Duplicate report!

68HC12 Microprocessor based Controller for a Card Reading Car Security System

Authors:
Tai Mong, Ming Deng, Reynold Tam

Microprocessor Systems
Fall 1999
December 10, 1999

Table of Contents

	Page
• Cover Page	1
• Table of Contents	2
• Abstract	3
• Introduction	3
• Materials and Methods	
- Materials	3
- Methods	3
- Card Reader	4-8
- Keypad	9
- Display	9-10
- I/O Circuitry	10-11
• Results	11
• Discussion	11-12
• Appendix A	13-14
• Appendix B	15-30
• References	31
• Bibliography	31

Abstract

Machines are capable of becoming personalized for the human user, as the world is being computerized. By incorporating a card reader with the microprocessor and some extra input/output components, a customizable security system is built for the automobile. The great aspect of this system is its expansiveness. When further developed, it can become the centralized computer system of the car itself, controlling and limiting the options of the car based on the user. It can also be used to control the maximum speed of the car. This might be a good idea for new young drivers who just got their license.

Introduction

Security systems have always been a non-stop innovative area. Many aspects of human life, ranging from monetary services to facility entry are becoming computerized. Whenever there is anything of value, security systems will be present to protect against would-be invaders. As seen on college campuses and many high tech companies, a single card can be used in many applications. The usage of the magnetic strip on plastic cards is steadily replacing the previous security technology, the mechanical keys and locks.

There are many advantages with a key card rather than a mechanical key. Cards are cheap and quick to mass-produce. It is lighter and easier to carry. As one gets more locks, the lot of mechanical keys can be a nuisance. Cards can contain more information than just for unlocking doors. A person can use a key card to enter his house and by using his/her card, a centralized computer can immediately turn on the lights to a desired luminance, favored music is played, certain temperature is set, etc.

To keep up with the trends in technology, we decided to implement a card reading system for the automobile. Our goal is to create a basic controller system that can distinguish different users, as well as detecting invalid users and entry to the car. It can also control several features of the automobile.

With the short development time, this controller only provides simple functions. But it can be used as a structure or a template for developing a more complex and comprehensive security system.

Materials and Methods

Materials:

1. Motorola 68HC12 and Evaluation Board
2. American Magnetics MagStrip™ Card Reader model# 40S5DA
3. Hitachi HD44780 LCD and Controller

4. 16-button Generic Keypad
5. 74LS04 Inverter
6. Light Emitting Diodes 4x
7. Various Resistors
8. Piezo Buzzer
9. Array of switches

Methods:

This project is divided in four part: the card reader, the keypad, the display, and some input / output circuitry. Each part was developed separately so operates freely as a component that can be added to an existing system. The methods for each component are discussed below.

The Card Reader

The magnetic card reader of the project was used to quickly get the identification number of the rightful user of the vehicle. By using a card reader, we save the user the time and memory to punch in their ID number into the keypad.

Hardware

For the hardware of the magnetic card reader, we used the American Magnetic MagStripe Card Reader. The American Magnetic MagStripe Card Reader can read two tracks on a magnetic card. For our project, we just needed to read one track. Four connections were needed to be made to the magnetic card reader. Two of these connections were ground and power of *5 volts*. The other two connections were the data connection, and the clock connection. For our particular project, we used *Pin 0* of *Port G* to sample the data and used *Pin 7* of *Port T* to check the clock pulse from the card reader.

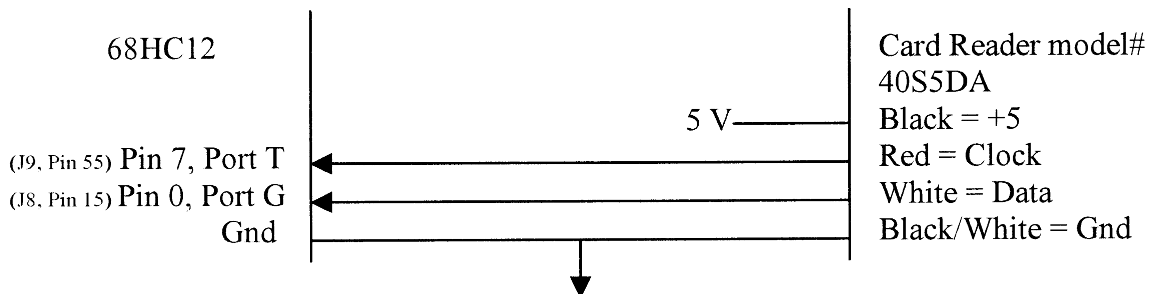

Figure C.1

The mechanism behind the magnetic card reader is simple. As the user swipe the card through the reader, there will be a 50 usec drop in the normally high clock signal.

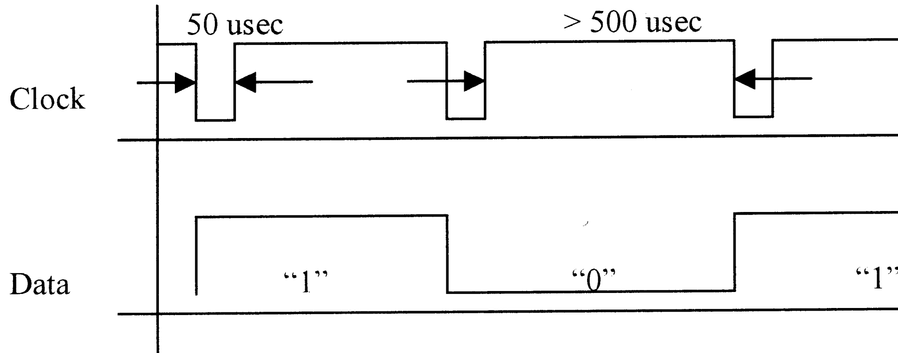

Figure C.2

The ideal time to sample the data would be on the falling edge of the clock signal. Fortunately, the 68HC12 has a Pulse Accumulator interrupt that can be triggered on a falling or rising edge. Pin 7 of Port T was set to trigger the Pulse Accumulator interrupt on a falling edge.

Software

The software portion of the card reader was more complicated then the hardware. After deciding to use the Pulse Accumulator interrupt service sub-routine to sample the data bit on *Pin 0 of Port G* at the falling edge of the Clock signal the next thing we had to do was deciding on how to store the bits and how to get human legible data from it.

Before we can start on coding to get human legible data from the magnetic card, we must understand the structure, which the digits are stored in. The digits are stored in binary codes in the magnetic strip of the card. Each digit is stored by 5 bits, the first 4 bits (Least significant bit first) stores the binary representation of the decimal digit and the 5th bit is the parity bit. The beginning and the end of the magnetic strip usually contain padding. The beginning bit of relevant data bits is after the delimiting bit pattern '11010' or 0xB in hexadecimal. There is also an end delimiter; the last 5 relevant bits are the 5 bits after the delimiting pattern of '11111' or 0xF in hexadecimal.

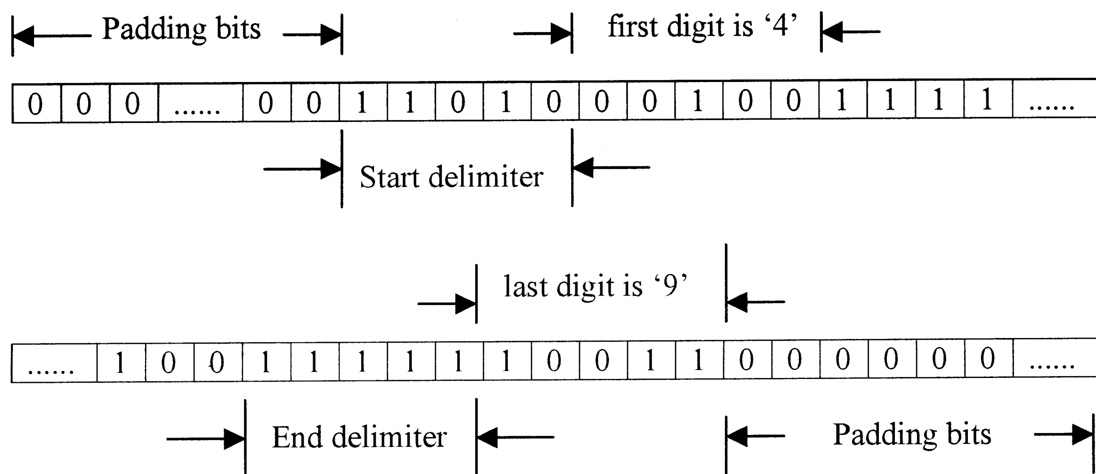

Figure C3

To store the bits, we decide to use an array of 'char.' We decided to use an array of 'char' because 'char' is the smallest data type and we only needed to store a '1' or an '0'. The array we used was *CDbitStream[256]*, we choose a size of 256 because most magnetic card can store up to 16 digits (16x5 = 80, plus some padding). We tried using 128 but that was not enough to feed all the data.

