AN727

Credit Card Reader Using a PIC12C509

Author: Andrew M Errington

INTRODUCTION

Many people carry one or more magnetically encoded cards with them for accessing a range of services. Perhaps the most common example is the credit card or bank ATM card, but increasingly they are being used for access control, employee time logging, customer loyalty schemes, club membership and other applications. This application note describes data formats found on these cards, and details an algorithm to decode that data with suitable hardware.

Often a card reader will be incorporated into a point-of-sale (POS) terminal or built into a security device. In a larger system, the reader control electronics may be integrated with other devices, however it is often useful to use a microcontroller to decode the data from the card remotely and send the data via a serial link to a host processor. The PIC12C509 is a good example of a suitable microcontroller, as it requires few external components, and is small enough to be incorporated into the card reader assembly itself. This allows a self-contained card reader with serial data output to be built.

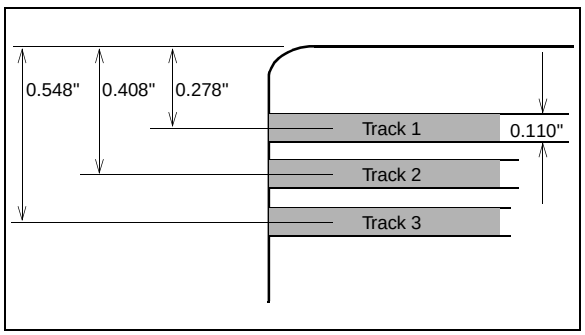
This document details the typical data found on a bank card, but there are also many custom-encoded cards in circulation for other purposes. However, even these cards usually conform to the ISO Track 2 standard which will be described later.

DATA ENCODING

Data is encoded on the magnetic stripe on a card in a similar way to original computer tape systems, mainly because at the time they were introduced, tape technology was widely available. While the details of the card data formats given here are brief, a full description of how the data is physically encoded on the magnetic stripe can be found in International Standards Organization document ISO7811/2-1985. In the US, this is also known as ANSI x 4.16 -1983, and in the UK, as BS7106:Part 2:1989. Full specifications for all aspects of "identification cards", including the physical size of the card and the embossed information on the front, can be found in ISO7811 parts 1 to 6.

The magnetic stripe on bank cards and credit cards typically has three tracks of information. Track 1 is alphanumeric and usually records the card holder's name in addition to their card number. Track 2 (the center track) is numeric only and encodes the card holder's card number and expiration date. Track 3 is typically not used, or reserved for the card organization's own use, and has many different data encoding standards. To read each track requires a magnetic reading head at the appropriate distance from the edge of the card. The position of each track is shown in Figure 1.

FIGURE 1: POSITION OF ISO TRACKS 1, 2 AND 3



This application note deals specifically with data encoded on Track 2. This data is numeric only and so is compact and easy to read, and there are many card reading modules with a single head in the Track 2 position available. In recent years, there has been a trend for organizations to read data from Track 1, thus allowing POS terminals to display the cardholder's name on the receipt.

Most card readers have three wires for data output, plus of course, one each for power and ground. They are typically powered from a 5V DC supply with TTL compatible output signals. Inside the reader assembly is a magnetic reader head, like a cassette tape head. A small circuit converts the analog signal from the head into clock and data signals, and a signal to indicate a card is present. For this application note, a Panasonic card reader (part no. PCR100-ND from Digi-Key) is used. Typically the signals are all active low, which means a high voltage (+5V) represents logic '0' and a low voltage (0V) represents logic '1'. Table 1 shows details of the connection to the reader module, with wire colors for the Panasonic interface cable (Digi-Key PCR101-ND).

