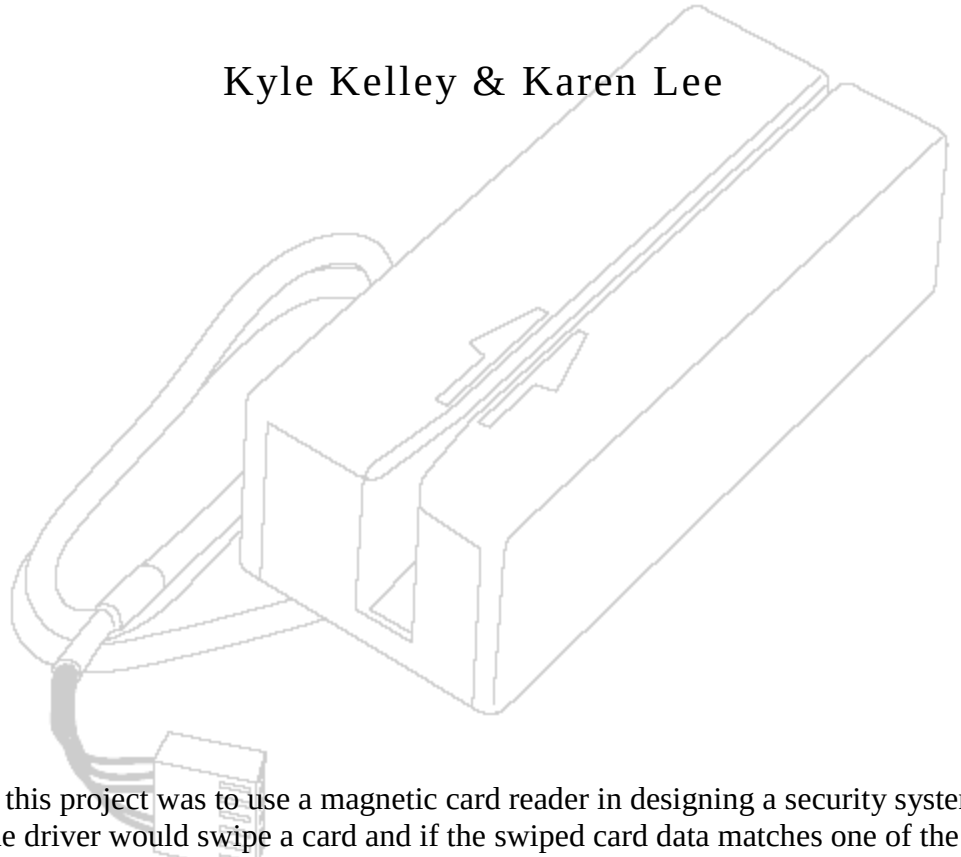
MAGNETIC CARD IDENTIFICATION SECURITY SYSTEM

Final Project Report
December 15, 2003
E155

Kyle Kelley & Karen Lee



Abstract:

The goal of this project was to use a magnetic card reader in designing a security system for a vehicle. The driver would swipe a card and if the swiped card data matches one of the four sets of data previously stored, he/she is allowed to start the car. The FPGA handles user control settings inputted via three sets of four pin dip switches that determine the mode and user of the system. It decodes the mode and user and outputs accordingly to the PIC. The card reader is connected directly to the PIC so that when a card is swiped, the data off the card is transmitted serially to the PIC. Depending on whether the system is in Program or Compare mode, the PIC either stores the swiped card data in EEPROM or compares it to previously stored data. For the purposes of this project we did not implement the system in an actual vehicle but used LEDs instead to demonstrate functionality. The PIC outputs to the correct LEDs indicating either a match or mismatch in data.

INTRODUCTION

Magnetic cards have become a part of all of our lives. The average person's wallet likely has several including a driver's license, ATM cards, and credit cards. Magnetic cards are used to identify us in banks, grocery stores, and here at Platt. Each card is unique in the data that is stored on the magnetic strip. Our project takes advantage of this fact in designing an automobile anti-theft security system. This project utilizes a magnetic card reader in designing a security system that can be implemented to prevent the unauthorized use of a vehicle. The main idea is that the driver must swipe a specific card before being allowed to start the car. One way to get to the data on the card is to swipe the card through a magnetic card reader. The one we used can be seen in Figure 2 in the next section.

The system has two states: Program and Compare. In Program mode, the user can store up to four different cards in memory. This allows up to four different users of the system which is especially useful in families where more than one person may drive the car. The user setting is inputted by way of a four pin dip switch. In Compare mode, the PIC will compare the data off of a swiped card to the data previously stored in memory. The PIC will in fact compare the data with all four possible sets of data in memory so that the user need not worry about the user setting when in Compare mode. The normal operating state of the system is the Compare mode and the system can only be put into Program mode when a specific eight bit code is entered by way of a pair of four pin dip switches.

If the system were to be fully implemented, we would output a signal which would drive a relay and untie the automobile's ignition wire from ground upon a matching card swipe. For the purposes of this project we will not actually implement the system into an automobile but rather utilize LEDs that signify whether or not the ignition is allowed to start. Further, while our proposed idea has the application of preventing unauthorized use of an automobile, this particular system could have a wide range of uses and be implemented to interact with various systems.

The three main components of our system are the FPGA, the PIC, and the card reader. Figure 1 on the next page depicts the block diagram of the system and how the components interact with each other.

