(curMenu==18){// Delete number from menu 14
for(;phoneNumberIndex<PHONE_NUMBER_BUFFER_SIZE-1;phoneNumberIndex++)
for(I=0;I<10;I++) PhoneNumbers[phoneNumberIndex][I] = PhoneNumbers[phoneNumberIndex+1][I];
for(I=0;I<10;I++) PhoneNumbers[phoneNumberIndex][I] = EMPTY;
}
// --22--
else if(curMenu==22){// Display current password
if(lastMenu==22){// Pressed 2or8 for up or down
if((passwordIndex > 1) && (newKeypadState & KP_2))
passwordIndex--;
else if((passwordIndex < PASSWORD_BUFFER_SIZE-1 && Passwords[passwordIndex+1][0]!
=EMPTY) && (newKeypadState & KP_8))
passwordIndex++;
 }
else{
passwordIndex=1; // Coming from a different menu
 }


//Display current password from password list
MoveCursorTo(20);
if(Passwords[passwordIndex][0] == EMPTY){
LCDOutString(" No Pws", 7);
 }
else{
for(I=0;I<PASSWORD_LENGTH;I++){
if(Passwords[passwordIndex][I] != EMPTY) LCDOutChar(48+Passwords[passwordIndex][I]);
else LCDOutChar(' ');
   }
 }
MoveCursorTo(LCD_LINE_LENGTH*2);
}
// --25--
else if(curMenu==25 && lastMenu==29){
for(I=0;I<KEYPAD_BUFFER_SIZE;I++) Passwords[0][I] = keypadBuffer[I];
}
else if(curMenu==25 && lastMenu==21){
for(I=0;I<PASSWORD_BUFFER_SIZE && Passwords[I][0]!=EMPTY;I++) ; // I==first available
password slot.
For(j=0;j<KEYPAD_BUFFER_SIZE;j++) Passwords[I][j] = keypadBuffer[j];
}
else if(curMenu==25 && lastMenu==22){// Delete Password from menu 22
if(Passwords[passwordIndex][0]==EMPTY){
```

```
  // User is trying to delete a non-existant password, go back a menu.
  CurMenu = lastMenu;
  DisplayMenu(curMenu);
  MoveCursorTo(20);
  LCDOutString(" No Pws", 7);
  MoveCursorTo(LCD_LINE_LENGTH*2);
   }
  else{
  for(;passwordIndex<PASSWORD_BUFFER_SIZE-1;passwordIndex++)
  for(I=0;I<PASSWORD_LENGTH;I++) Passwords[passwordIndex][I] = Passwords[passwordIndex+1][I];
  for(I=0;I<PASSWORD_LENGTH;I++) Passwords[passwordIndex][I] = EMPTY;
   }
 }
  // --26--
  else if(curMenu==26){// Display current pickup delay
  MoveCursorTo(14);
  LCDOutInt(pickupDelay);
  MoveCursorTo(menus[curMenu].initialCursorLocation);
 }
  // --27--
  else if(curMenu==27){// Display new pickup delay
  if(keypadBuffer[1]==EMPTY){
  pickupDelay = keypadBuffer[0];
  MoveCursorTo(27);
  LCDOutInt(pickupDelay);
  MoveCursorTo(LCD_LINE_LENGTH*2);
   }
  else{// Double digit value
  pickupDelay = 10*keypadBuffer[0] + keypadBuffer[1];
  MoveCursorTo(27);
  LCDOutInt(pickupDelay);
  MoveCursorTo(LCD_LINE_LENGTH*2);
   }
 }
  //--28--
  else if(curMenu==28){
  MoveCursorTo(13);
  Send(MO_GET_TEMP);
  replyArrived = false;
  while(!replyArrived) ProcessMessages();
  LCDOutInt(Pull());
  MoveCursorTo(LCD_LINE_LENGTH*2); //off screen
  replyArrived = false;
 }
  //--32--
```

```
else if(curMenu==32){
//First change the heater or AC
if(newKeypadState & KP_1){
Send(MO_TOGGLE_HEATER);
 }
else{
Send(MO_TOGGLE_COOLER);
 }


MoveCursorTo(10);
replyArrived = false;
Send(MO_GET_HEATER_STATE);
while(!replyArrived) ProcessMessages();
if(Pull()) LCDOutString(”on”,2);
else LCDOutString(”off”,3);
replyArrived=false;
Send(MO_GET_COOLER_STATE);
while(!replyArrived) ProcessMessages();
if(Pull()) LCDOutString(”on”,2);
else LCDOutString(”off”,3);
MoveCursorTo(LCD_LINE_LENGTH*2);
replyArrived = false;
}
// --33--
else if(curMenu==33){// Display current sensor delay
MoveCursorTo(14);
LCDOutInt(sensorDelay);
MoveCursorTo(menus[curMenu].initialCursorLocation);
}
// --34--
else if(curMenu==34){// Display new sensor delay
if(keypadBuffer[1]==EMPTY){
sensorDelay = keypadBuffer[0];
MoveCursorTo(27);
LCDOutInt(sensorDelay);
MoveCursorTo(LCD_LINE_LENGTH*2);
 }
else{// Double digit value
sensorDelay = 10*keypadBuffer[0] + keypadBuffer[1];
MoveCursorTo(27);
LCDOutInt(sensorDelay);
MoveCursorTo(LCD_LINE_LENGTH*2);
 }