After deciding on a data structure to store the bits, the next thing is deciding how to obtain the bits. As I mentioned in the hardware section, we decided to use an Interrupt Sub-Routine (ISR) to sample the data bit when triggered by the falling edge of the clock signal from the card reader. The ISR we used was *cardReader_PAEdgeInt()*. We tried to keep the ISR as short as possible to make sure the ISR is complete before the next clock cycle. To keep the ISR short, the ISR is only responsible for getting a bit and storing it in *CDbitStream*. After the ISR stores a reasonable amount of bits (currently set to 40), it will start checking for the end delimiter each time it is called. After reading 5 bits pass the end delimiter, the ISR will start calling a chain of helper methods within the *cardReader.h* to help do error detection, manipulate binary data into decimal, and validate the card swiped.

After reading the end of the relavent bits, the ISR calls the helper method *cardReader_extractID()*. The *cardReader_extractID()* is responsible for doing odd-parity check on the bit streams. Odd parity check is done by mod 2 the summation of each bit in each 5 bit code words and checking it see if it equals to one. If the bit stream passes parity check, it is then converted to decimal digits and stored in the *CDdigitArray*. Coverting from binary to decimal was accomplish by multiplying each of the bits in each code word by their weights (i.e.. 1, 2, 4, and 8) and adding them up. The helper methods, *cardReader_helper_findStartPos()* and *cardReader_helper_findEndPos()* was called by *cardReader_extractID()* to help find the starting and ending position in the raw bit stream to perform the above operations on. After extracting the decimal ID number, the ISR calls the *cardReader_validateCard()* method to check to see if this card has permission to

disarm and operate the security system. If the card has access to the system, the ISR will set the global variable *CDarmed* to 0 and then display the menu on the LCD screen.

The following is a flow chart of the general operations performed by the *cardReader.h*. For more detail on the implementations, please read the attached codes at the end of this document. The attached codes contain detailed comments on the algorithms and logics used in implementing the card reader part of our security system.

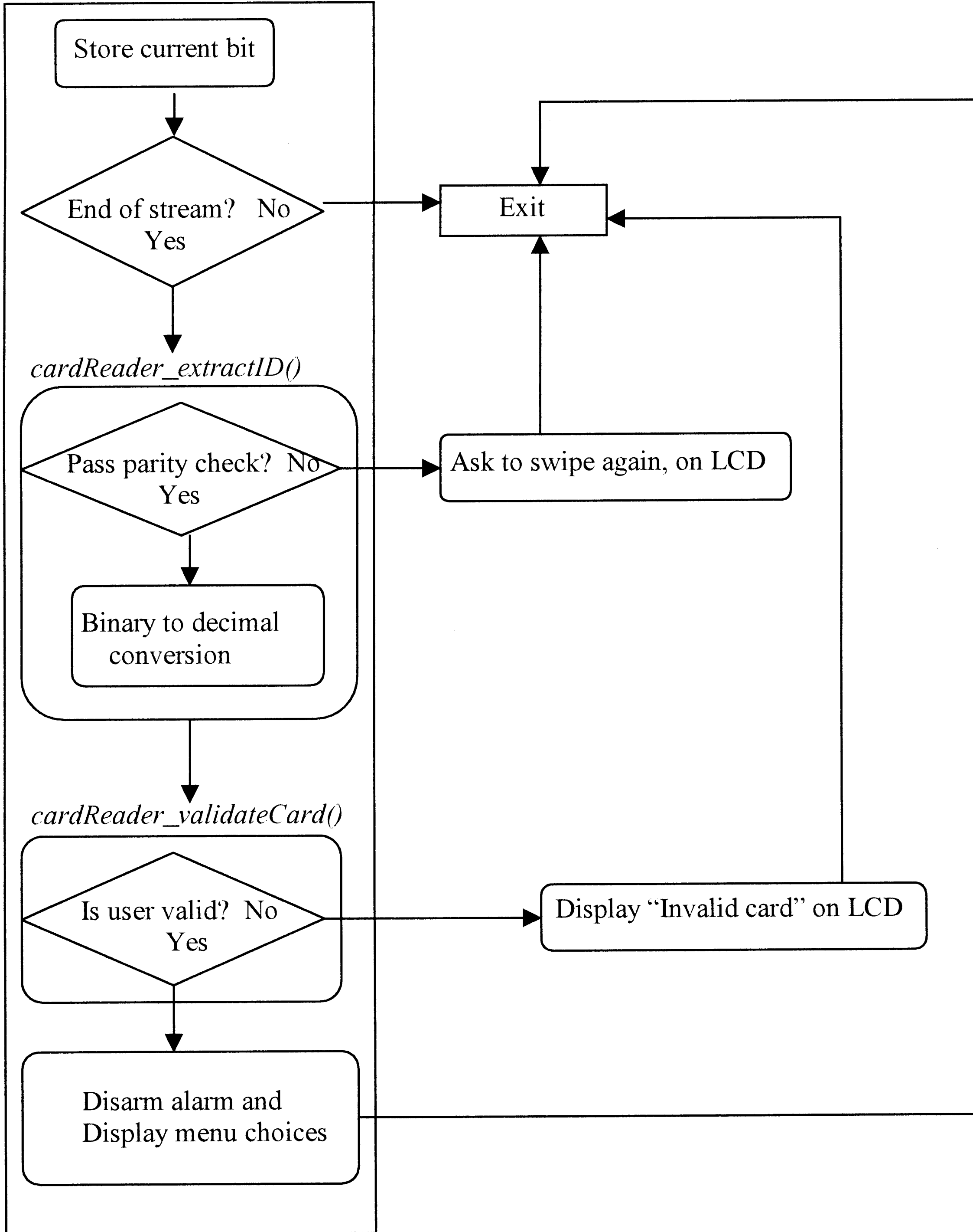

Figure C.4

The Keypad

The wiring schematics for this component can be found in Appendix A. Referring to Figure W1, we can see that the keypad utilizes the entire Port J. On the software side, four functions are implemented: *keypad_init()*, *keypadJ()*, *getrow()*, and *getcolumn()*.

The *keypad_init()* function prepares the keypad to be controlled by the HC12. It sets the interrupt to be triggered by falling edges, select and enable pull ups, enables all bits of Port J to be used by the keypad, sets the data direction of the MSB for output, and sets the LSB as input. Please refer to the code of *keypad.h* in Appendix B for the specific values that are set to the registers.

The keypad utilizes the Port J interrupt. Once a key is pressed, the system jumps to the *KeypadJ()* interrupt service routine. The routine, tests for the row and column of the keypad, and from these two values, the corresponding key is determined.

The function get*row()* works by reading the values of the LSB from Port J. A high value on a specific bit directly corresponds to the row. So a simple *switch* statement is used to determine the row in which the key lies.

The function *getcolumn()* is a bit more complex. A high needs to be sent to each of the MSB to determine the column. If all of the LSBs goes low following the high bit, then the column is the bit that was tested. Bit 7 corresponds to column 4, bit 6 to column 3, and so forth. For example, if a high is sent to bit 7, and all of the LSB goes low immediately after, then that signals that is the 4th column.