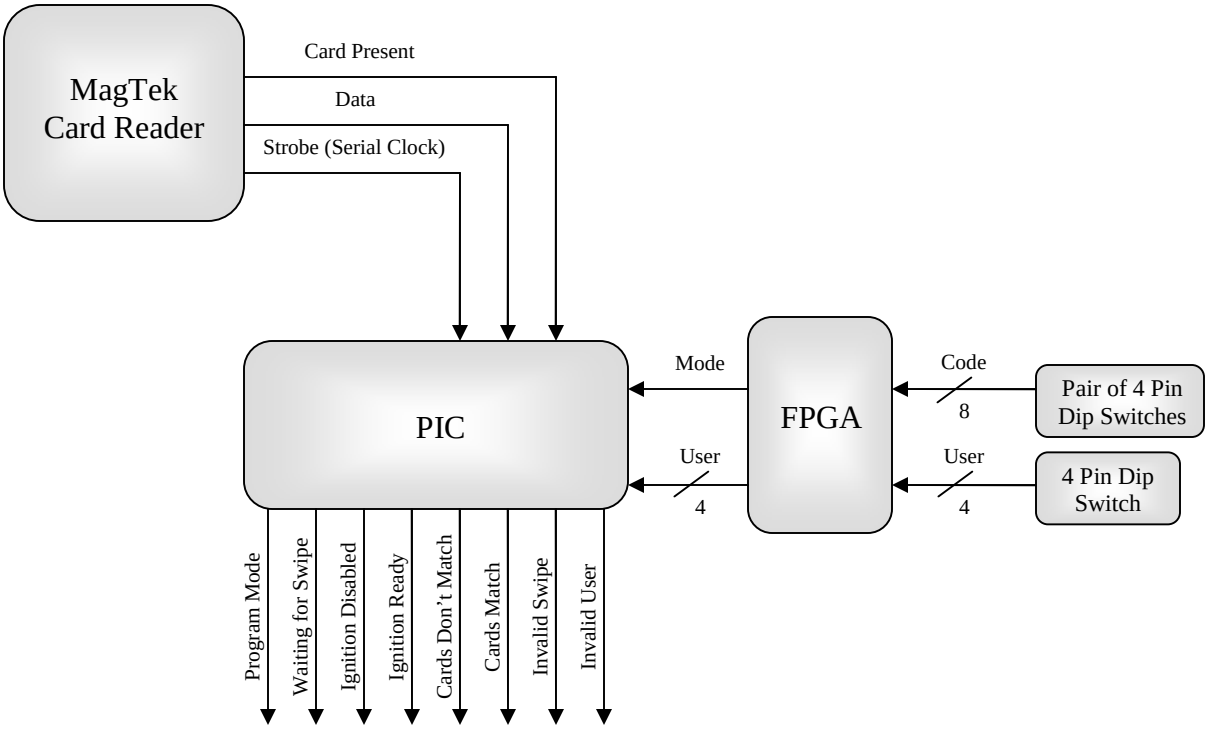


Figure 1: System Block Diagram

The card reader is connected directly to the pins of the PIC. The PIC is configured for slave serial transmission. The MagTek reader sends a signal whenever a card is being swiped (Card Present) and this signal is used to begin and stop storing data. The Strobe signal outputted from the card reader is used as a clocking signal by the PIC during transmission and indicates when the Data signal is valid and should be sampled. The Data signal carries the actual data encoded on the magnetic strip and is connected to the serial transmission input pin of the PIC.

The FPGA handles user control settings such as which mode the system is in as well as which user the system is currently set on. The inputs to the FPGA come from the pins connected to the three different four pin dip switches. It decodes which mode and user the system should be set to and then outputs accordingly to pins on the PIC. If an invalid user is inputted, the FPGA sends out all 0's.

The PIC either stores card data in EEPROM when in Program mode or compares swiped card data stored in data RAM to stored data in EEPROM when in Compare mode. If a valid card is swiped while in Compare mode, the PIC outputs to two green LEDs. One blinks and indicates a match has occurred while the other stays lit for fifteen seconds signifying the ignition is ready to

be started during the fifteen second window. If an invalid card is swiped in Compare mode, the PIC outputs to two red LEDs. One blinks and indicates a mismatch has occurred while the other stays lit and indicates the ignition is tied to ground and unable to be started. In addition, the PIC also outputs to two yellow LEDs, one indicating when the system is ready and waiting for a card to be swiped, the other indicating when the system is in Program mode. We also designed the system to distinguish from an invalid card from a badly swiped card. If the card is pulled out of the reader prematurely or swiped excessively slowly, the data can be read incorrectly even if it is a valid card. The PIC turns on a red "Invalid Swipe" LED indicating when an error in reading the data has occurred so that the user knows to swipe the same card again. Lastly, the PIC also outputs to a red "Invalid User" LED when an invalid user has been entered while the system is in Program mode.

NEW HARDWARE

MAGNETIC CARD READER

Our project made use of a 101mm card swipe reader from MagTek (Figure 2). It can read most cards with magnetic strips including bank cards, driver's licenses, and cards issued by super markets. The only cards we found that wouldn't work in the course of our testing were copy center cards and some calling cards.

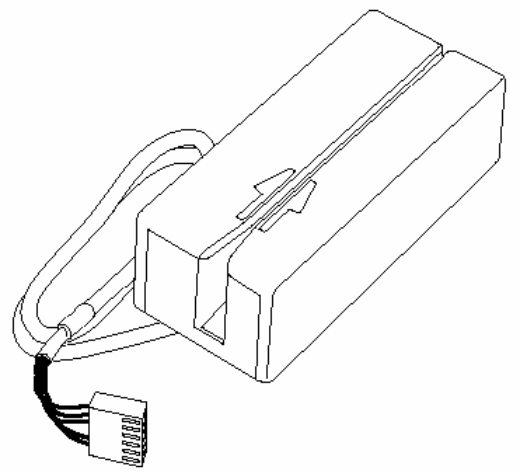


Figure 2: 101-millimeter Compatible Swipe Reader

Table 1: 8 Pin I/O Connector

Pin Number	Color	Signal
1	Yellow	$\overline{\text{DATA (Tk 2)}}$
2	White	$\overline{\text{CARD PRESENT}}$
3	Green	$\overline{\text{STROBE (Tk 2)}}$
4	–	KEY
5	Red	V_{cc}
6	Black	GND
7	Blue	$\overline{\text{STROBE (Tk 1 or Tk 3)}}$
8	Brown	$\overline{\text{DATA (Tk 1 or Tk 3)}}$

The reader has a dual track I/O connector. The connector has eight pins as shown in Table 1 above. We utilized the data off of track two since that's where the numeric information is mostly stored. We therefore made use of pins 1, 2, 3, 5, and 6. Pins 5 and 6 correspond to power and ground. Pins 1, 2, and 3 (i.e. Data, Strobe, and Card Present signals) are high when no card is being swiped. Card Present goes low when a card is being moved through the unit. This signal is used by the PIC to determine when the reader is sending data and when it is done. The Data signal carries the information stored on the card. When the Data signal is high, the bit is a zero and when the signal is low, the bit is a one. It is valid while the strobe is low and is connected to the input pin of the PIC for serial transmission. The Strobe signal indicates when Data is valid and is used by the PIC as the clock for serial transmission. The timing and interaction of these signals can be seen in Figure 3 below.