}
// --36--
else if(curMenu==36){
```

```
    MoveCursorTo(10);
    replyArrived = false;
    Send(MO_GET_MIN_TEMP);
    while(!replyArrived) ProcessMessages();
    LCDOutInt(Pull());
    MoveCursorTo(menus[curMenu].initialCursorLocation);
    replyArrived = false;
   }
    // --37--
    else if(curMenu==37){
    MoveCursorTo(10);
    replyArrived = false;
    Send(MO_GET_MAX_TEMP);
    while(!replyArrived) ProcessMessages();
    LCDOutInt(Pull());
    MoveCursorTo(menus[curMenu].initialCursorLocation);
    replyArrived = false;
   }
    // --38--
    else if(curMenu==38){
    I = keypadBuffer[0];
    if(keypadBuffer[1] != EMPTY)
    I += keypadBuffer[1];
    ClearKeypadBuffer();

    if(lastMenu==36) Send(MO_HERE_COMES_MIN_TEMP);
    else Send(MO_HERE_COMES_MAX_TEMP);
    Send(I); // value from keypad buffer

    // Display values on LCD
    MoveCursorTo(10);
    replyArrived=false;
    Send(MO_GET_MIN_TEMP);
    while(!replyArrived) ProcessMessages();
    LCDOutInt(Pull());
    MoveCursorTo(LCD_LINE_LENGTH+10);
    replyArrived = false;
    Send(MO_GET_MAX_TEMP);
    while(!replyArrived) ProcessMessages();
    LCDOutInt(Pull());
    MoveCursorTo(LCD_LINE_LENGTH*2);
    replyArrived = false;
   }
 }
//---------------------------------- menu.c ----------------------------------------------------
    #include “stdafx.h”
```

```
Menu menus[TOTAL_MENUS];

// Buttons map to the 4 entries as such:
// { 1     ,    2/8  ,      3/#    ,HIGHER}
// {Correct,Incorrect,Invalid Input,HIGHER} for Buffered input
byte menuInteractionTable[TOTAL_MENUS][4]={
{ 1, 0, 5, 0},//0
{ 9, 2, 1, 0},//1
{ 9, 3, 2, 2},//2
{ 9, 4, 3, 3},//3
{ 4, 4, 4, 4},//4
{ 6, 6, 6, 6},//5
{ 0, 6, 6, 6},//6
{ 8, 7, 0, 7},//7
{ 0, 8, 8, 8},//8
{10,19,15, 0},//9
{14,11,10, 9},//10
{13,12,12,10},//11
{13,12,12,10},//12
{10,10,10,10},//13
{17,14,16,10},//14
{26,33,15, 9},//15
{14,14,14,14},//16
{18,10,10,14},//17
{10,10,10,14},//18
{20,21,22, 9},//19
{29,20,20,19},//20
{25,21,21,19},//21
{25,22,24,19},//22
{ 6, 6, 6, 6},//23
{22,22,22,22},//24
{19,19,19,19},//25
{27,26,26, 9},//26
{ 9, 9, 9, 9},//27
{31,35,28, 9},//28
{25,30,29,20},//29
{19,19,19,19},//30
{32,32,31,28},//31
{28,28,28,28},//32
{99,99,99,99},//33
{99,99,99,99},//34
{36,37,35,31},//35
{38,36,36,35},//36
{38,37,37,35},//37
{28,28,28,28}};//38