The Display:

The display has 14 connections between itself and the microprocessor. Pin 1 goes to power **while pin 2 goes to ground. Pin 3 is connected to the wiper of a 10KΩ** potentiometer that controls the contrast. Pin 4 is the register select of the LCD screen. This is used to select between the instruction register or the address counter of the HD44780. Pin 5 controls the Read/Write select. Whenever the input to the pin is high, it is in read mode and when it is low, it is in write mode. Pin 6 is the LCD enable and is used to clock data and instructions of the LCD display. Pins 7 to 14 are the data pins. Pin 14 also handles the Busy Flag for the HD44780. The wiring diagram is included in the Appendix as Figure W2.

The C code was written as a header file that could be included in the program that interfaces with an LCD screen. The header file uses Port H for writing data and Port G for the control signals. Within the header file, it contains six functions to manipulate the LCD screen: *OpenXLCD*, *SetDDRamAddr*, *BusyXLCD*, *WriteCmdXLCD*, *WriteDataXLCD*, and *WriteBuffer*. These functions are basic features to display data and to position the cursor. *OpenXLCD* executes all the initialization routines. *SetDDRamAddr* sets the address of the cursor. BusyXLCD is used to check the Busy Flag from pin 14. WriteCMDXLCD is used to write commands to the LCD screen.

WriteDataXLCD is used to write data to the LCD. *WriteCMDXLCD* and *WriteDataXLCD* is very similar except that *WriteDataXLCD* must be written to display as ASCII character. *WriteBuffer* is a command used to write a string of characters that is stored in buffer to the LCD screen. It basically loops and calls the *WriteDataXLCD* function until the entire buffer has been transmitted.

We used *display.h*, which is a simplified version of *lcd.h*. It uses some of the functions from *lcd.h*. *void display_init(void)* is an initialization function and it also contains a function to clear the display of the LCD screen. By passing a *0x01* to the *WriteCmdXLCD* function, the display of the LCD is cleared. *void display_send1(char *buffer)* includes *WriteCmdXLCD* and *WriteBuffer* functions. It basically sets the cursor at the beginning of the top row of the display and then writes to the screen from buffer. This is the same for *void display_send2(char *buffer)* only that this writes to the second line of the screen.

I/O Circuitry:

For the I/O circuitry, we included a few components to interface and show certain states to the user. For outputting information, we used 5 LEDs and a buzzer. There are three LEDs that represent different functionalities of our system, one LED to show an alarmed state, another LED for our alarming state, and finally a buzzer as a siren.

We designated an output data direction for Port S's S2 to S6 on the EVB. To set the data direction, *_H12DDRS = 0x7F;* was used in our code. S2 to S6 all are connected to the **Motorola 74LS04 Inverter and then to 1kΩ resistors. Connected to the resistors are the** individual LEDs that are tied to *5 volts*. See Figure W3 in Appendix for the wiring schematic of our output system. S2, S3, and S4 are the three LEDs that display the functionality. Presently our code has the three LEDs with the following functionality: unlocking hood, unlocking and then locking doors, and unlocking trunk. S5 controls the blinking light to show an armed security system. S6 is the alarm system. It has an LED and a Piezo Buzzer connected in parallel. This will cause the LED to light and the buzzer to sound if the alarm is tripped.

Since one end of the LED is connect to *5 volts*, it will always be high. When a user presses one of the buttons on the keypad to activate a function, a high signal (i.e. - *_H12PORTS = 0x08*) will be generated to the respected Port S. It will be converted to a low signal after it passes out of the Inverter. The difference in voltage between the LED will cause it to light. This goes for the buzzer as well. Whenever there is a voltage difference between the positive and negative terminals, the buzzer will sound.

The buzzer is used is used in conjunction to the break-in detector. Break-in detectors work by running a continuous current along a suspected break-in area, normally that would be a glass window. When the window is broken, the continuity will break and the alarm is tripped. This signifies as a break-in. For our system, we wired *5 volts* going through four switches that are in series, to a resistor and then to ground. Branching off between the last switch and the resistor is a connection going into S7 of Port S. See figure W4 in appendix for the schematics. The data direction of S7 is set for input.

When the switches are closed, which signifies that the system is not broken into, a constant current is directed to the port on the EVB. Once there is a break-in (i.e. any of the switches are opened), then the input is pulled low to ground. This triggers the code to generate a high output at S6 causing the Piezo Buzzer to sound and the LED connected in parallel to the buzzer to light. This is our alarming state and will remain at this state until a valid user swipes his/her card. The system checks the continuity of S7 only if our alarm state variable, *CDarmed*, is set to 1. Otherwise, it will ignore the position of the switches by not utilizing S7.

Results

After performing several tests aimed at crashing/breaking the system, we did not find any flaws or glitches. It basically gives permission to certain users and presents the user with options afterwards.

The user swipes his/her card and depending on the card, grants admission into the system. For our current system, we have coded in two master users and the system can distinguish between the two master users and non-valid users. If permission is allowed, the user is presented with a menu on the LCD display that describes the choices he/she can access. Using the keypad, the user enters the desired option or by pressing any other key will display the menu again. The results will also be displayed on the LCD. Unfortunately, we do not have an actual car to implement with our system. We therefore have LEDs that act as if an option happened. The idea is to show that the EVB is in fact outputting. Whenever the user prefers to arm the system again, he/she may do so by pressing the letter "D". The system will be in its armed state and a red blinking light will reassure us of its status. At this point, the keypad becomes inoperable. We also implemented a break-in detection feature. A series circuit with switches is connected to an input port on the EVB. This series circuit has continuity and therefore means the system is not being broken into. If this link is broken (an open circuit), the buzzer will sound and a LED will turn on. This is the system's alarm state

Discussion

Originally from the start, we had a totally different idea for our project. We initially proposed a stock ticker project where the code would grab information about certain stocks and display them on a LCD display. After careful consideration and discussion with Prof. Kelley, we determined that the stock ticker would be a bad choice for our final project. It would be too software oriented and would not utilize majority of the 68HC12's abilities.

To be different, we wanted to use the card reader. Further brainstorming persuaded us to consider a security system. A user can swipe his/her card and the database would permit match the card number and grant certain permissions. This project will make use of the card reader, LCD display, and the keypad. As we started to work with designing the hardware and software, we were informed that our project needed to be more specific.

Creating a security system is too vague and can be interpreted as any security system. Our final project has to be of something specific and practical.

Without deciding to rethink and redesign our project again, we chose to remain with a security system idea but needed to narrow down our objectives. Careful reevaluation of the tasks involved and the time we had, the decision to design a Car Security System was reached. It includes all the hardware components we wanted to use and it is focused enough to be a marketable product.

Given more time for development, we would include more functionality. One of the main objectives we believe should be completed is to be able to control and grant specific access. After access is granted, certain permissions are allowed. Once we completed this objective, we then move on to more options for extra security. We would like to include a pin input along with swiping a card for access. This will put a larger obstacle for anyone trying to break into the system. We would like to include more options the user has command over. With a pin access, personalization can be achieved. There could be different cards for different users. This in turn, can lead to many more options. The user could set chair settings, mirror settings, temperature settings, etc. That information could be stored in a record and retrieved when that specific user uses his/her card and inputs the individual pin number. Currently, we only have three options: unlocking of hood, doors, and trunk. We might implement a more variety of functionality given more time.