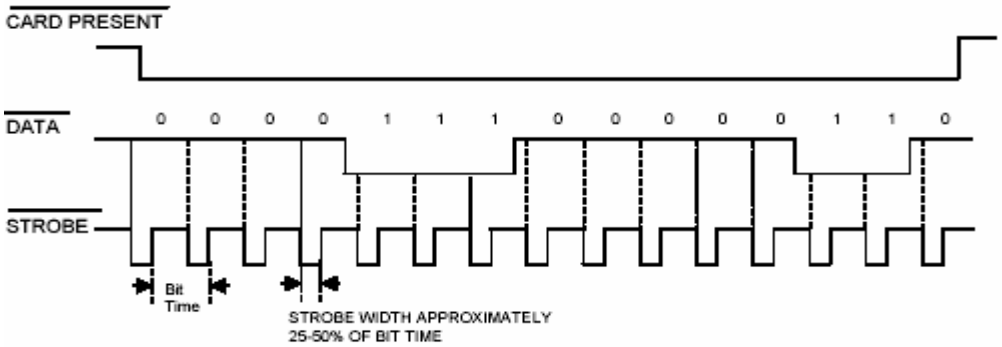


Figure 3: Signal Timing

SCHEMATIC

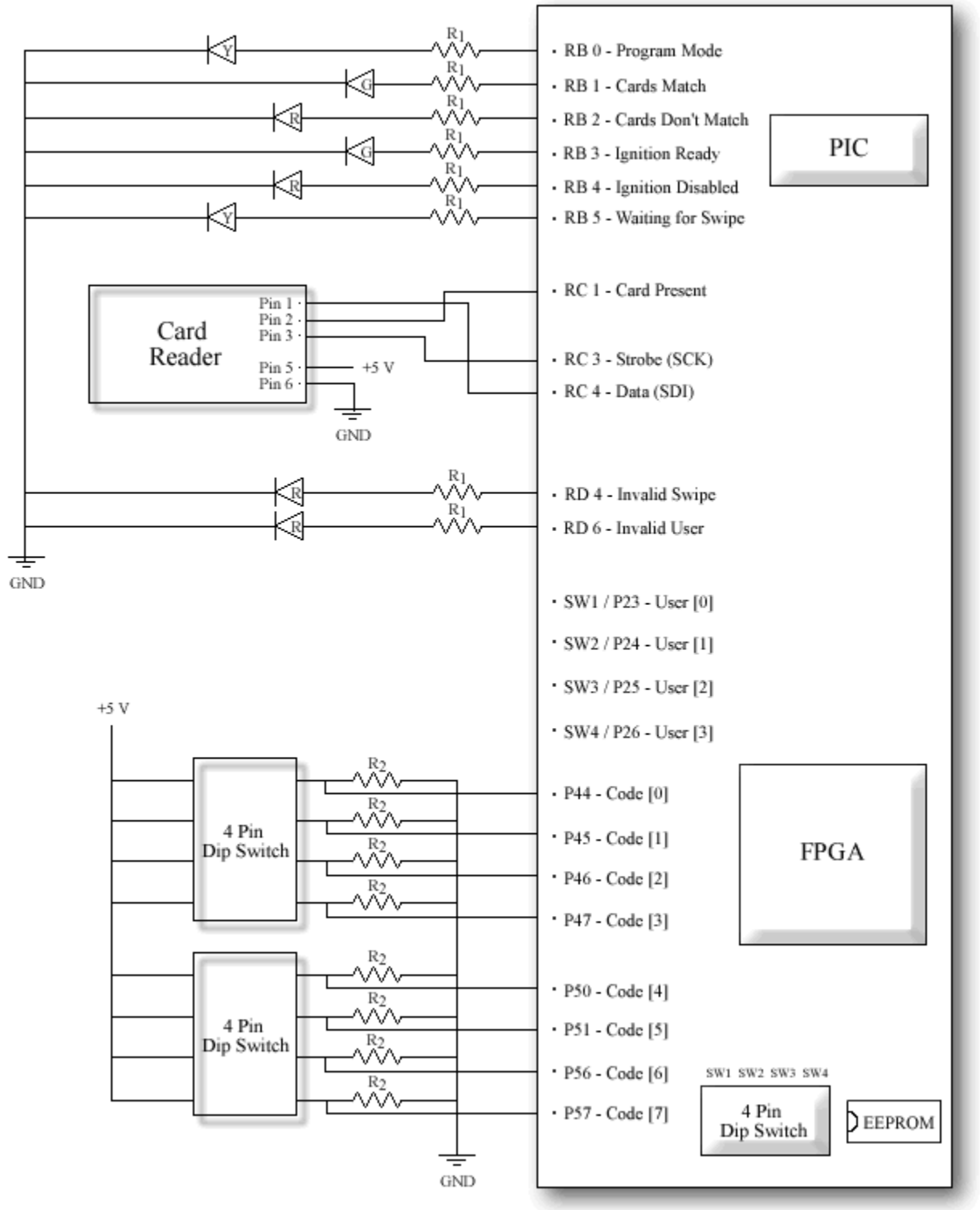


Figure 4: Breadboard Schematic - R1 = 330 Ω; R2 = 1 kΩ

MICROCONTROLLER DESIGN

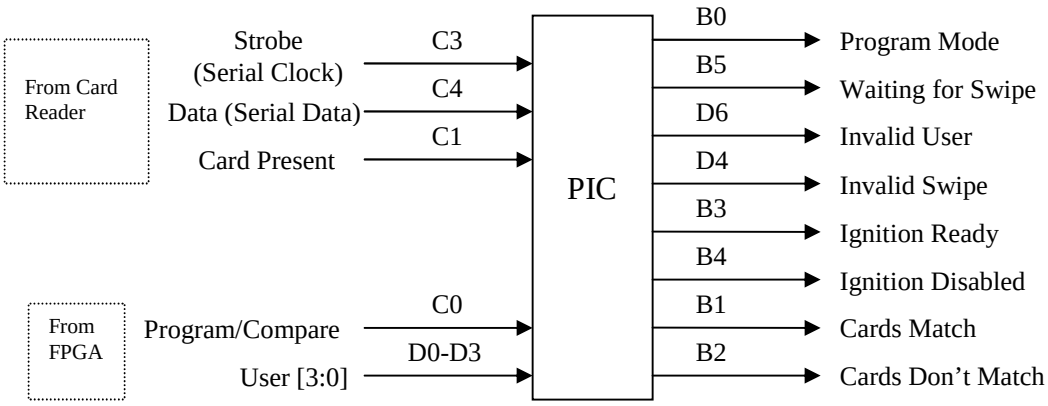


Figure 5: Inputs and Outputs of the Microcontroller

The inputs and outputs of the microcontroller can be seen in Figure 5 above, labeled with both the pin number and the signal name. The PIC receives the data from the card reader and stores it into data RAM in blocks of 8 bits. It then goes through and parses the data into 5 bit blocks so that it can then do a parity check on the data to see if any errors in transmitting occurred. (See Reference [3] for details on parity encoding) After the data is checked, the PIC either stores the data into EEPROM if it is in Program mode or compares it to data previously stored in EEPROM if it is in Compare mode. The PIC then outputs to the correct pins corresponding to either a match or mismatch in data. Below is a description of each of the main sections of the program.

waiting

After initialization, the microcontroller enters a "waiting" state, in which it asserts a high output on B5, indicating that it is ready and waiting for a card to be swiped. It also continually checks "Program / Compare" on C0 to determine the mode of the system. If it is in Program Mode, it will poll the inputs D0-D3 looking for a valid user. If none of D0-D3 are high, a high output is asserted on D6 indicating an invalid user and the program will not exit the waiting state until a valid user is found.

The Waiting Loop also polls the "Card Present" input on C1. Once it is determined a card is being swiped (indicated by "Card Present" falling from high to low), the microcontroller exits the waiting state and proceeds to initializeSend.

intializeSend

Here the output on B5 is driven low to indicate the system is no longer waiting for a card to be swiped. The PIC continually polls the SSPSTAT[0] bit (Buffer Full Status bit) as the card reader sends its data serially, and stores the data as an array into the data RAM as 8 bit bytes. Once "Card Present" returns to high, the sending of data is complete and the PIC exits this loop and proceeds to sendDone.

sendDone

Once the PIC has finished receiving serial data, it temporarily disables and then re-enables serial mode. The purpose of this is to eliminate any trailing bits in the serial receive buffer which would otherwise appear at the beginning of subsequent card swipes. The PIC then enters beginCheckParity.

beginCheckParity

This is the beginning of a large block of code with a very specific and simple function. First, the PIC parses the stored input beginning with the first bit looking for the ISO 7813 standard start sentinel 11010 (or in our case 00101, since our card reader inverts the data). Once the start sentinel is found, it is stored as a 5 bit block in a new data memory location (3 leading zeros make up the full byte), and the PIC advances to foundStart0. If no start sentinel is found and the end of the data is reached, the PIC branches to invalidRead.

foundStart0

The PIC continues parsing the continuous data into 5 bit blocks and stores them into data memory (placing 3 leading 0s on each block). Before it stores each 5 bit block it performs a parity check on it. The parity bit is the least significant bit of each string of 5 bits and makes the total number of 1s odd (again, since our card reader inverts the data, the total 1s in each byte should be even). If the parity of any single bit is found to be invalid, the PIC branches to invalidRead. Once the end sentinel 11111 (00000 for us) is found, the PIC branches to endOfData.