void InitializeMenus(void){
// Home Menus
```

```
  PopulateMenu(0, "1)Settings      3)Arm          ", EMPTY);
  PopulateMenu(1, "Enter Admin PW:               ", LCD_LINE_LENGTH);
  // ARMED MENUS
  PopulateMenu(2, "Incorrect. 2nd  Attempt:       ", LCD_LINE_LENGTH+8);
  PopulateMenu(3, "Incorrect.Last  Attempt:       ", LCD_LINE_LENGTH+8);
  PopulateMenu(4, "--Keypad Locked---Calling Owner-", EMPTY);
  PopulateMenu(5, "------ARMED----- Press any key  ", EMPTY);
  PopulateMenu(6, "Time Left:      Code:          ", LCD_LINE_LENGTH+6);
  // Smoke Alarm Menus
  PopulateMenu(7, "! 1)FIRE ALARM !! 3)Main Menu   !", LCD_LINE_LENGTH+4);
  PopulateMenu(8, "-Calling Owner- Press 1 to cancel", LCD_LINE_LENGTH+4);
  // Admin Menu
  PopulateMenu(9, "1Phone#s 2Codes 3Timers 4Heat/AC", EMPTY);
  // Phone Number Menus
  PopulateMenu(10,"1Edit Phone List2Add New Number ", EMPTY);
  PopulateMenu(11,"Enter a Phone #                 ", LCD_LINE_LENGTH);
  PopulateMenu(12,"Invalid. Enter#                 ", LCD_LINE_LENGTH+6);
  PopulateMenu(13,"Added number at lowest priority.", EMPTY);
  PopulateMenu(14,"For help press 3-->         <--", EMPTY);
  PopulateMenu(15,"1 Pickup Delay  2 Sensor Delay  ", EMPTY);
  PopulateMenu(16,"Select Number:1 Up/Down: 2/8    ", EMPTY);
  PopulateMenu(17,"1)Delete 2)Move up 3)Move down  ", EMPTY);
  PopulateMenu(18,"Number deleted. Press any key.  ", EMPTY);
  // Password Menus
  PopulateMenu(19,"1Edit Admin PW  2Add PW  3Del PW", EMPTY);
  PopulateMenu(20,"Enter New Admin PW:             ", LCD_LINE_LENGTH+3);
  PopulateMenu(21,"Enter New PW:                   ", LCD_LINE_LENGTH);
  PopulateMenu(22,"For help press 3--->       <---", EMPTY);
  PopulateMenu(24,"Del Password: 1 Up / Down: 2 / 8", EMPTY);
  PopulateMenu(25,"Password List   Updated         ", EMPTY);
  // Pickup time
  PopulateMenu(26,"Current Delay:  New Delay:      ", LCD_LINE_LENGTH+10);
  PopulateMenu(27,"Pickup Delay    changed to      ", EMPTY);
  // New Additions
  PopulateMenu(29,"Confirm NewAdminPW:             ", LCD_LINE_LENGTH+3);
  PopulateMenu(30,"Those Passwords   didn't match. ", EMPTY);
  PopulateMenu(33,"Current Delay:  New Delay:      ", LCD_LINE_LENGTH+10);
  PopulateMenu(34,"Sensor Delay    changed to:     ", EMPTY);
  PopulateMenu(23,"--Armed Fire--  Calling Owner   ", EMPTY);
  // Temperature menus
  PopulateMenu(28,"Current Temp:   1Mode 2Hi/Lo Tmp", EMPTY);
  PopulateMenu(31,"1 Toggle Heater 2 Toggle AirCond", EMPTY);
  PopulateMenu(32,"Heater is       A/C is          ", EMPTY);
  PopulateMenu(35,"1Change Min Temp2Change Max Temp", EMPTY);
  PopulateMenu(36,"Current Min:    Change To:      ", LCD_LINE_LENGTH+10);
  PopulateMenu(37,"Current Max:    Change To:      ", LCD_LINE_LENGTH+10);
  PopulateMenu(38,"Min Temp:       Max Temp:       ", EMPTY);
}
```

```
void PopulateMenu(byte menuIndex, char message[32], byte initialCursorLocation){
    // Helper function to build menus
    byte I=0;
    for(;I<LCD_LINE_LENGTH;I++){
    menus[menuIndex].line1[I] = message[I];
    menus[menuIndex].line2[I] = message[I+LCD_LINE_LENGTH];
   }
    menus[menuIndex].initialCursorLocation = initialCursorLocation;
 }


void DisplayMenu(byte showMe){
    // Refresh LCD screen with the menu index showMe
    MoveCursorTo(0);
    LCDOutString(menus[showMe].line1, LCD_LINE_LENGTH);
    MoveCursorTo(16);
    LCDOutString(menus[showMe].line2, LCD_LINE_LENGTH);
    if(menus[showMe].initialCursorLocation!=EMPTY)
    MoveCursorTo(menus[showMe].initialCursorLocation);
 }


 //-------------------------------- messages.c ----------------------------------------------------
    #include “stdafx.h”
    // A simple message passing scheme. The message is an integer value equal to the number of Inner()’s
    called
    // between calls to Outer().
    byte interruptCounter = 0;
    bool msgFlag = false;
    byte inputBuffer[INPUT_BUFFER_SIZE] =
    {EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMP
    TY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,E
    MPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY,EMPT
    Y,EMPTY,EMPTY,EMPTY,EMPTY,EMPTY};


void Send(byte out){
    byte I = 0;
    //CALL OUTER INT
    PTM |= 0x10;
    delay(1);
    PTM &= 0xef;
    delay(1);