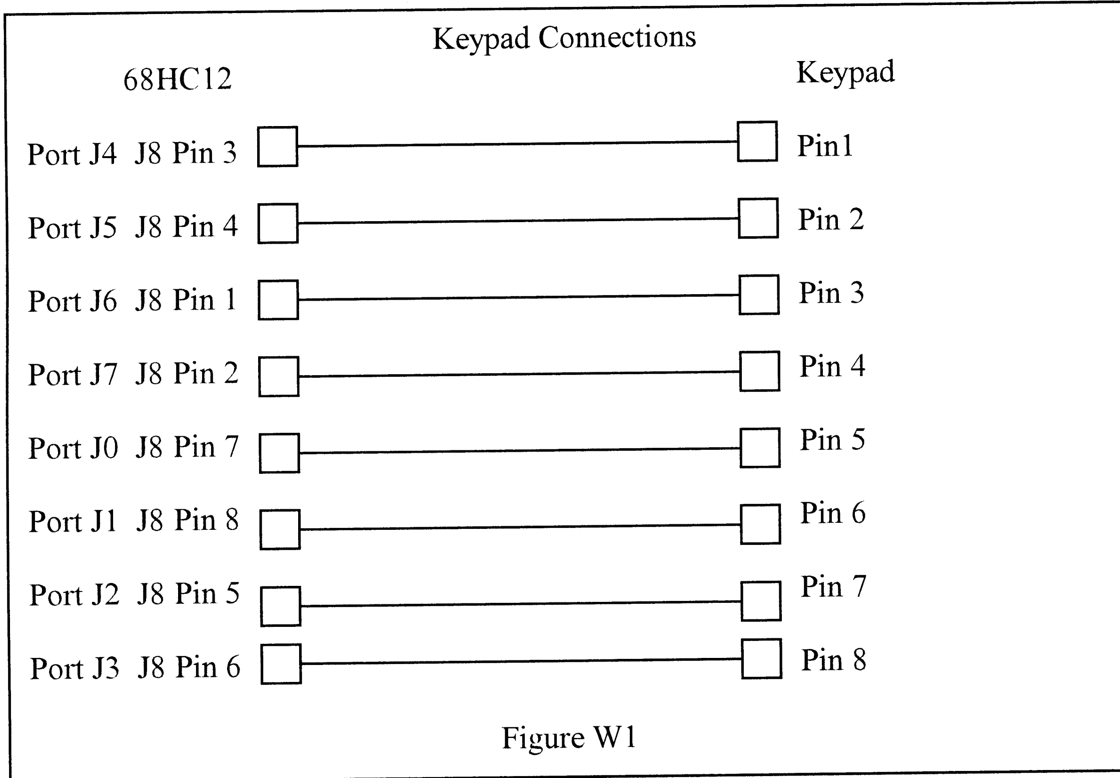

Figure W1

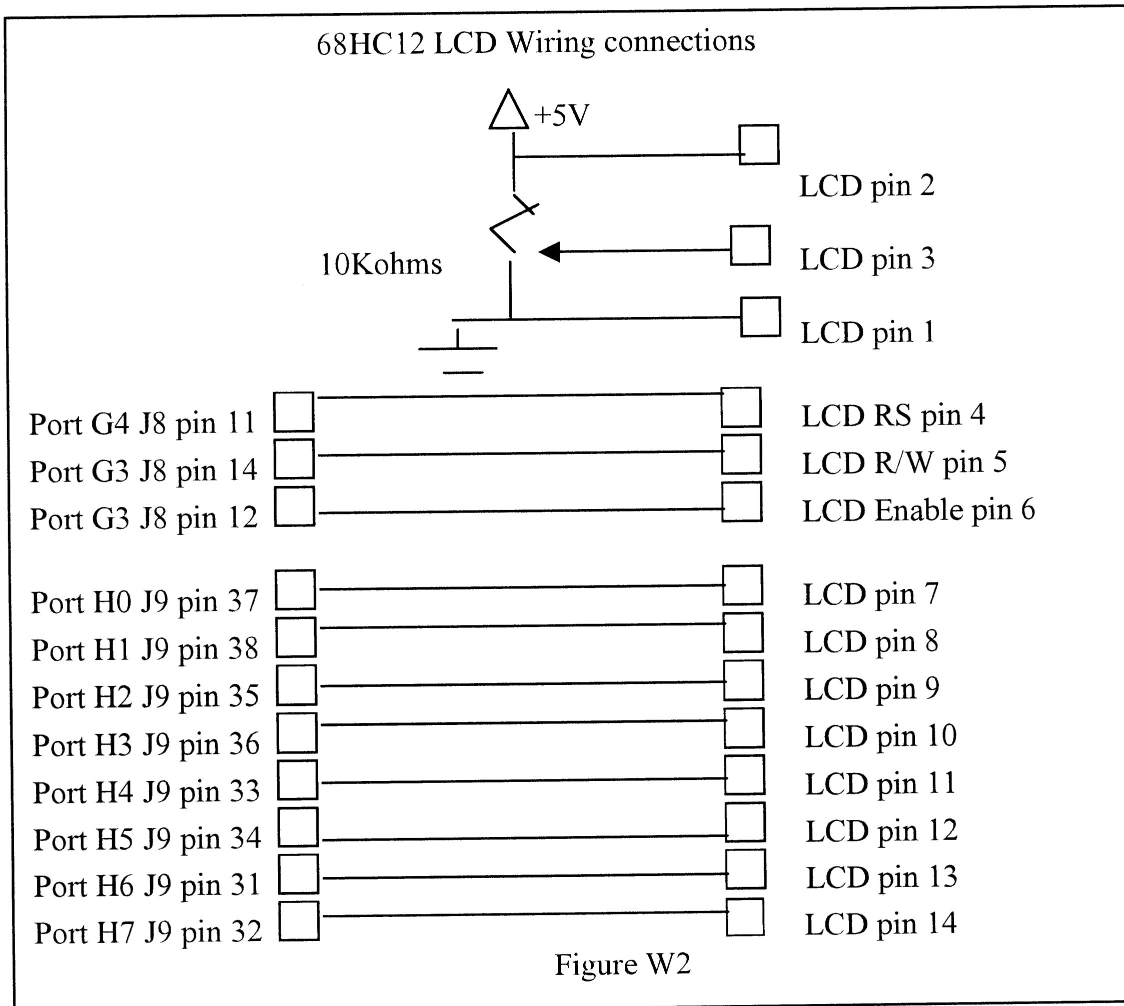

Figure W2

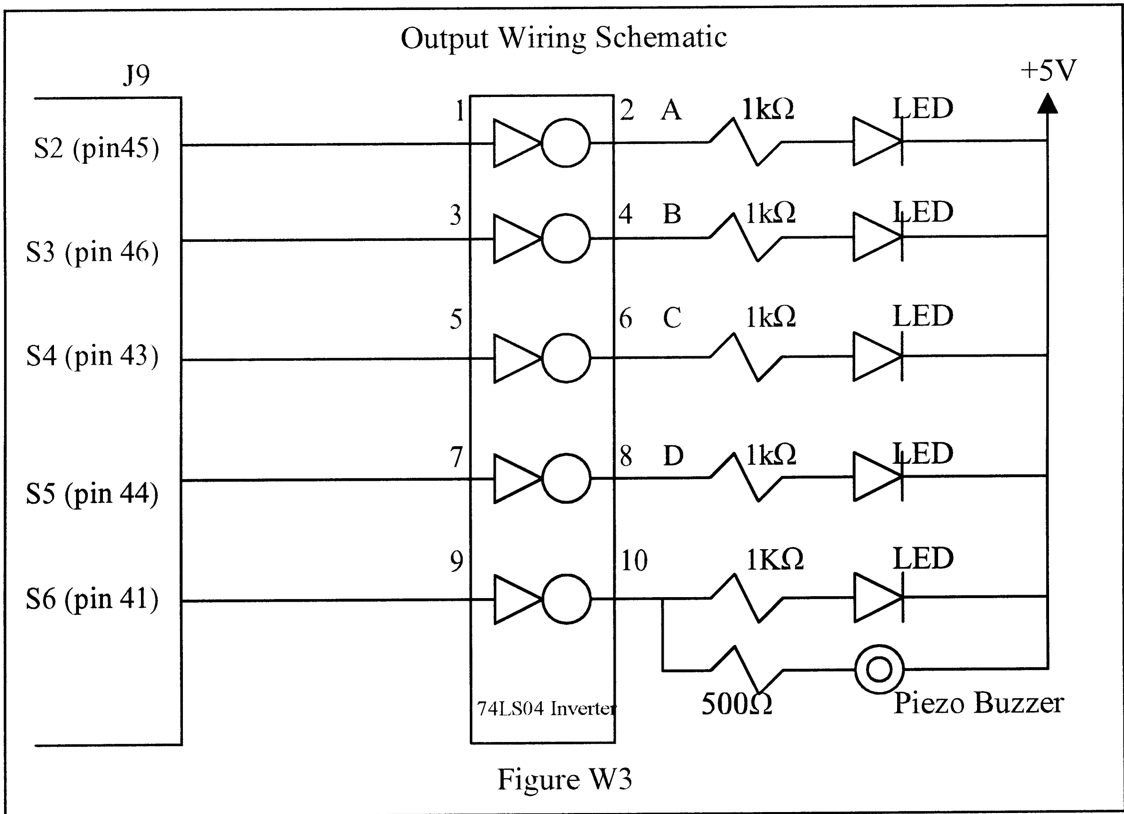

Figure W3

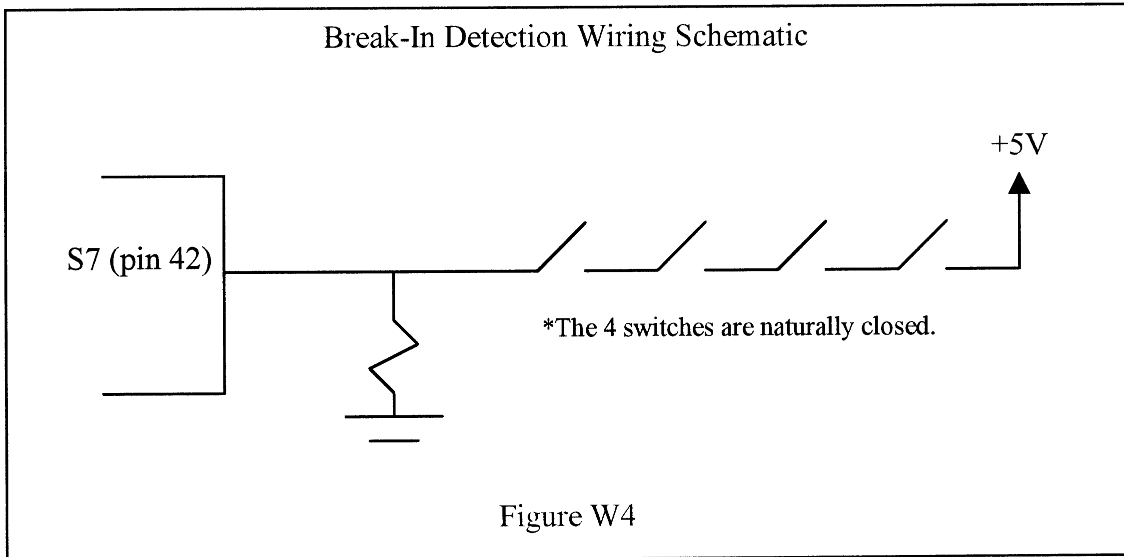

Figure W4

Appendix B. C code

cardReader.h

```
////////////////////////////////////////////////////////////////
//
// File:        cardReader.h
// Discription:  Header file containing methods to access the
//              magnetic card reader
// Authors:      Ming Deng, Tai Mong, Reynold Tam
// Date:        November 1, 1999
//
// Tasks requirements:
//   1. Real Time Interrupt sub-routine to pick up a bit on a clock
//      signal falling edge.
//   2. Shift bits into a memory word. When the parttern is $B
//      (%11010) appears, set up a counter to count bits modulo 5
//   3. After each 5 bits arrive, check the parity bit for odd parity,
//      and if all is well, store the 4-bit BCD code in a buffer and
//      exclusive-or the 5 bit pattern into a memory word, otherwise
//      record an error.
//   4. After the stop pattern $F (%11111) arrives, pick up exactly
//      one more word, and then compute the even parity across the bit
//      positions verified by the message parity character.
//   5. If any errors occur, put out a message "Bad Read", otherwise
//      convert the data stored in the buffer to ASCII and print it
//      out, followed by a carriage return.
//
// Approach:
//   1. Have the Real Time Interrupt (RTI) routine do minimum work
//      Just read in the data bit and attached it to a bit stream
//      when the RTI routine is triggered
//   2. Parse the byte stream using the beginning delimiter and
//      finish delimiter
//   3. Have a method scan and do parity checking on the final
//      byte stream
//   4. Change the binary data into a array of decimals
//   5. Check to see if the ID number of card swiped is a good
//      user of the system
//   6. Only interface is method:
//      int CDarmed = 1 if card swiped is not valid
//          CDarmed = 2 if card swiped is good
//
// MicrProcessor Systems Fall 99
// Ming Deng, Tai Mong, Reynold Tam
//
// NOTE: This code was written to work with the American Magnetics
//      MagStripe (TM) Card Reader. With with Clock connected to
//      port T, pin 7 of the EVB board and Data bit is attached to
//      port G, pin 0 of hte EVB board.
//
////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////
//
// REVISION INFO:
//
// 12/8/99  - change the interface to the cardReader..outside
//          procedures only needs to check if CDarmed is 0 or 1
//          cardReader_validateCard() is used to check if the card
//          is valid
//                                  -MD
//
// 11/19/99 - Added this revision thing, realized Tai or Reynold might
//          need to edit/modify this file when I'm not around
//
//          Change the interface to the cardReader to
//          void cardReader_getID(int* numDigit, int idNum[])
//
//          Added preprocessor states to easily disable or enable
//          debugging printf statements
```

```
//                                                  -MD
//
//
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
//
// Access users id Numbers
//
// For our current system we will use Price Chopper id numbers.
//
// Ming   Deng 44024280846
// Tai    Mong 44024280507
// Reynold Tam ???????????
//
////////////////////////////////////////////////////////////////////////

#ifndef KEYPAD_H
#include "keypad.h"
#endif

#ifndef DISPLAY_H
#include "display.h"
#endif

#ifndef CARDREADER_H
#define CARDREADER_H
// enable printf debugging statements
//#define CD_DEBUG


////////////////////////////////////////////////////////////////////////
//
// forward declaration
//
////////////////////////////////////////////////////////////////////////

// the following 2 methods will be defined in main.c
void menu(void);
void delay(void);

__mod2__ void cardReader_PAEdgeInt(); // Pulse Accum interrupt routine
void cardReader_init(); // initialize the hardware, called in main
void cardReader_reset(); // reset the variables and state of the reader
int cardReader_helper_findStartPos(); // helper methods used by this
int cardReader_helper_findEndPos();  // header file only
int cardReader_extractID(); // method to manipulate the bitstream into
                        // good human readible data
// interface to this header
int cardReader_validateCard(); // check if this card is valid user

// used only if we are debuggin
#ifdef CD_DEBUG