endOfData

Once the end sentinel is found, there is one more valid 5 bit block of data, the LRC parity bit. Although we did not implement the multiple-error parity checking that uses this bit, this bit is stored in data RAM anyway. Once this parity bit is stored, the PIC branches to allDataStored.

invalidRead

Output D4 (Invalid Swipe) is asserted for 2 seconds and the PIC branches back to waiting.

allDataStored

Here the PIC checks input C0 (“Program / Compare”). If it is low, it branches to compareOuter. Otherwise, it branches to program.

compareOuter

The function of this section is to compare the parsed, parity checked card data recently stored in data RAM to that programmed into EEPROM. Since our system allows multiple users to program their cards into EEPROM, the program must check the recently stored data against each

stored user. If a byte fails to match the corresponding byte of a stored user, the program immediately moves on to check the next user. If a byte fails to match and it is on the last user, it branches to noUserMatch. If the program reaches the end of a user's stored data (and therefore has not found any non-matching bytes), it branches to endCompare.

noUserMatch

When a card not matching one programmed into the system is swiped, this routine blinks the "Cards Don't Match" red led fives times. Each time, the led stays on for approximately .125 seconds. This is accomplished by toggling the value outputted to (B2) ten times and calling on a .125 seconds delay subroutine each time. The program then returns to waiting for the next card swipe.

endCompare

This routine is accessed when data from the swiped card matches data from one of the stored cards. Similar to the routine called when the card doesn't match, this routine blinks a green "Cards Match" led five times. This is accomplished in the same manner as above; the pin controlling the green led (B1) is toggled ten times and a delay lasting about .125 seconds is called after each time. In addition to the blinking led, this routine also turns off the red "Ignition Disabled" led (B4) and turns on the green "Ignition Ready" led (B3) for a duration of fifteen seconds, signifying that the driver has a fifteen second window in which he/she can start the car. The fifteen seconds duration is accomplished by calling another delay routine written to last approximately one second. Calling this routine repeatedly in a loop a specific number of times produces the longer delay we desire. At the end of the fifteen seconds, the green "Ignition Ready" led (B3) is turned off, the red "Ignition Disabled" led (B4) is turned back on, and the program branches back to waiting.

program

The purpose of this section is to store the parsed, parity checked data in data RAM into the appropriate section of the EEPROM. The start address within the EEPROM was previously determined in the waiting loop when it polled inputs D0-D3. Once all of the data has been stored into EEPROM, the program stores the end address for the appropriate user into another specified EEPROM location (this address is used when performing later compares, as the program must be able to "know" when to stop comparing).

SUBROUTINES

delay

This routine produces a delay lasting about one eighth of a second, or .125 seconds. It is used to blink the "Cards Match" (B1) and "Cards Don't Match" (B2) LEDs after a card has been swiped in Compare mode. It is accomplished by implementing loops that execute a sequence of instructions that don't affect any other factors of the program.

oneSecDelay

This produces a delays lasting for approximately one second. It is used for asserting D4 (Invalid Swipe) and B3 (Ignition Ready) for a duration determined by how many times we call the routine.

FPGA DESIGN

The FPGA is used to manage some of the control signals of the system as shown in Figure 1. Two sets of 4 pin dip switches are used for inputting a code to put the system in program mode and another set of 4 pin dip switches is used for inputting a user. The FPGA takes inputs from the pins connected to these dip switches and outputs the proper values allowing the system to have the correct settings. Three Verilog modules were created to accomplish this and are described below.

FPGA CONTROL SIGNALS MODULE

This high level module puts all the functions of the FPGA together by calling the other two Verilog modules created. Together, these modules handle outputting the proper values to indicate which mode and user the system is currently set to.

USER MODULE

The system can currently accommodate four different users. That is, four different cards can be programmed into the system and any one of them will produce a match when swiped in compare mode. A four pin dip switch is used to indicate which user the system is currently set on. Currently the users are encoded using the one hot method where 0001 indicates user 1 and 1000 indicates user 4. This module takes the input from the four pin dip switch and then sends the value to the pins that control PORTD [0:3] so that it can then be used by the PIC. On the occasion that the FPGA receives an invalid user input (i.e. one that doesn't follow the one hot encoding), the FPGA will output 0000.

PROGRAM MODE MODULE

In order for the system to be in program mode, an eight bit code must be entered via two 4 pin dip switches. The code is currently set to 10101010. This Verilog module takes the inputs from the pins connected to these dip switches and then outputs a logical high when the input matches the code and a logical low when it doesn't. This output is sent to a pin (C0) the PIC can use to determine when the system should be in program mode and when it should be in compare mode.

RESULTS

In the initial proposal we said that the programmed card data was going to be stored in the PIC's instruction memory but in the final design, it is stored in the PIC's EEPROM. We looked into the EEPROM at the suggestion of Professor Harris and decided the EEPROM was easier to work with, gave us what we wanted (i.e. the stored data would not be lost upon powering down), and we wouldn't have to worry about the data interfering with the program.

In the initial proposal we had also planned on using the FPGA to interface between the card reader and the PIC. That is, the FPGA would first store the data coming off of the reader and then send it to the PIC serially. The data was going to be stored in 7 bit blocks. We eliminated this intermediary step in the final design so that the card reader sends the data directly to the PIC where it is initially stored in 8 bit blocks but then parsed to 5 bit blocks to facilitate error checking. The 5 bit block size was needed because the data and the parity bits are encoded as a function of 5 bits on the magnetic strip.

Deciding whether or not we needed the FPGA to act as a middleman in the data transmission was one of the most difficult aspects of this project. The FPGA was ultimately eliminated as an intermediate step in data transmission because it proved to be an unnecessary step that would lead to substantially more Verilog code that needed to be written and debugged. We originally planned on using the FPGA to resolve timing issues with data transmission. We thought it might be necessary to slow down the transmission of the data in order for it to be received and stored correctly by the PIC. However, testing the card reader with the PIC directly proved otherwise, once the settings for slave mode were correct. Since the initial Verilog code written to support the use of the FPGA in this manner was riddled with bugs that had already consumed many hours in simulation and debugging, we decided to simply take it out. Therefore, since the card reader could be interfaced with the PIC directly and reliably, the FPGA was eliminated to avoid lots of unnecessary code as well as the opportunity for more errors to arise.

Another difficult aspect of our design process was getting the PIC properly set up for slave serial transmission. Because we had only previously used the PIC in master mode when doing serial transmission, we had to learn on our own how to use the PIC in slave serial mode. Part of the reason we had trouble getting it to work was because the PIC we were using was slightly faulty. We discovered this by using two PICs, one configured in master mode sending data to the other configured in slave mode. We watched the different signals on the digital oscilloscope and noticed that the serial clock signal would get irregular when using one of the PICs in slave mode but would look uniform and as we would expect when using the other PIC. Replacing the faulty PIC allowed us to configure the PIC properly so that we could send and receive test data correctly and verify that the slave serial mode on the PIC works as expected.

Another problem we encountered was a bug in the debugging program of MPLAB. The window used to watch the EEPROM registers while running and debugging the program does not update and show the current contents of the EEPROM. It simply shows FF as the contents of every EEPROM register. This at first led us to believe that we weren't writing to the EEPROM correctly but upon trying to read from the EEPROM we would get the correct value we stored, indicating our writing algorithm was indeed correct. Therefore, in one phase of our design and

testing, we had to resort to writing a loop that would rewrite the contents of the EEPROM registers to data RAM so that we could see its contents.