    for(;I<out;I++)
   {
    //CALL INNER INT;
    PTM |= 0x20;
    delay(1);
```

```
    PTM &= 0xdf;
    delay(1);
   }


    //CALL OUTER INT
    PTM |= 0x10;
    delay(1);
    PTM &= 0xef;
    delay(1);
 }


void Outer(void){
    if(msgFlag)
   {
    Push(interruptCounter);
    interruptCounter = 0;
    msgFlag = false;
   }
    else
   {
    msgFlag = true;
   }
 }


void Inner(void){
    interruptCounter++;
 }


void Push(byte addMe)
 {
    byte I;
    for(I=0;I<INPUT_BUFFER_SIZE;I++)
   {
    if(inputBuffer[I] == EMPTY)
     {
    inputBuffer[I] = addMe;
    return;
     }
   }
 }
```

```
byte Pull(void)
 {
    byte I;
    byte returnValue = inputBuffer[0];
    // pull messages off input buffer
    if(returnValue != EMPTY)
   {
    // Shift buffer up a slot
    for(I=1;I<INPUT_BUFFER_SIZE;I++)
     {
    inputBuffer[I-1] = inputBuffer[I];
     }
    inputBuffer[INPUT_BUFFER_SIZE-1] = EMPTY;
   }
    return returnValue;
 }


byte Peek(byte index){
    return inputBuffer[index];
 }


 //---------------------------------- stdafx.c ----------------------------------------------------
    #include “stdafx.h”

    extern byte keypadBuffer[KEYPAD_BUFFER_SIZE];
    extern byte PhoneNumbers[PHONE_NUMBER_BUFFER_SIZE][10];

    void delay(unsigned long int ms) {
    for (;ms > 0; ms--) {
    asm {
    PSHX
    LDX #$640
    Loop:
    NOP
    NOP
    DBNE X, Loop
    PULX
     }
  }
 }
```

```
void InitializePorts(void){
    DDRM = DDRM_Button_Mask_Slave;
    DDRAD = DDRAD_Button_Mask;
    DDRT = DDRT_Button_Mask;

    PERT |= 0x1c; // pull-down/up enable
    PPST |= 0x1c; // 1 is pull-down      to pull of the extra voltage
    PERM |= 0x04; // pull-down/up enable
    PPSM |= 0x04; // 1 is pull-down      to pull of the extra voltage



    TIOS |= 0x20; // enable OC5
    TSCR2 = 0x7; // 16 us clock
    TIE |= 0x23;//83; // Arm OC0  on PT0 & Arm OC1  on PT1   pg 279
    TC5 = TCNT+RATE; // First in 10 ms



    TCTL4 = 0x05; // rising edge on PT0 and PT1        pg 279
    TFLG1 = 0x03;//0x83; // clears interrupt PT0 and PT1 ready for the next interrupt
    TSCR1 = 0x80; // enable TEN
 }


bool Equals(byte* a, byte* b){
    // Compares password buffers
    byte I=0;
    for(;I<KEYPAD_BUFFER_SIZE;I++)
    if(a[I]!=b[I]) return false;
    return true;
 }


bool IsValidPassword(){
    if(keypadBuffer[3]==EMPTY) // not >3 characters
    return false;
    if(keypadBuffer[PASSWORD_LENGTH] != EMPTY) // not <max password length
    return false;
    return true;
 }


void AddPhoneNumber(byte* num){
    // Add a phone number to the bottom of the phone number list if possible
    byte I=0;
    byte j=0;

    if(PhoneNumbers[PHONE_NUMBER_BUFFER_SIZE-1][0] != EMPTY) // Phone Number Buffer is full
    return;
```

```
    while(PhoneNumbers[I][0] != EMPTY) I++;

    for(;j<10;j++)
    PhoneNumbers[I][j] = num[j];
  }

void BufferKeypress(byte key){
    // Bufferes Keypresses from the console keypad
    byte I=0;
    for(;I<KEYPAD_BUFFER_SIZE;I++){
    if(keypadBuffer[I] == EMPTY){
    keypadBuffer[I] = key;
    LCDOutChar(key+48);
    return;
     }
   }
  }

 //---------------------------------- stdafx.h ----------------------------------------------------
    #include <hidef.h>      /* common defines and macros */
    #include <mc9s12c32.h>     // derivative information

    // Outgoing Messages
    #define MO_PICKUP 1
    #define MO_NO 2 //Should be followed by the Question that was asked with Incoming Message encoding
    #define MO_YES 3//Should be followed by the Question that was asked with Incoming Message
    encoding
    #define MO_KEYPAD_ALERT 4
    #define MO_HERE_COMES_A_NUMBER 5
    #define MO_HERE_COMES_PICKUP 6
    #define MO_DONE 7
    #define MO_SMOKE 8
    #define MO_DOOR 9
    #define MO_MOTION 10
    #define MO_WINDOW 11
    #define MO_ADMIN_LOCKOUT 12
    #define MO_ADMIN_UNLOCK 13
    #define MO_HERE_COMES_A_SENSOR 14
    #define MO_CANCEL_SMOKE 15
    #define MO_GET_TEMP 16
    #define MO_TOGGLE_HEATER 17
    #define MO_TOGGLE_COOLER 18
    #define MO_GET_HEATER_STATE 19
    #define MO_GET_COOLER_STATE 20
    #define MO_GET_MIN_TEMP 21
    #define MO_GET_MAX_TEMP 22
```