void dumpID(); // test method for debuggin, to be removed later
void dumpBS(); // test method for debugging. to be removed later
#endif



////////////////////////////////////////////////////////////////////////
//
// Specific variables for the card reader methods
//
// Initially I use a struct CardData but somehow the compile
// did not like structs so we just add prefix CD to each of the
// variables to signify that the variable belongs to this cardReader.h
//
////////////////////////////////////////////////////////////////////////
// store the current position in the bit stream
```

```
int CDcurrentPos;

// an array to store the streams of bits
char CDbitStream[256];

// array to store idNum as array of digits
int CDdigitArray[20];

// flag if the security system is armed or not.
int CDarmed;



/////////////////////////////////////////////////////////////////////
//
// METHOD DEFINITIONS
//
/////////////////////////////////////////////////////////////////////

// method to initialize the the Pulse accumulator and set
// the Pulse Accumulator Edge trigger routine
void cardReader_init()
{
#ifdef CD_DEBUG
  DB12->printf("\n\rEntering cardReader_init()");
#endif
  // assign the subroutine to be triggered by the Pulse Accum
  DB12->SetUserVector(PAEdge, cardReader_PAEdgeInt);
  // pulse accumulator enable, triggered by fallling edges
  // on Port T bit 7, PA clock rates set to 8MHz
  _H12PACTL = 0x45;
  CDarmed = 1;
  key = 'y';
#ifdef CD_DEBUG
  DB12->printf("\n\rExiting cardReader_init()");
#endif
}



// resets the cardData structure
void cardReader_reset()
{
  int c;
#ifdef CD_DEBUG
  DB12->printf("\n\rEntering cardReader_reset()");
#endif
  CDcurrentPos = 0;
  for (c=0; c<256; c++)
  {
    CDbitStream[c] = '3'; // signify a error, 1s and 0s are valid
  }
  for (c=0; c<20; c++)
  {
    CDdigitArray[c] = 11; // valid digit is only from 0-9
  }

#ifdef CD_DEBUG
  DB12->printf("\n\rExiting cardReader_reset()");
#endif
  _H12PAFLG=0x01; // clear the Pulse Accum flag
  return;
}



// this method will be the real time interrupt subroutine to be
// called when triggered by the clock pulse on P7PT of EVB board
// all this does is push the bit from Port G bit 0 into the
// cardData's bitStream array and check when done reading card
```

```
__mod2__ void cardReader_PAEdgeInt()
{
 // make sure Port G bit 0 is set for input
 _H12DDRG = 0xFE;
 // get bit from Port G bit 0 here  **PIN 15**
 if ((_H12PORTG & 0x01) == 0x01)
 {
  CDbitStream[CDcurrentPos] = '1';
 }
 else
 {
  CDbitStream[CDcurrentPos] = '0';
 }

 // done reading all the valid bits on the magnetic strip yet?
 // checking for the end delimiter of the current bit stream
 if(CDcurrentPos > 40)
  if(CDbitStream[CDcurrentPos - 5] == '1')
   if(CDbitStream[CDcurrentPos - 6] == '1')
    if(CDbitStream[CDcurrentPos - 7] == '1')
     if(CDbitStream[CDcurrentPos - 8] == '1')
      if(CDbitStream[CDcurrentPos - 9] == '1')
      {
       // set key (var from keypad.h) to y as a flag not
       // to display menu on LCD yet.
       key = 'y';
       if(cardReader_extractID() == 0) // parity check
       {
#ifdef CD_DEBUG
        DB12->printf("\n\rError, can not extract ID");
        DB12->printf("\n\rPossible parity error");
#endif
        display_clear();
        display_send1("Swipe again");
                cardReader_reset();
        // FUTURE print out error on LCD
        // "Please swipe card again"
        return;
       }
           if (cardReader_validateCard() > 0)
           {
            if (cardReader_validateCard() == 1)
                {
                 display_clear();
                 display_send1(" Hello Ming " );
                 delay();
                }
                if (cardReader_validateCard() == 2)
                {
                 display_clear();
                 display_send1(" Hello Tai " );
                 delay();
                }
        // valid user has been verified at his point
        // so disarm alarm and display menu
              CDarmed = 0;
        menu();
           }
           else
           {
                display_clear();
                display_send1(" Invalid Card! ");
                display_send2("  Swipe again  ");
                CDarmed = 1;

                _H12RTICTL = 0x00; //Disable Timeout
        // reset the time out so system will not
        // timeout prematurely
                TimeCount=0;
           }
```

```
#ifdef CD_DEBUG
          dumpBS();
          dumpID();
#endif
          cardReader_reset();
          return;