The system is ready to stand alone independent of the computer. The Verilog code was burned onto the external EEPROM and the PIC assembly code was downloaded onto the PIC so that as soon as the board is hooked up to a power supply, it is ready to go. Since the programmed data is stored in the PIC's EEPROM, it is still there even after powering down and back up so that the system doesn't need to be reprogrammed if the user doesn't wish to do so.

In the end, we are very pleased with the final outcome of our project. It in fact works better than we had anticipated. In addition to the initial system we had proposed, we incorporated a few extra features. We implemented multiple users so that the system can currently support up to four different cards. We also did some parity checking in the data which allowed us to cut down on the amount of data stored since we could eliminate the leading and trailing bits before and after the begin and end sequences in the data; this greatly facilitated comparing. In doing the parsing and parity checking we were constraining ourselves to cards that conform to the ANSI/ISO Track 2 BCD standard [3], but since almost all cards we checked conformed to this, with the exception of a copy card and a calling card, this was not a problem. The parity checking allowed us to be able to distinguish between an invalid card and a badly swiped card. When a card is simply swiped poorly so that there is an error in the data transmitted, a red led is lit indicating that it was a bad swipe so that the user knows to simply swipe again. This means that when a valid card is swiped, only the "Cards Match" or "Invalid Swipe" LEDs will ever be lit and a "Cards Don't Match" shouldn't ever be produced.

REFERENCES

[1] MagTek Magnetic Card Reader Products,
http://www.magtek.com/prod_guide/cards/SwipeInsert/SwipeInsert.html

[2] MagTek Card Reader Technical Manual,
http://www.magtek.com/documentation/public/99821101-6.pdf

[3] Magnetic Card Information: ANSI/ISO BCD Data Encoding,
http://www.hhhh.org/~joeboy/EE/hardware/magcards/trackdata_ANSI-ISO_BCD.html

PARTS LIST

Part	Source	Vendor Part #	Price
101-MILLIMETER COMPATIBLE SWIPE READER	MagTek	21050004	< $10 Used *

* We obtained ours from a fellow student but similar ones can be purchased used for less than $10 from www.allelectronics.com

Appendix A: FPGA Control Signals Module

```
module fpga_control(code_switch, user_switch, mode, user);

    input [7:0] code_switch;
    input [3:0] user_switch;

    output mode;
    output [3:0] user;

    code code(code_switch, mode);
    user user(user_switch, user);

endmodule
```

Appendix B: User Module

```
module user(s,user);
    input [3:0] s;
    output [3:0] user;

    reg [3:0] user;

    always @ (s)
        case (s)
            4'b0001:    user <= s;
            4'b0010: user <= s;
            4'b0100: user <= s;
            4'b1000: user <= s;
            default: user <= 0;
        endcase

endmodule
```

Appendix C: Program Mode Module

```
module code(s, q);
    input [7:0] s;
    output q;

    assign q = s[7] & ~s[6] & s[5] & ~s[4] & s[3] & ~s[2] & s[1] & ~s[0];

endmodule
```