TABLE 1: CARD READER MODULE CONNECTIONS

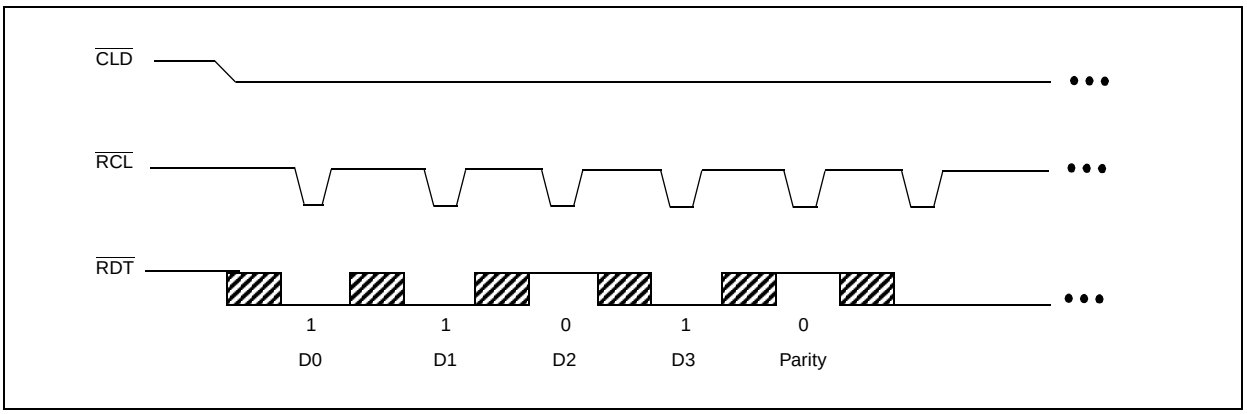
Wire Color	Function	Description
Brown	$\overline{CLD}$	Card presence indicator. When low, a card is in the reader.
Red	$\overline{RCL}$	Clock signal. When low, the data bit on the $\overline{RDT}$ pin is valid.
Orange	$\overline{RDT}$	Data signal. Data bits are read from the card sequentially and output on this pin. When low, the data bit is a '1' and when high, it is a '0'. The data is only valid when the $\overline{RCL}$ pin is low.
Yellow	+5V supply	Connect to power supply.
Green	0V	Connect to ground.
Blue	Frame Ground	Connect to ground if necessary.

For a reader with more than one read head there will be more than one clock and data line, and the software to read the card becomes more complicated.

Figure 2 shows a representation of the signals generated by the card reader as a card is passed through. First, $\overline{CLD}$ goes low to indicate a card is in the reader, then a series of pulses on $\overline{RCL}$ indicate when the data on the $\overline{RDT}$ pin is valid. The sequence shown is for the first character on Track 2, which is the start sentinel 1011b. This is encoded LSb first and followed by a parity bit.

Since the card is being passed through the reader by a human, the timing of the $\overline{RCL}$ pulse will be irregular, but the speed of the card as it passes the read head is so slow with respect to the operation of the microcontroller's sampling loop, that this is not really a problem.

FIGURE 2: CARD READER SIGNALS



CARD DATA FORMAT

Data is encoded LSb first on the three tracks as follows:

Track 1 - IATA

The data standard for Track 1 was developed by the International Air Transportation Association (IATA) and contains alphanumeric information for airline ticketing or other database accesses. On a credit card, this track typically contains the cardholder's name as embossed on the front of the card. The specification allows up to 79 characters. Each character is 7 bits long, comprising a 6 bit character code and a parity bit. The data is encoded at 210 bpi (bits per inch).

SS	FC	PAN	FS	CC	Name	FS	Additional Data	ES	LRC

SS ..... Start Sentinel
FC ..... Format Code
PAN ..... Primary Account Number (19 digits max.)
FS ..... Field Separator
CC ..... Country Code (3 characters max.)
Name ..... Name (26 characters max.)
ES ..... End Sentinel
LRC ..... Longitudinal Redundancy Check Character

Additional Data:

Expiration Date (4 characters)

Interchange Designator (1 Character)

Service Code (2 characters)

Custom Data

Track 2 - ABA

The data standard for Track 2 was developed by the American Bankers Association (ABA), and contains numeric information only. On a credit card, this track typically contains the cardholder's credit card number as embossed on the front of the card. The specification allows up to 40 digits. Each digit is 5 bits long, comprising a 4-bit BCD digit and a parity bit. The data is encoded at 75 bpi.

SS	PAN	FS	Additional Data	ES	LRC

SS ..... Start Sentinel
PAN ..... Primary Account Number (19 digits max.)
FS ..... Field Separator
ES ..... End Sentinel
LRC ..... Longitudinal Redundancy Check Character

Additional Data:

Country Code (3 characters)

Expiration Date (4 characters)

Interchange Designator (3 Character)

Service Code (3 characters)

Custom Data

Track 3 - THRIFT

The data standard for Track 3 was developed by the Thrift Industry, and contains numeric only information which may be re-recorded or updated when the card is used. There are many different uses and specifications for Track 3, so no details are shown here. The Track 3 specification allows up to 107 digits. Each digit is 5 bits long, a 4-bit BCD digit and a parity bit. The data is encoded at 210 bpi.

While the Primary Account Number (PAN) can be up to 19 digits, a MasterCard PAN is variable up to 16 digits, and VISA is 13 or 16 digits, including a modulo-10 check digit.