        }
  CDcurrentPos++;
  _H12PAFLG=0x01; // clear the flag
}



// helper method to find the starting position of valid idNumber
// this method assumes there are bit paddings on the magnetic card
// meaning that valid data bits do not start in the very beginning
// * return value of 0 indicates error
int cardReader_helper_findStartPos()
{
  int c;
#ifdef CD_DEBUG
  DB12->printf("\n\rEntering cardReader_helper_findStartPos()");
#endif
  //  DB12->printf("\n\rEntering findStartPos()");
  // find the starting delimiter (11010)
  for (c=0; c<256; c++)
    if (CDbitStream[c] == '1')
     if (CDbitStream[c+1] == '1')
           if (CDbitStream[c+2] == '0')
             if (CDbitStream[c+3] == '1')
               if (CDbitStream[c+4] == '0')
               {
#ifdef CD_DEBUG
                 DB12->printf("\n\rStart Position : %d, %d", c, c);
          DB12->printf("\n\rExiting cardReader_helper_findStartPos()");
#endif
                 return c+5;
               }
#ifdef CD_DEBUG
  DB12->printf("\n\rExiting cardReader_helper_findStartPos()");
#endif
  return 0;
}



// helper method to find the ending position of the valid idNumber
// * return value of 0 indicates error
int cardReader_helper_findEndPos()
{
  int c;
#ifdef CD_DEBUG
  DB12->printf("\n\rEntering cardReader_helper_findEndPos()");
#endif
  // find the ending delimiter (11111)
  for (c=0; c<256; c++)
    if (CDbitStream[c] == '1')
     if (CDbitStream[c+1] == '1')
           if (CDbitStream[c+2] == '1')
             if (CDbitStream[c+3] == '1')
               if (CDbitStream[c+4] == '1')
               {
#ifdef CD_DEBUG
                  DB12->printf("\n\rEnd Position : %d, %d", c, c);
           DB12->printf("\n\rExiting cardReader_helper_findEndPos()");
#endif
                      return c+5;
               }
#ifdef CD_DEBUG
```

```
  DB12->printf("\n\rExiting cardReader_helper_findEndPos()");
#endif
  return 0;
}



// method to extract actually idNum from the bitStream
// also does odd parity checking
int cardReader_extractID()
{
  int c;
  int last;
  int currentDigit;
  currentDigit = 0;
#ifdef CD_DEBUG
  DB12->printf("\n\rEntering cardReader_extractID()");
#endif
  last = cardReader_helper_findEndPos();

  for (c=cardReader_helper_findStartPos();
       c<last;
       c = c+5)
  {
   // add the 5 bits and make sure we get an odd number of 1s
   // if odd parity check fails return 0;
   if ((CDbitStream[c] - '0')
            + (CDbitStream[c+1] - '0')
            + (CDbitStream[c+2] - '0')
            + (CDbitStream[c+3] - '0')
            + (CDbitStream[c+4] - '0')%2 == 0)
     return 0;

   CDdigitArray[currentDigit] = (CDbitStream[c] - '0')*1 +
                           (CDbitStream[c+1] - '0')*2 +
                           (CDbitStream[c+2] - '0')*4 +
                           (CDbitStream[c+3] - '0')*8;
   currentDigit++;
  }
#ifdef CD_DEBUG
  DB12->printf("\n\rExiting cardReader_extractID()");
#endif
  return 1;
}

#ifdef CD_DEBUG
void dumpID()
{
  int c;
  cardReader_extractID();
  DB12->printf("\nID Number is :\n\r");
  for (c=0; c<20; c++)
  {
           if(CDdigitArray[c] < 10)
             DB12->printf( "%d, **********%d\n\r", CDdigitArray[c], CDdigitArray[c] );
  }
}
#endif


#ifdef CD_DEBUG
void dumpBS()
{
  int c;
  char d;
  DB12->printf("\n\n\rBITSTREAM :");
  for (c=0; c<256; c++)
  {
   d = CDbitStream[c];
   if (d=='0') { DB12->printf("0"); }
```

```
   else if (d=='1') {DB12->printf("1"); }
  }
}
#endif

// a very system specific method.
// leave it for time being..
// must use different approach in production
// Ming    Deng 44024280846
// Tai     Mong 44024280507
// Return  1 if Ming 2 if Tai
int cardReader_validateCard()
{
  if (CDdigitArray[0] == 4)
    if (CDdigitArray[1] == 4)
      if (CDdigitArray[2] == 0)
        if (CDdigitArray[3] == 2)
          if (CDdigitArray[4] == 4)
            if (CDdigitArray[5] == 2)
              if (CDdigitArray[6] == 8)
                if (CDdigitArray[7] == 0)
                  if (CDdigitArray[8] == 8)
                          {
                            if (CDdigitArray[9] == 4)
                              if (CDdigitArray[10] == 6)
                          return 1;
                          }
                          else if (CDdigitArray[8] == 5)
                          {
                            if (CDdigitArray[9] == 0)
                              if (CDdigitArray[10] == 7)
                                      return 2;
                          }
  return 0;
}

#endif
```