Appendix D: PIC Assembly Code

```
; CardReader5.asm
; Updated December 7, 2003 by kkelley@hmc.edu and kmlee@hmc.edu
; Most code is based on smaller test code
; Places serial input into data EEPROM when in program mode.
; Places serial input into data RAM when in compare mode.
; Allows Multiple Users To Store Card Data in EEPROM
; Compares data RAM values with those stored in data EEPROM memory.
; Uses Parity Bit Error Checking to Distinguish Between Bad Swipe and Wrong
; Card

; Use the 18F452 PIC microprocessor
       LIST p=18F452
       include "p18f452.inc"


; define variables

DDATA       EQU 0x00    ; start of data in data memory
DDATAEND    EQU 0x01
DDATA_TRUE  EQU 0x0D
DDATAEND_TRUE EQU 0x0E
EDATA       EQU 0x02    ; start of data in program memory
EDATAEND    EQU 0x03

EDATAEND0   EQU 0xC0
EDATAEND1   EQU 0xC1
EDATAEND2   EQU 0xC2
EDATAEND3   EQU 0xC3

USER        EQU 0x13
USERTEMP    EQU 0x14

matchCounter EQU 0xD0
counter1    EQU 0xD1
counter2    EQU 0xD2
blinkCount  EQU 0xD3

; define constants
EDATA0      EQU 0x00
EDATA1      EQU 0x30
EDATA2      EQU 0x60
EDATA3      EQU 0x90

blinkDur    EQU 0x0A

; subroutine variables

INDEX1      EQU 0x04
INDEX2      EQU 0x05
INDEX3      EQU 0x06
DUR         EQU 0x07
DURIND      EQU 0x08
PARITY      EQU 0x09
BITPOINTER  EQU 0x0A
```

```
CHARCOUNTER EQU 0x0B
CURRENTCHAR EQU 0x0C

; subroutine constants

INSA        EQU 0xFA    ; 250 decimal
INSB        EQU 0xC7    ; 199 decimal

; begin main program
        org 0

main:

        clrf      TRISA       ; set PORTA as output
        clrf      PORTA

        movlw     0x20
        movwf     DDATA       ; set beginning address in data mem

        movlw     0x50
        movwf     DDATA_TRUE  ; set beginning address of actual encoded bytes

        movlw     0x00
        movwf     EDATA       ; set beginning address in EEPROM

        clrf      TRISB       ; set all A ports as Output
        clrf      PORTB       ; initially turn OFF all LEDs

        bsf       PORTB, 4    ; indicate ignition is disabled

        movlw     0x1B
        movwf     TRISC       ; enable SCK(4) and SDI(3) and CardPresent(1)
                              ; and Program/Compare(0)

        bcf       TRISD, 4    ; enable bad swipe output LED
        clrf      PORTD

        bcf       TRISD, 6    ; enable bad swipe output LED
        clrf      PORTD

        movlw     0x35
        movwf     SSPCON1     ; enable serial ports, slave mode, SS pin
                              ; control disabled, clk idle high

        movlw     0x00
        movwf     SSPSTAT     ; clear SMP for slave mode, transmit on rising
                              ; edge to ensure data quality

waiting:

        btfsc     PORTC, 0    ; check Program/Compare mode
        bra       userCheck
        bra       cont

userCheck:

        call      user        ; determine which user
```

```
        btfsc     PORTD, 6    ; check if error
        bra       userCheck

cont:   btfsc     PORTC, 0          ; check Program/Compare mode
                                    ; => indicate with LEDs
        bra       programModeOn
        bcf       PORTB, 0          ; turn off yellow LED if in compare mode
        bra       stillWaiting


; determine which user and assign appropriate start locations in memory

user:
        btfsc     PORTD, 0
        bra       user0
        btfsc     PORTD, 1
        bra       user1
        btfsc     PORTD, 2
        bra       user2
        btfsc     PORTD, 3
        bra       user3
        bra       invalid

user0:
        movlw     0x00
        movwf     USER
        movlw     EDATA0
        bra       storeMemStart

user1:
        movlw     0x01
        movwf     USER
        movlw     EDATA1
        bra       storeMemStart

user2:
        movlw     0x02
        movwf     USER
        movlw     EDATA2
        bra       storeMemStart

user3:
        movlw     0x03
        movwf     USER
        movlw     EDATA3

storeMemStart:

        movwf     EDATA
        bcf       PORTD, 6    ; turn off error led
        return

invalid:

        bsf       PORTD, 6          ; turn on error led
        return
```

```
programModeOn:

        bsf         PORTB, 0    ; turn on yellow LED if in program mode

stillWaiting:

        bsf         PORTB, 5    ; show that system is waiting for input
        btfsc       PORTC, 1    ; poll CardPresent, wait until a card is being
swiped
        bra         waiting
        bra         initializeSend


; STORE card swipes in data RAM

initializeSend:

        bcf         PORTB, 5    ; system is no longer waiting for input
        movf        DDATA, 0
        movwf       FSR0L

store:

        btfsc       PORTC, 1    ; check CardPresent
        bra         sendDone

        btfss       SSPSTAT, 0  ; poll BF bit of SSPSTAT
        bra         store
        movf        SSPBUF, 0   ; load the received data into WREG
        movwf       INDF0       ; store received data in data memory
        incf        FSR0L       ; increment the indirect pointer
        bra         store

sendDone:

        movlw       0x01        ; store 1 in WREG
        subwf       FSR0L, 0    ; subtract 1 from stored data RAM address
        movwf       DDATAEND    ; store end of RAM data address

        bcf         SSPCON1, 5  ; temporarily disable Serial Mode
        movlw       0x15
        movwf       SSPCON1
        bsf         SSPCON1, 5  ; re-enable Serial Mode

beginCheckParity:

        movff       DDATA, FSR0L      ; start at beginning of Data
        movff       DDATA_TRUE, FSR1L


        clrf        PARITY            ; clear parity register
        movlw       0x07
        movwf       BITPOINTER        ; initialize bit pointer register to 7
        clrf        CHARCOUNTER       ; clear character counter register
        clrf        CURRENTCHAR
```

```
findStart0:

continueParity0:

        ; check for full byte character count (5)

        movlw     0x05
        cpfseq    CHARCOUNTER
        bra       checkBit70
        bra       checkforStart0

checkforStart0:

        ; check for start of data sentinel

        movlw     0x05              ; place start sentinel into WREG
        cpfseq    CURRENTCHAR
        bra       notFoundYet0
        bra       foundStart0

notFoundYet0:
        decf      CHARCOUNTER       ; decrease CHARCOUNTER by 1 so it will
                                    ; be 5 next time it is increased

        ; check for end of data

        movf      DDATAEND, 0       ; move end of data address into WREG
        cpfseq    FSR0L
        bra       checkBit70        ; still bits left to be checked
        bra       lastPICByte0

lastPICByte0:
        movlw     0x04
        cpfslt    BITPOINTER
        bra       checkBit70
        bra       invalidRead       ; no start sentinel found!!!

checkBit70:
        movlw     0x07
        cpfseq    BITPOINTER
        bra       checkBit60
        bra       examineBit70

checkBit60:
        movlw     0x06
        cpfseq    BITPOINTER
        bra       checkBit50
        bra       examineBit60

checkBit50:
        movlw     0x05
        cpfseq    BITPOINTER
        bra       checkBit40
        bra       examineBit50

checkBit40:
```

```
        movlw     0x04
        cpfseq    BITPOINTER
        bra       checkBit30
        bra       examineBit40

checkBit30:
        movlw     0x03
        cpfseq    BITPOINTER
        bra       checkBit20
        bra       examineBit30

checkBit20:
        movlw     0x02
        cpfseq    BITPOINTER
        bra       checkBit10
        bra       examineBit20

checkBit10:
        movlw     0x01
        cpfseq    BITPOINTER
        bra       checkBit00
        bra       examineBit10

checkBit00:
        movlw     0x00
        cpfseq    BITPOINTER
        bra       checkBitEnd0
        bra       examineBit00

checkBitEnd0:
        incf      FSR0L
        movlw     0x07
        movwf     BITPOINTER        ; initialize bit pointer register to 7
        bra       checkBit70

examineBit00:
        btfss     INDF0, 0
        bra       addZero0
        bra       addOne0

examineBit10:
        btfss     INDF0, 1
        bra       addZero0
        bra       addOne0

examineBit20:
        btfss     INDF0, 2
        bra       addZero0
        bra       addOne0

examineBit30:
        btfss     INDF0, 3
        bra       addZero0
        bra       addOne0

examineBit40:
        btfss     INDF0, 4
```

```
        bra     addZero0