Each of the three specifications includes three special characters: a start sentinel, an end sentinel and an LRC (Longitudinal Redundancy Check) character. This means that the actual number of characters that can be stored is three less than the maximum specified. The sentinel codes are special character codes that are used to tell the microprocessor that it is reading the data where the start and end of the data is. Any unused space before or after the data on the card is filled with zeroes. The LRC character provides one of the error detection mechanisms described below.

ERROR DETECTION

There are two error detection methods incorporated into the data encoding standard. The first is parity checking. For alphanumeric data, there are 7 bits per character. The lower 6 bits are the character itself, and the MSb is a parity bit. Each character is encoded with odd parity, meaning that the total number of '1's in the character will be odd. Similarly for numeric data, there are 5 bits per character, 4 are the character itself, and the MSb is the parity bit. This is shown in Table 2 and Table 3. To check the parity, count the number of '1's in each character as it is read from the card. If the count is even, then there was a parity error when reading that character.

The LRC is a character code which is calculated when the card is written and checked when the card is read. It contains a bit pattern that makes the total number of '1's encoded in the corresponding bit location of all the characters (including the start and end sentinel and the LRC itself) in the data on that track even. To check the LRC, XOR all of the character codes, ignoring the parity bits, as they are read from the card, including the start and end sentinels and the LRC itself. The result (excluding the parity bit) should be zero.

The reason for having two error detection methods is to make the error detection more robust. For example, if a character is read with two bits wrong, the parity will appear to be okay, but the LRC check will fail.

CHARACTER SET

TABLE 2: TRACK 1 AT 7 BITS PER CHARACTER (PARITY BIT NOT SHOWN)

	D5:D4			
D3:D0	**00**	**01**	**10**	**11**
0000	SPC	0		P
0001		1	A	Q
0010		2	B	R
0011		3	C	S
0100	$	4	D	T
0101	%(start sentinel)	5	E	U
0110		6	F	V
0111		7	G	W
1000	(	8	H	X
1001	)	9	I	Y
1010			J	Z
1011			K	[
1100			L	/
1101	-		M	]
1110	.		N	^(separator)
1111	/	?(end sentinel)	O	

Characters not shown are not supported in the alpha-numeric character set, although they may appear on the card in the LRC position. The three shaded characters may differ for national character sets.

TABLE 3: TRACK 2 AND 3 AT 5 BITS PER CHARACTER (PARITY BIT SHOWN)

P	D3	D2	D1	D0	
1	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	2
1	0	0	1	1	3
0	0	1	0	0	4
1	0	1	0	1	5
1	0	1	1	0	6
0	0	1	1	1	7
0	1	0	0	0	8
1	1	0	0	1	9
1	1	0	1	0	
0	1	0	1	1	Start Sentinel
1	1	1	0	0	
0	1	1	0	1	Separator
0	1	1	1	0	
1	1	1	1	1	End Sentinel

Data Decoding

Knowing what the card reader signals mean, and how the characters are encoded, makes it a simple matter to devise an algorithm to decode the data from a card when it is swiped through the reader. The `card509.asm` file for the PIC12C509 is very compact and may be easily adapted for other PICmicro® devices.

The program is designed to read data from a Track 2 magnetic card reader, because equipment for reading this track is widely available, and Track 2 has a manageably small number of bits encoded. This is important since the data is buffered before sending it out via the serial port. As each character is read, its parity bit is checked, then stored in a memory buffer. After reading the end sentinel, the LRC is read and checked, and all of the data characters in the buffer are sent out serially from an I/O pin. If there are any parity errors, an error flag is set, and if the LRC check is bad, then another error flag is set. The state of these two flags is indicated by two characters sent after the card data. On power-up and after every card read, the PICmicro device sends 'Ready' from the serial port.

The card data is stored in a memory buffer, which is a block of data memory not used by any variable in the program. Since each character is a 4-bit BCD digit, each byte can hold two characters, so 20 bytes are reserved, enough to hold 40 characters. In fact, space is only needed for 37 characters as there is no need to store the start and end sentinels or the LRC character.

On power-up, all of the program memory from 0x07 to 0x0F is cleared. This is not strictly necessary, but some registers are not specifically initialized on reset and may contain random data on power up.

The main loop starts by clearing the memory buffer and initializing the memory pointers and other variables. When starting, the `bad_LRC` flag is set and it is only cleared if a bad LRC is not found after reading all the data from the card. Also, the 4 bits that hold the LRC check in the parity register are initialized to the same bit pattern as the start sentinel. This is because the start sentinel is never stored, but must be included in the LRC calculation.