display.h

```
//
// Household Security System
// Micro-Processor Systems:  Final Project
//
// File:  display.h
//
// Purpose: Interfacing Hitachi HD44780 to Motorola to 68HC12
//                    Simplification of display commands from "lcd12.h"
//
// Author(s): Tai Mong, Mind Deng, Reynold Tam
// Revision:  11/1/1999
//
// Notes:
//
//////////////////////////////////////////////////////////////////////

#ifndef DISPLAY_H
#define DISPLAY_H

#include "lcd12.h"              /* Include built in LCD functions */

//////////////////////
// Function Prototypes
//////////////////////
void display_init(void);                // Init Prototype

void display_clear(void);               // Clear display
void display_send(char *buffer);        // WriteBuffer command

void display_send1(char *buffer);       // WriteBuffer 1st column
```

```
void display_send2(char *buffer);         // WriteBuffer 2nd column
void display_shiftleft();

/////////////////////
// Functions
/////////////////////

void display_init(void)                   // Initialization Function
{
 OpenXLCD(0x80);          /* Init the LCD */
 display_clear();        /* clear and reset */
}

void display_clear(void)    // Clear LCD display
{
 WriteCmdXLCD(0x01);                      /* Clear LCD screen */
}

void display_send(char *buffer)
{
 WriteBuffer(buffer);
}

void display_send1(char *buffer)/* Write on first line */
{
 WriteCmdXLCD(0x80);               /* Set cursor to 1st column, 1st cell */
 WriteBuffer(buffer);
}

void display_send2(char *buffer)/* Write on second line */
{
 WriteCmdXLCD(0xC0);               /* Set cursor to 2nd column, 1st cell */
 WriteBuffer(buffer);
}

void display_shiftleft()
{
 WriteCmdXLCD(0x18);               /* shift text left */
}

#endif
```

keypad.h

```
////////////////////////////////////////////////////////////////////////
// Household Security System
// Micro-Processor Systems Final Project
//
// File:     "keypad.h"
// Author(s): Tai Mong, Mind Deng, Reynold Tam
////////////////////////////////////////////////////////////////////////
//
// Purpose: Interfacing Keypad to Motorola 68HC12
//
// Revision:
//   12/7/1999  Slightly modified to work with our security system
//                                                                      -TM
//
// Notes:  This code could be more efficient, but it was common practice
//         to reuse existing and functional code rather than recreating it.
//
////////////////////////////////////////////////////////////////////////
#ifndef KEYPAD_H
#define KEYPAD_H

#include "dbug12.h"          // Debug 12 functions
#include "hc812a4.h"                    // Hc12 stuff

//////////////////////
// Global Variables
```

```
//////////////////////
char key;                       // Actual key being pressed in ascii
char temp;                      // Temporary dummy variable

int TimeCount;                  // keeps track of Real Time for RTI
int column, row;                // stores column and row number of keypad

//////////////////////
// Function Prototypes
//////////////////////
__mod2__ void KeypadJ(void);    // ISR Prototype
void keypad_init(void);         // Init Prototype
void getrow(void);              // Internal function to determine row
void getcolumn(void);           // Internal function to determine column

/////////////////////
// Functions
/////////////////////
__mod2__ void KeypadJ(void)     // ISR for keypad using port J of HC12
{
 temp = _H12PORTJ;              // Save current state of portJ
 TimeCount = 0;                 // Reset Real time count

 getrow();                      // Sets the row
 getcolumn();                   // Sets the column

 if (row==0x01)                 // Based on the row and column,
 {                              // we determine the key that is pressed
  switch(column)
  {
  case 0x10:                    // 1st row from top, 1st column from left
  key = 'A';
  break;
  case 0x20:                    // 1st row, 2nd column
  key = '3';
  break;
  case 0x30:                    // 1st row, 3rd column
  key = '2';
  break;
  case 0x40:                    // 1st row, 4th column
  key = '1';
  break;
  }
 }
 if (row==0x02)
 {
  switch(column)
  {
  case 0x10:                    // 2nd row, 1st column
  key = 'B';
  break;
  case 0x20:                    // 2nd row, 2nd column
  key = '6';
  break;
  case 0x30:                    // 2nd row, 3rd column
  key = '5';
  break;
  case 0x40:                    // 2nd row, 4th column
  key = '4';
  break;
  }
 }
 if (row==0x03)
 {
  switch(column)
  {
  case 0x10:                    // 3rd row, 1st column
  key = 'C';
  break;
  case 0x20:                    // 3rd row, 2nd column
```

```
  key = '9';
  break;
  case 0x30:                    // 3rd row, 3rd column
  key = '8';
  break;
  case 0x40:                    // 3rd row, 4th column
  key = '7';
  break;
  }
 }
 if (row==0x04)
 {
  switch(column)
  {
  case 0x10:                    // 4th row, 1st column
  key = 'D';
  break;
  case 0x20:                    // 4th row, 2nd column
  key = '#';
  break;
  case 0x30:                    // 4th row, 3rd column
  key = '0';
  break;
  case 0x40:                    // 4th row, 4th column
  key = '*';
  break;
  }
 }
 _H12PORTJ = 0x0F;                  // Reset Port J
 _H12KWIFJ = _H12KWIFJ;                 // Clear the flag
}


void keypad_init(void)             // Initialization Function
{                              // Assign the Vector Address
 DB12->SetUserVector(PortJKey, KeypadJ);
 _H12KPOLJ  = 0x00;             // Falling Edge sets Flag
 _H12KWIFJ  = 0xFF;             // Clear Any flags that may be set
 _H12PUPSJ  = 0xFF;             // Pull up
 _H12PULEJ  = 0x0F;             // Pull up enabled all bits
 _H12KWIEJ  = 0x0F;             // Enable all bits of J for keypad
 _H12DDRJ         = 0xF0;              // Data direction Out(MSB)/In(LSB)
 _H12PORTJ  = 0x0F;             // Initialize Port J
}

void getrow(void)                 // This function Determines the ROW
{                              // of the key being pressed
 switch (temp)
 {
 case 0x07: row =0x01;          // Set Row to 1
  break;
 case 0x0B: row =0x02;          // Set Row to 2
  break;
 case 0x0D: row =0x03;          // Set Row to 3
  break;
 case 0x0E: row =0x04;          // Set Row to 4
  break;
 default: row = 0;            // Set Row to 0
  break;
 }
}

void getcolumn(void)              // This function Determines the COLUMN
{                              // of the key being pressed

 _H12PORTJ = 0x10;                     // Send info to Port J testing for Column 1
 temp = _H12PORTJ;                           // Obtain Port J response
 if (temp == 0x1F)
 {
  column = 0x10;               // Column is 1 if lower bits goes low
```

```
  retum;
 }

 _H12PORTJ = 0x20;          // Test for Column 2
 temp = _H12PORTJ;          // Extract upper bits
 if (temp == 0x2F)
 {
  column = 0x20;            // Column is 2 if lower bits goes low
  retum;
 }

 _H12PORTJ = 0x4F;          // Test for Column 3
 temp = _H12PORTJ;          // Extract upper bits
 if (temp == 0x4F)
 {
  column = 0x30;            // Column is 3 if lower bits goes low
  return;
 }

 _H12PORTJ = 0x80;          // Test for Column 4
 temp = _H12PORTJ;          // Extract upper bits
 if (temp == 0x8F)
 {
  column = 0x40;            // Column is 4 if lower bits goes low
  retum;
 }
 column = 0;
}

#endif
```