```
#define MO_HERE_COMES_MIN_TEMP 23
#define MO_HERE_COMES_MAX_TEMP 24


// Incoming Messages
#define MI_NO 0
#define MI_YES 1
#define MI_ARM 2
#define MI_DISARM 3
#define MI_LOCKDOWN 4
#define MI_IS_ADMIN_PASSWORD 5
#define MI_IS_ANY_PASSWORD 6
#define MI_GET_PICKUP_DELAY 7
#define MI_NEW_ADMIN_PASSWORD 8
#define MI_NEW_REGULAR_PASSWORD 9
#define MI_NEW_PICKUP_DELAY 10
#define MI_PHONE_RANG 11
#define MI_GET_PHONE_NUMBER 12  // THIS VARIABLE MUST STAY ABOVE 9
#define MI_GET_SENSOR_DELAY 13
#define MI_NEW_SENSOR_DELAY 14
#define MI_DONE 15
#define MI_HERE_COMES_A_TEMP 16


// Magic Numbers
#define TOTAL_MENUS 39
#define KEYPAD_INPUTS 12
#define LCD_LINE_LENGTH 16
#define INPUT_BUFFER_SIZE 40
#define KEYPAD_BUFFER_SIZE 10
#define PASSWORD_BUFFER_SIZE 10
#define PHONE_NUMBER_BUFFER_SIZE 10
#define PASSWORD_LENGTH 8


// JumpToMenu unique values, must not match any menu index
#define PROCESS_KEYPAD_BUFFER TOTAL_MENUS
#define EFFECTS_ONLY PROCESS_KEYPAD_BUFFER + 1


// MenuInteractionTable Indices
#define CORRECT_MENU 0
#define INCORRECT_MENU 1
#define INVALID_MENU 2
#define HIGHER_MENU 3
#define MENU_1 0
#define MENU_2 1
#define MENU_8 1
#define MENU_3 2
#define MENU_POUND 2
#define EMPTY 0xFF
```

```
// Menu Macros + Indices
#define IS_ARMED (curMenu>=2 && curMenu<=6)
#define IS_ADMIN (curMenu>=9)
#define UNARMED_MENU 0
#define LOCKDOWN_MENU 4
#define ARMED_MENU 5
#define ADMIN_MENU 9
#define SMOKE_MENU 7
#define UNARMED_FIRE_MENU 8
#define ARMED_FIRE_MENU 23


// Port masks
#define DDRAD_Button_Mask 0xFF  // outputs = 1 & input = 0
#define DDRT_Button_Mask 0x00 // outputs = 1 & input = 0
#define DDRM_Button_Mask_Slave 0xFF


// Keypad masks for KeypadState values
#define KP_1 0x800
#define KP_2 0x400
#define KP_3 0x200
#define KP_4 0x100
#define KP_5 0x080
#define KP_6 0x040
#define KP_7 0x020
#define KP_8 0x010
#define KP_9 0x008
#define KP_STAR 0x004
#define KP_0 0x002
#define KP_POUND 0x001


// Timer value
#define RATE 62466;


// Definitions for non-ascii symbols
#define RIGHT_ARROW 0b01111110
#define LEFT_ARROW  0b01111111


// bool definition
typedef unsigned char bool;
#define true (0==0)
#define false !true


// Menu definition
typedef struct{
char line1[LCD_LINE_LENGTH];
char line2[LCD_LINE_LENGTH];
byte initialCursorLocation;
} Menu;
```

```
// all functions
extern void delay(unsigned long int ms);
extern void JumpToExactMenu(byte newMenuIndex);
extern void JumpToMenu(byte newMenuIndex);
extern void ProcessMessages(void);
extern void InitializePorts(void);
extern bool Equals(byte* a, byte* b);
extern bool IsValidPassword(void);
extern void AddPhoneNumber(byte* num);
extern void BufferKeypress(byte key);

// Timers
extern bool PhoneTimerUp(void);
extern bool SensorTimerUp(void);
extern bool SmokeTimerUp(void);
extern bool KeypadTimerUp(void);
extern void UpdateTimers(void);
void TimedInterruptHandler(void);