Next, the program waits to see if a card is present and loops indefinitely while the $\overline{CLD}$ signal is high. Once it is low, the program drops through to another loop to wait for the $\overline{RCL}$ line to go low. When the $\overline{RCL}$ line is low, the data line is valid, so the $\overline{RDT}$ line can be tested. Remember the $\overline{RDT}$ line is active low, so if it is high, the data on the card is a '0' and the carry flag is cleared. If $\overline{RDT}$ is low, the data on the card is a '1', therefore the carry flag is set and the parity bit in the `parityLRC` register is toggled. Toggling the parity bit is like having a one bit counter, which is all that is needed to count '1's and see if there is an odd number.

A byte of memory is reserved as an input buffer, `char_buf`, and a single bit as a flag, `found_start`. Each time a bit is read from the reader, it is placed in the carry flag as described above, and the input buffer is shifted right to roll the bit from the carry flag into the MSb. This means that all of the characters are formed in the upper 5 bits of `char_buf`. The `found_start` flag is cleared at the beginning of the main loop, and while it is clear, the top 5 bits are checked for the start sentinel bit pattern every time a new bit is rotated in from the reader. As soon as the start sentinel is seen, the `found_start` flag is set.

When the carry flag is rotated into the input buffer with RRF `char_buf`, the LSb of `char_buf` rotates out into the carry. Until the start sentinel is seen, the low three bits of `char_buf` are continually cleared, so a zero rotates out and is of no concern. Once the start sentinel is seen, the bits that were read need to be grouped into 5-bit characters. This is done by setting bit 4 of `char_buf` when ready to read a new character. When 5 bits have been rotated in from the reader, the bit that was set will be rotated out into the carry flag. This bit is known as a sentinel bit (not to be confused with the start and end sentinels on the card).

The carry flag is checked to see if the sentinel bit has rolled out, and if it has, then the top 5 bits of `char_buf` contain a character from the card. The program checks the parity (by looking at the parity bit in the `parityLRC` register), then XORs the character with the LRC to update it. If the character is not the end sentinel or LRC, the parity bit is discarded and the 4-bit character is stored in the memory buffer. If it was the end sentinel, a flag (`found_end`) is set to show that the next character will be the LRC and that it's possible to finish.

When the last character (the LRC) has been read, the program jumps to the `dump_buffer` routine, or if the buffer has been filled, sets the `buf_end` flag, which causes a jump there.

Characters are stored and fetched from the memory buffer by the `get_put_char` routine. The variable `buf_ptr` effectively points to a particular nibble in the PIC12C509 register banks. The `read_buf` flag indicates whether to store or fetch from the buffer, and the character is moved between `char_buf` and the buffer accordingly. The buffer locations are not in a contiguous address space and some care must be taken to deal with register banks correctly.

The `dump_buffer` routine loops through the memory buffer address space, takes each character (each nibble), converts it to an ASCII code and then calls the `send_char` routine to send the character out serially. If the PICmicro device is connected to a serial port on a PC running a terminal program, the data from the card will appear in the terminal window. When all characters have been sent, a 'P' is sent if there was a parity error, and an 'L', if there was an LRC error. If there were no errors, a period '.' is sent, then the program loops back, clears the buffer, and waits for another card.

A simple serial output routine, `send_char,` sends the character code held in the W register serially from an output pin. It is timed to run at 1200 baud, no parity, 1

stop bit with a 4 MHz oscillator, and the PC serial port should be configured appropriately to receive it. The `send_char` routine could easily be replaced by a routine that displays the character on an LCD module, for example. Higher baud rates could be achieved using an external crystal, but the internal oscillator has been seen to be stable enough to run at 1200 baud with no errors.

CIRCUIT LAYOUT

As can be seen from Figure 3, there is very little to do other than wire up the PICmicro device directly to the reader. The 5V supply can be taken from a bench Power Supply Unit, or a 9V battery and a voltage regulator. The connections to the reader and to the PC serial port should match the I/O pin declarations in the code. For this application the PICmicro device should be programmed for internal oscillator mode, internal $\overline{\text{MCLR}}$ and watchdog disabled. The connections to the PC via the serial port should use a level shift IC, such as the MAX232A from Maxim. (This device is not shown in Figure 3.)