        bra     addOne0

examineBit50:
        btfss   INDF0, 5
        bra     addZero0
        bra     addOne0

examineBit60:
        btfss   INDF0, 6
        bra     addZero0
        bra     addOne0

examineBit70:
        btfss   INDF0, 7
        bra     addZero0
        bra     addOne0

addZero0:
        decf    BITPOINTER
        incf    CHARCOUNTER
        rlncf   CURRENTCHAR, 1
        bcf     CURRENTCHAR, 5
        bra     continueParity0

addOne0:
        decf    BITPOINTER
        incf    CHARCOUNTER
        rlncf   CURRENTCHAR, 1
        bsf     CURRENTCHAR, 0
        bcf     CURRENTCHAR, 5
        bra     continueParity0


foundStart0:
        movff   CURRENTCHAR, INDF1
        incf    FSR1L
        clrf    CHARCOUNTER
        clrf    CURRENTCHAR

continueParity:

        ; check for full byte character count (5)

        movlw   0x05
        cpfseq  CHARCOUNTER
        bra     checkBit7
        bra     checkParity

checkParity:

        ; check for end of data

        movlw   0x00            ; place end sentinel into WREG
        cpfseq  CURRENTCHAR
        bra     notAtEnd
        bra     endOfData
```

```
notAtEnd:
        clrf CHARCOUNTER
        btfss      PARITY, 0
        bra        evenParity
        bra        oddParity

checkBit7:
        movlw      0x07
        cpfseq     BITPOINTER
        bra        checkBit6
        bra        examineBit7

checkBit6:
        movlw      0x06
        cpfseq     BITPOINTER
        bra        checkBit5
        bra        examineBit6

checkBit5:
        movlw      0x05
        cpfseq     BITPOINTER
        bra        checkBit4
        bra        examineBit5

checkBit4:
        movlw      0x04
        cpfseq     BITPOINTER
        bra        checkBit3
        bra        examineBit4

checkBit3:
        movlw      0x03
        cpfseq     BITPOINTER
        bra        checkBit2
        bra        examineBit3

checkBit2:
        movlw      0x02
        cpfseq     BITPOINTER
        bra        checkBit1
        bra        examineBit2

checkBit1:
        movlw      0x01
        cpfseq     BITPOINTER
        bra        checkBit0
        bra        examineBit1

checkBit0:
        movlw      0x00
        cpfseq     BITPOINTER
        bra        checkBitEnd
        bra        examineBit0


checkBitEnd:
```

```
        incf    FSR0L
        movlw   0x07
        movwf   BITPOINTER      ; initialize bit pointer register to 7
        bra     checkBit7

examineBit0:
        btfss   INDF0, 0
        bra     addZero
        bra     addOne

examineBit1:
        btfss   INDF0, 1
        bra     addZero
        bra     addOne

examineBit2:
        btfss   INDF0, 2
        bra     addZero
        bra     addOne

examineBit3:
        btfss   INDF0, 3
        bra     addZero
        bra     addOne

examineBit4:
        btfss   INDF0, 4
        bra     addZero
        bra     addOne

examineBit5:
        btfss   INDF0, 5
        bra     addZero
        bra     addOne

examineBit6:
        btfss   INDF0, 6
        bra     addZero
        bra     addOne

examineBit7:
        btfss   INDF0, 7
        bra     addZero
        bra     addOne

addZero:
        decf    BITPOINTER
        incf    CHARCOUNTER
        rlncf   CURRENTCHAR, 1
        bra     continueParity

addOne:
        decf    BITPOINTER
        incf    CHARCOUNTER
        incf    PARITY
        rlncf   CURRENTCHAR, 1
        bsf     CURRENTCHAR, 0
```

```
        bra       continueParity

evenParity:
        ; VALID BYTE!
        ; continue checking parity

        clrf      PARITY
        movff     CURRENTCHAR, INDF1
        incf      FSR1L
        clrf      CURRENTCHAR

        bra       continueParity

oddParity:

        ; BYTE NOT VALID!
        ; indicate invalid read

        clrf      CURRENTCHAR

        bra       invalidRead

invalidRead:
        bsf       PORTD, 4
        call      oneSecDelay
        call      oneSecDelay
        bcf       PORTD, 4

        bra       waiting


endOfData:

        ; store END sentinel

        movff     CURRENTCHAR, INDF1
        incf      FSR1L
        clrf      CHARCOUNTER
        clrf      CURRENTCHAR
        ; store one more 5 bit byte (LRC Character)

continueParity1:

        ; check for full byte character count (5)

        movlw     0x05
        cpfseq    CHARCOUNTER
        bra       checkBit71
        bra       checkParity1

checkParity1:

        ; store final LRC Character

        movff     CURRENTCHAR, INDF1
        movff     FSR1L, DDATAEND_TRUE
```

```
        bra     allDataStored

checkBit71:
        movlw   0x07
        cpfseq  BITPOINTER
        bra     checkBit61
        bra     examineBit71

checkBit61:
        movlw   0x06
        cpfseq  BITPOINTER
        bra     checkBit51
        bra     examineBit61

checkBit51:
        movlw   0x05
        cpfseq  BITPOINTER
        bra     checkBit41
        bra     examineBit51

checkBit41:
        movlw   0x04
        cpfseq  BITPOINTER
        bra     checkBit31
        bra     examineBit41

checkBit31:
        movlw   0x03
        cpfseq  BITPOINTER
        bra     checkBit21
        bra     examineBit31

checkBit21:
        movlw   0x02
        cpfseq  BITPOINTER
        bra     checkBit11
        bra     examineBit21

checkBit11:
        movlw   0x01
        cpfseq  BITPOINTER
        bra     checkBit01
        bra     examineBit11

checkBit01:
        movlw   0x00
        cpfseq  BITPOINTER
        bra     checkBitEnd1
        bra     examineBit01


checkBitEnd1:
        incf FSR0L
        movlw   0x07
        movwf   BITPOINTER      ; initialize bit pointer register to 7
        bra     checkBit71
```

```
examineBit01:
        btfss      INDF0, 0
        bra        addZero1
        bra        addOne1

examineBit11:
        btfss      INDF0, 1
        bra        addZero1
        bra        addOne1

examineBit21:
        btfss      INDF0, 2
        bra        addZero1
        bra        addOne1

examineBit31:
        btfss      INDF0, 3
        bra        addZero1
        bra        addOne1

examineBit41:
        btfss      INDF0, 4
        bra        addZero1
        bra        addOne1

examineBit51:
        btfss      INDF0, 5
        bra        addZero1
        bra        addOne1

examineBit61:
        btfss      INDF0, 6
        bra        addZero1
        bra        addOne1

examineBit71:
        btfss      INDF0, 7
        bra        addZero1
        bra        addOne1

addZero1:
        decf       BITPOINTER
        incf       CHARCOUNTER
        rlncf      CURRENTCHAR, 1
        bra        continueParity1

addOne1:
        decf       BITPOINTER
        incf       CHARCOUNTER
        incf       PARITY
        rlncf      CURRENTCHAR, 1
        bsf        CURRENTCHAR, 0
        bra        continueParity1
```

```
allDataStored:


        btfss     PORTC, 0          ; check Program / Compare mode
        bra       compareOuter      ; Enter Compare mode
        bra       program           ; Enter Program mode

; COMPARE MOST RECENT SWIPE WITH DATA STORED IN EEPROM

compareOuter:

        clrf      USERTEMP

compare:

        movff     DDATA_TRUE, FSR0L

checkUser0:
        movlw     0x00
        cpfseq    USERTEMP
        bra       checkUser1

        movlw     EDATAEND0
        movwf     EEADR
        bcf       EECON1, EEPGD     ; point to DATA memory
        bcf       EECON1, CFGS      ; access program flash OR data EEPROM
        bsf       EECON1, RD        ; EEPROM read

        movff     EEDATA, EDATAEND

        movlw     EDATA0
        movwf     EEADR
        bra       keepComparing

checkUser1:
        movlw     0x01
        cpfseq    USERTEMP
        bra       checkUser2

        movlw     EDATAEND1
        movwf     EEADR
        bcf       EECON1, EEPGD     ; point to DATA memory
        bcf       EECON1, CFGS      ; access program flash OR data EEPROM
        bsf       EECON1, RD        ; EEPROM read

        movff     EEDATA, EDATAEND
        movlw     EDATA1
        movwf     EEADR
        bra       keepComparing

checkUser2:
        movlw     0x02
        cpfseq    USERTEMP
        bra       checkUser3

        movlw     EDATAEND2
```

```
        movwf   EEADR
        bcf     EECON1, EEPGD     ; point to DATA memory
        bcf     EECON1, CFGS      ; access program flash OR data EEPROM
        bsf     EECON1, RD        ; EEPROM read

        movff   EEDATA, EDATAEND
        movlw   EDATA2
        movwf   EEADR