// Keypad
extern void UpdateKeypad(void);
extern void DoKeypadActions(short doAction);
extern void ClearKeypadBuffer(void);
//LCD
extern void InstructLCD(short instruction, byte delayAfterInstruction, byte delayAfterDisable);
extern void InitializeLCD(void);
extern void LCDOutChar(char);
extern void LCDOutString(char*, byte);
extern void LCDOutInt(byte);
extern void MoveCursorTo(byte);
//Menu
extern void InitializeMenus(void);
extern void DisplayMenu(byte showMe);
extern void PopulateMenu(byte menuIndex, char message[32], byte initialCursorLocation);
//Messages
extern void Send(byte);
extern void Inner(void);
extern void Outer(void);
extern void Push(byte);
extern byte Pull(void);
extern byte Peek(byte);
//Sensors
extern void CheckSensors(void);
extern bool GetDoorState(void);
extern bool GetMotionState(void);
extern bool GetSmokeState(void);
extern bool GetWindowState(void);
```

```
//---------------------------------- lcd.c ----------------------------------------------------------
   #include “stdafx.h”

void InstructLCD(short instruction, byte delayAfterInstruction, byte delayAfterDisable){
   PTM |= 2;// enable
   if(instruction & 0x200) PTM |= 0x01;// RS = 1
   else PTM &= 0xfe;// RS = 0
   PTAD = instruction & 0xff;
   delay(delayAfterInstruction);
   PTM &= 0xfd;   // disable
   delay(delayAfterDisable);
   PTAD = 0;
 }


void InitializeLCD(void){
   delay(20);
   InstructLCD(0b0000111000, 100, 5);     // Sets to 8-bit operation, 2-line display, 5x8 unsigned character
   font
   InstructLCD(0b0000001110, 100, 5);     // Turn on display, cursor
   InstructLCD(0b0000000110, 100, 5);     // Sets to increment adrs by one & shift to right on DDRAM write
   InstructLCD(0b0000000001, 100, 100);   // Clear LCD and set address counter to 1
   InstructLCD(0b0000000010, 100, 5);     // Cursor to head of line 1
 }


void LCDOutChar(char charOut){
   InstructLCD( 0b1000000000 | charOut, 5, 5); //0x30 is the unsigned char map index
   delay(1);
 }


void LCDOutString(char* stringOut, byte length){
   byte I = 0;
   for(;I<length;I++){
   LCDOutChar(stringOut[I]);
  }
 }


void LCDOutInt(byte byteOut){
   if(byteOut>999) LCDOutChar(48+(byteOut/1000)%10);
   if(byteOut>99) LCDOutChar(48+(byteOut/100)%10);
   if(byteOut>9) LCDOutChar(48+(byteOut/10)%10);
   LCDOutChar(48+byteOut%10);
 }


void MoveCursorTo(byte CursorLocation){
```

```
    int I;
    if(CursorLocation < 16){
    InstructLCD(0b0000000010, 5, 5);
    for(I=0; I < CursorLocation; I++)
    InstructLCD(0b0000010100, 5, 5);    // Right shift cursor
   }
    else if(CursorLocation>15) InstructLCD(0b0011000000 + CursorLocation-16, 5, 5);
 }


 //---------------------------------- keypad.c ----------------------------------------------------
    #include “stdafx.h”


    extern Menu menus[TOTAL_MENUS];
    extern byte curMenu;
    extern byte keypadBuffer[KEYPAD_BUFFER_SIZE];


    // KeypadStates have one bit for each keypad input. Bitwise & with KP_X to mask into a boolean for each
    key.
    Short oldKeypadState = 0;
    short curKeypadState = 0;
    short newKeypadState = 0;


void UpdateKeypad(void){
    oldKeypadState = curKeypadState;


    PTAD = 0x00; // No Data
    delay(1);


    PTAD = 0x01; // Turn on Keypad1
    delay(1);
    curKeypadState = PTT & 0x1C;  // read the 3 status lines
    curKeypadState<<=3;
    PTAD = 0x00; // No Data
    delay(1);


    PTAD = 0x02; // Turn on Keypad2
    delay(1);
    curKeypadState |= (PTT & 0x1C);  // read the 3 status lines
    curKeypadState<<=3;
    PTAD = 0x00; // No Data
    delay(1);


    PTAD = 0x04; // Turn on Keypad3
    delay(1);
    curKeypadState |= (PTT & 0x1C);  // read the 3 status lines
```

```
    curKeypadState<<=3;
    PTAD = 0x00; // No Data
    delay(1);