main.c

```
//
// Household Security System
// Micro-Processor Systems Final Project
//
// File:  main.c
// Author(s):  Ming Deng, Tai Mong, Reynold Tam
// Revision:
//
// TM       12/7/99 -  This version contains the timeout function but
//                does not have the pin number function
//
//     11/22/99 - added preprocessor statements to ease testing..
//                can disable the cardReader part by commenting out
//                #define CARD_ENABLED              -MD
//     11/17/99 - Change functions A-D
//                          - Blinking LED indicates security system active
//                          - Buzzer sounds when continuity broken
//          11/10/99 - Accessing PORT S for I/O
//          11/7/99  - Included code to work with Decoder stuff
//
////////////////////////////////////////////////////////////////////
// Comment out the follow if card reader is not used for
// current testing..
#define CARD_ENABLED

#ifndef KEYPAD_H
#include "keypad.h" /* Keypad interface functions */
#endif

#ifndef DISPLAY_H
#include "display.h" /* include program specific LCD functions */
#endif

#ifndef CARDREADER_H
#include "cardReader.h"      /* include the Card Reader stuff */
#endif
```

```
int counter;                    /* Counter to regulate blinking light */

/////////////////////////////
// Function prototypes
/////////////////////////////
__mod2__ void RTIInt(void); // Real Time Interrupt Service Routine

void menu(void);     // display menu on LCD to user

void Func_A(void);  // functions to be called when the user
void Func_B(void);  // press the corresponding key on the keypad
void Func_C(void);  // key: A, B, C, D..
void Func_D(void);

void delay(void);  // delay system.. halts it for a short time
void blink_light(); // set a LED to blink
void buzz();        // turn on the buzzer

void determine_function(void);/* Cases of different functions */

#ifndef KEYPAD_H
  int TimeCount;                // Counter to regulate timeout function
#endif


/////////////////////////////
// Main
/////////////////////////////
void __main()
{
  char temp;
  int i, j;
  _H12DDRS = 0x7F;            /* Set data direction */
  _H12PORTS = 0x00;           /* Initially turn off all lights */

  key = 'x';          /* reset the key, 'x' is not a valid key on keypad
  TimeCount = 0;

  _H12TSCR = 0x80;            /* Init the free running counter */

  display_init();       /* Initialize the LCD */
  keypad_init();        /* Initialize Keypad */

#ifdef CARD_ENABLED
  cardReader_init(); /* Initialize the Card Reader */
  cardReader_reset();
  display_clear();
  display_send1("Swipe card");
#endif

  DB12->SetUserVector(RTI, RTIInt); // set the RTI to be used for timeout

DB12->printf("\n\r\n\r\n\rSystem running");

  // the following is an infinite loop, its the guts of the system
  while(1)
  {
#ifndef CARD_ENABLED
    menu();                                     /* display menu */
#endif

#ifdef CARD_ENABLED
    if (CDarmed == 0)
    {
      determine_function(); /* determine the function to execute */
    }
#endif

#ifndef CARD_ENABLED
```

```
    determine_function(); /* determine the function to execute */
#endif
   // if system is armed, show status by blinking LED
   if (CDarmed==1)
   {
    blink_light();
    temp = _H12PORTS & 0x80;
    //  DB12->printf("\n\r %c, %c",temp,temp);
    // if break in detector circuit is broken, sound alarm.
    if (temp == 0x00)
    {
     buzz();
    }
   }
#ifdef CARD_ENABLED

  // delay system
  for (i=0; i<10; i++)
  {
   for (j=0; j<2000; j++)
   {
   }
  }
#endif
  }
}


// Real Time Interrupt Service Routine
__mod2__ void RTIInt(void)
{

  TimeCount++;
  _H12RTIFLG = 0x80;                    // Clear Flag

  // (153 x 65.536 ms) ~ 10 secconds
  if ((CDarmed==0)&&(TimeCount > 153))
  {
   CDarmed = 1;     // Disable keypad controls / Arm system
   TimeCount = 0;     // Resets counter
   _H12RTICTL = 0x00;        // Disable RTI

   display_clear();
   display_send1("System Timeout  ");
   _H12RTICTL = 0x00;        // Disable RTI
   delay();

   display_send1("System timeout  ");
   delay();

   display_send1("System armed    ");
   display_send2(" Swipe card     ");
  }
}



void delay(void)
{
  int i, j;
  for (i=0; i<10; i++)
  {
   for (j=0; j<30000; j++)
   {
   }
  }
}


void menu(void)
```

```
{
 int i;
 _H12PORTS = (_H12PORTS & 0xFB); // Buzz OFF
 _H12PORTS = _H12PORTS & 0xBF;   // Light OFF

 display_clear();

 display_send1("    Options     ");
 delay();
 display_send1("A. Unlock Hood  ");
 display_send2("B. Unlock Door  ");
 delay();

 display_send1("C. Unlock Trunk ");
 display_send2("D. Arm Alarm    ");
 delay();

 display_send1("Choose A,B,C,D  ");
 display_send2("Other key->MENU ");
 _H12RTICTL = 0x87; // enable time out
}


//////////////////////////
// Menu functions
//////////////////////////


// current this is used to unlock the hood
void Func_A(void)
{
 //  char temp;
 _H12DDRS=0x7F;/* last port is input all else is output */

 // temp = _H12PORTS & 0x08;
 display_clear();
 display_send1("Hood Unlocked");

 _H12PORTS = 0x08;             /* S3 on for hood */
 delay();

 display_clear();
 display_send1("Choose A,B,C,D  ");
 display_send2("Other key->MENU ");

 _H12PORTS = 0x00;
 delay();

 return;
}

// used to lock or unlock the doors
void Func_B(void)
{
 //  char temp;
 _H12DDRS=0x7F;
 //  temp = _H12PORTS;
 display_clear();
 display_send1("Doors Unlocked");
 _H12PORTS = 0x10;                     /* S4 for doors */
 delay();
 delay();
 delay();

 display_clear();
 display_send1("Doors Locked");

 _H12PORTS = 0x00;
 delay();
```

```
  display_clear();
  display_send1("Choose A,B,C,D  ");
  display_send2("Other key->MENU ");

  return;
}

// used to unlock the trunk
void Func_C(void)
{

  //  char temp;
  _H12DDRS = 0x7F;  /* last port is input all else is output */
  //  temp = _H12PORTS;
  display_clear();
  display_send1("Trunk Unlocked");

  _H12PORTS = 0x20;              /* S5 on for trunk */
  delay();


  _H12PORTS = 0x00;
  delay();

  display_clear();
  display_send1("Choose A,B,C,D  ");
  display_send2("Other key->MENU ");

  return;
}

// used to arm the security system
void Func_D(void)
{
  //  char temp;
  _H12DDRS=0x7F;
  //  temp = _H12PORTS;
  _H12PORTS = 0x00;
  CDarmed = 1;
  display_send1(" System Armed    ");
  display_send2("  Swipe card     ");
  delay();

  _H12RTICTL = 0x00;                          // Disable RTI

}

/////////////////////////
// Other functions
/////////////////////////

void blink_light()
{
  char temp = _H12PORTS & 0x40;

  counter++;
  if ((counter % 10)==0)   /* Execute every 100th cycle */
  {
    if (temp == 0x40)
    {          /* if on, turn off */
      _H12PORTS = _H12PORTS & 0xBF; /* Sets the s6 to 0 */
    }
    else
    {
      _H12PORTS = _H12PORTS | 0x40; /* if off, turn on */
    }
  }
}

void buzz()
```

```
{
 // char temp = _H12PORTS & 0x04;
 _H12PORTS = (_H12PORTS | 0x04);  /* Set buzzer (S2) on */

 // DB12->printf("\n\r Inside Buzz()");
 // delay();
 // _H12PORTS = (_H12PORTS & 0xFA);
}

void determine_function(void)
{
 switch(key)              // Key from Keypad used instead
 {
 case 'a':
 case 'A':
  Func_A();
  break;
 case 'b':
 case 'B':
  Func_B();
  break;
 case 'c':
 case 'C':
  Func_C();
  break;
 case 'd':
 case 'D':
  Func_D();
  break;
 default:
  if (key!='y' && CDarmed == 0)
  {                              // Redisplay menu for any other keys pressed
   menu();                       // Display Menu
   key = 'x';            // Reset key to 'x', menu already displayed
  }
  break;
 }
 key='y';               // using key as a flag, menu not yet displayed
}
```

To develop the entire C code, we split the duties into two parts: the card reader and the *main.c*. The main program includes the LCD and the keypad but we have prior experience with it. Since we never worked with the card reader before, we would test the main program without it and assumed the user has passed the card swiping. The same is for the card reader when we tried to understand how the bit stream was entered into the 68HC12 and figure out how the algorithm should be set up. This method also helps us isolate problems that are related to either the card reader or the main program itself.

References

Course notes:

- Interfacing a Hitachi HD44780 to a Motorola 68HC11 or Motorola 68HC12
- Motorola 68HC12 User's Manual – Lee Rosenberg ECSE RPI, revision 1.0 4/10/99

Bibliography

- MPS in class lab materials
- Motorola website (www.mot.com) for the 68HC12 data sheets