        bra     keepComparing

checkUser3:
        movlw   0x03

        movlw   EDATAEND3
        movwf   EEADR
        bcf     EECON1, EEPGD     ; point to DATA memory
        bcf     EECON1, CFGS      ; access program flash OR data EEPROM
        bsf     EECON1, RD        ; EEPROM read

        movff   EEDATA, EDATAEND
        movlw   EDATA3
        movwf   EEADR

        bra     keepComparing

keepComparing:

        bcf     EECON1, EEPGD     ; point to DATA memory
        bcf     EECON1, CFGS      ; access program flash OR data EEPROM
        bsf     EECON1, RD        ; EEPROM read
        movf    EEDATA, 0         ; store EEPROM data into WREG

        cpfseq  INDF0             ; compare EEPROM data with current data
        bra     different
        incf    EEADR
        incf    FSR0L

        movf    EDATAEND, 0       ; checks if at end of data in EEPROM
                                  ; memory
        cpfsgt  EEADR
        bra     keepComparing

        bra     endCompare


different:

        ; CARDS DON'T MATCH
        ; check next user
        ; output proper signal
        ; quit comparing if all 4 users have been checked

        ; car doesn't start

        incf    USERTEMP
        movlw   0x03
        cpfsgt  USERTEMP
```

```
        bra     compare
        bra     noUserMatch


noUserMatch:

        clrf    blinkCount

RedBlink:
        movlw   0x01
        addwf   blinkCount
        movlw   blinkDur
        cpfseq  blinkCount
        bra     RedBlinkCont
        bra     red

RedBlinkCont:
        btg     PORTB, 2
        call    delay
        bra     RedBlink

red:
        bcf     PORTB, 2
        bra     waiting

endCompare:

        ; CARDS MATCH!
        ; output proper signal
        ; car does start

        bcf     PORTB, 4
        clrf    matchCounter

        bsf     PORTB, 3

        clrf    blinkCount

GreenBlink:
        movlw   0x01
        addwf   blinkCount
        movlw   blinkDur
        cpfseq  blinkCount
        bra     GreenBlinkCont
        bra     green

GreenBlinkCont:
        btg     PORTB, 1
        call    delay
        bra     GreenBlink

green:
        bcf     PORTB, 1
        call    oneSecDelay
        movlw   0x01
        addwf   matchCounter
        movlw   0x05
```

```
        cpfseq     matchCounter
        bra        green
        bcf        PORTB, 3
        bsf        PORTB, 4

        bra  waiting


; PROGRAM THE MASTER CARD IN EEPROM MEMORY

program:
        movff      EDATA, EEADR      ; Initialize EEPROM starting address
        movff      DDATA_TRUE, FSR0L ; Set indirect address pointer to
                                     ; beginning of DataRAM


writeByte:                           ; EEPROM WRITE SEQUENCE

        movff      INDF0, EEDATA     ; byte to be stored in EEPROM

        bcf        EECON1, 7   ; bit 7 = EEPGD (point to data EEPROM memory)
        bcf        EECON1, 6   ; bit 6 = CFGS (access data EEPROM memory)
        bsf        EECON1, 2   ; bit 2 = WREN (enable writes)

        bcf        INTCON, 7   ; bit 7 = GIE (disable interrupts)
        movlw      0x55
        movwf      EECON2            ; write 55h
        movlw      0xAA
        movwf      EECON2            ; write AAh
        bsf        EECON1, 1   ; bit 1 = WR (begin write, cleared when done)

loop1:
        btfsc      EECON1, 1   ; poll WR bit for writing to be done
        bra        loop1

        movf       FSR0L, 0
        cpfseq     DDATAEND_TRUE     ; check to see if last byte has been
                                     ; written to EEPROM
        bra        keepWriting       ; continue writing
        bra        writeDone         ; done writing to EEPROM

keepWriting:
        incf       EEADR       ; increment address
        incf       FSR0L
        bra        writeByte

writeDone:
        bcf        EECON1, WREN      ; full transfer to EEPROM done, disable
writes

setEnd0:
        movlw      0x00
        cpfseq     USER
        bra        setEnd1

        movff      EEADR, EEDATA     ; byte to be stored in EEPROM
        movlw      EDATAEND0
```

```
        movwf     EEADR

        bcf       EECON1, 7   ; bit 7 = EEPGD (point to data EEPROM memory)
        bcf       EECON1, 6   ; bit 6 = CFGS (access data EEPROM memory)
        bsf       EECON1, 2   ; bit 2 = WREN (enable writes)

        bcf       INTCON, 7   ; bit 7 = GIE (disable interrupts)
        movlw     0x55
        movwf     EECON2            ; write 55h
        movlw     0xAA
        movwf     EECON2            ; write AAh
        bsf       EECON1, 1   ; bit 1 = WR (begin write, cleared when done)

loop10:
        btfsc     EECON1, 1   ; poll WR bit for writing to be done
        bra       loop10

        bra       waiting

setEnd1:
        movlw     0x01
        cpfseq    USER
        bra       setEnd2

        movff     EEADR, EEDATA     ; byte to be stored in EEPROM
        movlw     EDATAEND1
        movwf     EEADR

        bcf       EECON1, 7   ; bit 7 = EEPGD (point to data EEPROM memory)
        bcf       EECON1, 6   ; bit 6 = CFGS (access data EEPROM memory)
        bsf       EECON1, 2   ; bit 2 = WREN (enable writes)

        bcf       INTCON, 7   ; bit 7 = GIE (disable interrupts)
        movlw     0x55
        movwf     EECON2            ; write 55h
        movlw     0xAA
        movwf     EECON2            ; write AAh
        bsf       EECON1, 1   ; bit 1 = WR (begin write, cleared when done)

loop11:
        btfsc     EECON1, 1   ; poll WR bit for writing to be done
        bra       loop11

        bra       waiting

setEnd2:
        movlw     0x02
        cpfseq    USER
        bra       setEnd3

        movff     EEADR, EEDATA     ; byte to be stored in EEPROM
        movlw     EDATAEND2
        movwf     EEADR

        bcf       EECON1, 7   ; bit 7 = EEPGD (point to data EEPROM memory)
        bcf       EECON1, 6   ; bit 6 = CFGS (access data EEPROM memory)
        bsf       EECON1, 2   ; bit 2 = WREN (enable writes)
```

```
        bcf         INTCON, 7   ; bit 7 = GIE (disable interrupts)
        movlw       0x55
        movwf       EECON2            ; write 55h
        movlw       0xAA
        movwf       EECON2            ; write AAh
        bsf         EECON1, 1   ; bit 1 = WR (begin write, cleared when done)

loop12:
        btfsc       EECON1, 1   ; poll WR bit for writing to be done
        bra         loop12

        bra         waiting

setEnd3:
        movlw       0x03

        movff       EEADR, EEDATA     ; byte to be stored in EEPROM
        movlw       EDATAEND3
        movwf       EEADR

        bcf         EECON1, 7   ; bit 7 = EEPGD (point to data EEPROM memory)
        bcf         EECON1, 6   ; bit 6 = CFGS (access data EEPROM memory)
        bsf         EECON1, 2   ; bit 2 = WREN (enable writes)

        bcf         INTCON, 7   ; bit 7 = GIE (disable interrupts)
        movlw       0x55
        movwf       EECON2            ; write 55h
        movlw       0xAA
        movwf       EECON2            ; write AAh
        bsf         EECON1, 1   ; bit 1 = WR (begin write, cleared when done)

loop13:
        btfsc       EECON1, 1   ; poll WR bit for writing to be done
        bra         loop13

        bra         waiting


oneSecDelay:

; a subroutine that takes approx 1 second to execute


        clrf        INDEX1


for1:                               ; repeat for INSA iterations
        movlw       INSA
        cpfslt      INDEX1
        bra         break1
        incf        INDEX1

        incf        INDEX3
        incf        INDEX3

        clrf        INDEX2
```

```
        movlw     INSB

for2:                                ; repeat for INSB iterations
        cpfslt    INDEX2
        bra       break2
        incf      INDEX2

        incf      INDEX3
        incf      INDEX3
        incf      INDEX3
        incf      INDEX3
        incf      INDEX3

        bra       for2
break2:
        bra       for1

break1:

        return


; a subroutine that takes approx .125 second to execute

delay:  movlw     0x00
        movwf     counter1
        movwf     counter2

dloop1:
        movf      counter1, 0
        sublw     0x19
        bz        done
        incf      counter1
        clrf      counter2

dloop2:
        movf      counter2, 0
        sublw     0xFA
        bz        dloop1
        nop
        nop
        nop
        nop
        incf      counter2
        bra       dloop2

done:   return


        end
```