    PTAD = 0x08; // Turn on Keypad4
    delay(1);
    curKeypadState |= (PTT & 0x1C);  // read the 3 status lines
    curKeypadState>>=2;
    PTAD = 0x00; // No Data
    delay(1);


    newKeypadState = curKeypadState & ~oldKeypadState;
  }

void DoKeypadActions(short doAction) {
    // Handles default keypad actions like jumping to a new menu or buffering the input.
    Byte I=0;
    if(menus[curMenu].initialCursorLocation != EMPTY){
    // Currently buffering input
    if(doAction & KP_1) BufferKeypress(1);
    else if(doAction & KP_2) BufferKeypress(2);
    else if(doAction & KP_3) BufferKeypress(3);
    else if(doAction & KP_4) BufferKeypress(4);
    else if(doAction & KP_5) BufferKeypress(5);
    else if(doAction & KP_6) BufferKeypress(6);
    else if(doAction & KP_7) BufferKeypress(7);
    else if(doAction & KP_8) BufferKeypress(8);
    else if(doAction & KP_9) BufferKeypress(9);
    else if(doAction & KP_0) BufferKeypress(0);
    else if(doAction & KP_STAR){
    if(keypadBuffer[0] == EMPTY) JumpToMenu(HIGHER_MENU);
    else{
    for(;I<KEYPAD_BUFFER_SIZE;I++) keypadBuffer[I] = EMPTY;
    DisplayMenu(curMenu);
       }

     }
    else if((doAction & KP_POUND) && keypadBuffer[0]!=EMPTY){
    JumpToMenu(PROCESS_KEYPAD_BUFFER);
    for(;I<KEYPAD_BUFFER_SIZE;I++) keypadBuffer[I] = EMPTY; // Clear Keypad Buffer after it is
    processed
     }

   }
    else // Not buffering input
   {
    if(doAction & KP_STAR) JumpToMenu(HIGHER_MENU);
    else if(doAction & KP_1) JumpToMenu(MENU_1);
```

```
    else if(doAction & KP_2) JumpToMenu(MENU_2);
    else if(doAction & KP_3) JumpToMenu(MENU_3);
    else if((doAction & KP_8) && (curMenu==14 || curMenu==22))JumpToMenu(MENU_8);
    else if(doAction && (curMenu==5 || curMenu==13 || curMenu==16 || curMenu==18 || (curMenu>=23
    && curMenu<=25) || curMenu==27 || curMenu==34))//These menus trigger on any keypress
    JumpToMenu(MENU_1);
    else if((doAction & KP_4) && curMenu==9) JumpToExactMenu(28);//Temperature menu
   }

 }


void ClearKeypadBuffer(void){
    int I;
    for(I=0; I<KEYPAD_BUFFER_SIZE; I++)
    keypadBuffer[I] = EMPTY;
 }


 //--------------------------------- sensors.c ----------------------------------------------------
    #include “stdafx.h”
    extern byte curMenu;


    bool sensorDoor = false;
    bool sensorMotion = false;
    bool sensorWindow = false;


    extern bool countingSensorTimer;
    extern byte sensorTimer;


void CheckSensors(void){
    // Smoke sensor (Handled differently on Armed and Unarmed)
    if(GetSmokeState()) {
    if(IS_ARMED) JumpToMenu(ARMED_FIRE_MENU);
    else if(curMenu!=SMOKE_MENU && curMenu!=UNARMED_FIRE_MENU)
    JumpToExactMenu(SMOKE_MENU);
   }


    // Check the armed only sensors
    if(GetDoorState()) sensorDoor = true;
    if(GetMotionState()) sensorMotion = true;
    if(GetWindowState()) sensorWindow = true;
    if(curMenu!=LOCKDOWN_MENU && IS_ARMED && (sensorDoor||sensorMotion||sensorWindow)){
    countingSensorTimer = true;
    JumpToExactMenu(6);
   }

 }
```

```
    bool GetDoorState(void){
    bool returnVal = true;
    PTAD |= 0x10; // Enable Door plunger
    delay(1);
    if((PTT & 0x10) != 0){ // Check return value
    returnVal = false;
   }
    PTAD &= 0xEF; // Disable Door plunger
    delay(1);
    return returnVal;
  }


bool GetMotionState(void){
    if(PTT & 0x40) return true;
    else return false;
  }


bool GetSmokeState(void){
    if(PTT & 0x80) return true;
    else return false;
  }


bool GetWindowState(void){
    bool returnVal = true;
    PTAD |= 0x20; // Enable Window plunger
    delay(1);
    if((PTT & 0x10) != 0){ // Check return value
    returnVal = false;
   }
    PTAD &= 0xDF; // Disable Window plunger
    delay(1);
    return returnVal;
  }


 //---------------------------------- timers.c ----------------------------------------------------
    #include “stdafx.h”


    byte phoneTimer = 0;
    bool countingPhoneTimer = false;
    byte sensorTimer = 0;
    bool countingSensorTimer = false;
    byte smokeTimer = 0;
    bool countingSmokeTimer = false;
```

```
    bool countingKeypadTimer = false;
    byte keypadTimer = 0;
    byte intCount = 0;


    extern bool sensorDoor;
    extern bool sensorMotion;
    extern bool sensorWindow;
    extern byte pickupDelay;
    extern byte smokeDelay;
    extern byte sensorDelay;
    extern bool curMenu;
    extern Menu menus[TOTAL_MENUS];
    extern byte keypadBuffer[KEYPAD_BUFFER_SIZE];


bool PhoneTimerUp(void){
    return phoneTimer >= pickupDelay;
 }
bool SensorTimerUp(void){
    return sensorTimer >= sensorDelay;
 }
bool SmokeTimerUp(void){
    return smokeTimer >= smokeDelay;
 }
bool KeypadTimerUp(void){
    return keypadTimer >= sensorDelay;
 }


void TimedInterruptHandler(void){
    int I=0;
    if(intCount==1){ // Incremented at 1Hz
    intCount = 0;
    if(countingPhoneTimer) phoneTimer++;
    if(countingSensorTimer) sensorTimer++;
    if(countingSmokeTimer) smokeTimer++;


    if(curMenu==6){
    JumpToMenu(EFFECTS_ONLY);
    if(menus[curMenu].initialCursorLocation == 0)
    while(keypadBuffer[I++] != EMPTY) LCDOutInt(keypadBuffer[I]);
     }

   }

 }


void UpdateTimers(void){
```

```
  TimedInterruptHandler();

  if(countingPhoneTimer && PhoneTimerUp()){
  countingPhoneTimer = false;
  phoneTimer = 0;
  Send(MO_PICKUP);
 }

  if(countingSensorTimer && SensorTimerUp()){
  if(sensorDoor) Send(MO_DOOR);
  else if(sensorMotion) Send(MO_MOTION);
  else if(sensorWindow) Send(MO_WINDOW);
  sensorDoor = false;
  sensorMotion = false;
  sensorWindow = false;
  countingSensorTimer = false;
  sensorTimer = 0;
  JumpToExactMenu(LOCKDOWN_MENU);
 }

  if(countingSmokeTimer && SmokeTimerUp()){
  JumpToExactMenu(UNARMED_FIRE_MENU);
  countingSmokeTimer = false;
  smokeTimer = 0;
 }

  if(countingKeypadTimer && KeypadTimerUp()){
  JumpToExactMenu(LOCKDOWN_MENU);
  countingKeypadTimer = false;
  keypadTimer = 0;
 }
}
```

Appendix B Bill of Materials

Bill of Materials

Item	Quantity	Label	Mfgr Part Number	Manufacturer/Supplier
LCD Screen	1	*	LCD1031	BGMicro
Motion Sensor	1	*	SB006	www.futurlec.com
Fire Sensor	1	*	FG 83R	www.electricsuppliesonline.com
Door/Window Sensor	2	*	SEN-3005N	GE/www.homesecurityandautomation.com
Micro-controller	2	*	MC9S12C32	Freescale/Motorola
Keypad	1	*	102	Greyhill
DTMF Decoder	1	*	CM8870	California Micro Devices
Op Amp	1	*	LM324	ON Semiconductor
5V Micro-relay	1	*	275-232	RadioShack
Microphone	1	*	270-090	RadioShack
DAA	1	*	CH1837A	Cermetek
High Current Peripheral Driver	1	*	DS3658	National Semiconductor
Text-to-Speech	1	*	RC8660	RC Systems
1/8 Audio Jack	1	*	274-333	RadioShack
150mA Fuses	2	*	*	*
3.579545 MHz Crystal Oscillator	1	*	*	*
Capacitors				
100 nF Capacitor	11	.1 uF	*	*
1 uF Capacitor	2	1 uF	*	*
10 uF Capacitor	1	10 uF	*	*
8.2 nF Capacitor	1	8.2 nF	*	*
39 pF Capacitor	1	39 pF	*	*
.25 Watt Resistors				
300kΩ Resistor	3	300k	*	*
100kΩ Resistor	3	100k	*	*
73kΩ Resistor	1	73k	*	*
1MΩ Resistor	4	1M	*	*
5.6kΩ Resistor	1	5.6k	*	*
32kΩ Resistor	1	32k	*	*
10kΩ Resistor	1	10k	*	*
1kΩ Resistor	1	1k	*	*

* N/A

Appendix C References

[1] Hitachi, Dot Matrix Liquid Crystal Display Controller/Driver, HD44780U

[2] Grayhill Standard Keypads, Series 96, www.grayhill.com

[3] DAA, Data Access Arrangement Module, V.34bis, www.cermetek.com

[4] V-Stamp, RC Systems, www.rcsys.com

[5] Double Talk RC8660, RC Systems, www.rcsys.com

[6] Quad High Current Peripheral Driver, DS3658, www.national.com

[7] CMOS Intergrated DTMF Receiver, CM8870, www.calmicro.com

[8] MC9S12C Family, Freescale, www.freescale.com

Project Website:

http://sites.google.com/a/eng.utah.edu/hocss/