A simple program is included, `CARDLOG.EXE`, which monitors a PC serial port for data from the reader circuit shown. It records each unique card number as it is seen, and logs the date and time it was used, together with a notification when the number was seen for the first time. Once the card numbers are in a list in a PC program, they can be easily manipulated for the applications mentioned earlier in this document. However, it would be possible to extend the `CARD509.ASM` program to store card numbers in an external EEPROM, for example, or to verify card numbers read against those stored in an EEPROM to construct a stand-alone access control system or card data logger.

CONCLUSION

Although smart cards are gaining greater acceptance, magnetic cards have been around for some time, and it seems they will remain in use for a few years to come. This application note demonstrates the simplicity of reading magnetic card data using a low-cost embedded microcontroller, and interfacing to a larger, more complex system for many diverse applications.

FIGURE 3: CIRCUIT SIMPLIFIED BLOCK DIAGRAM

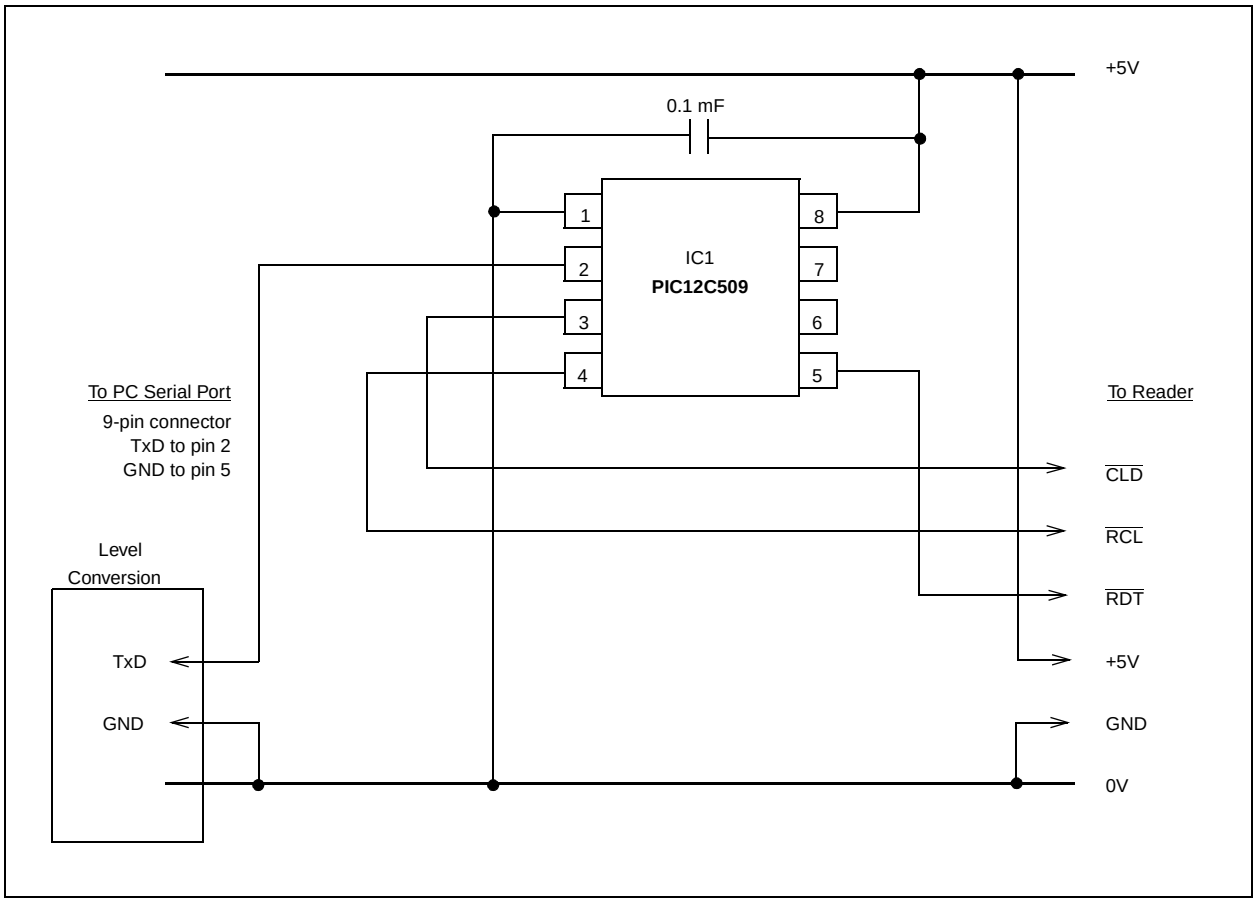
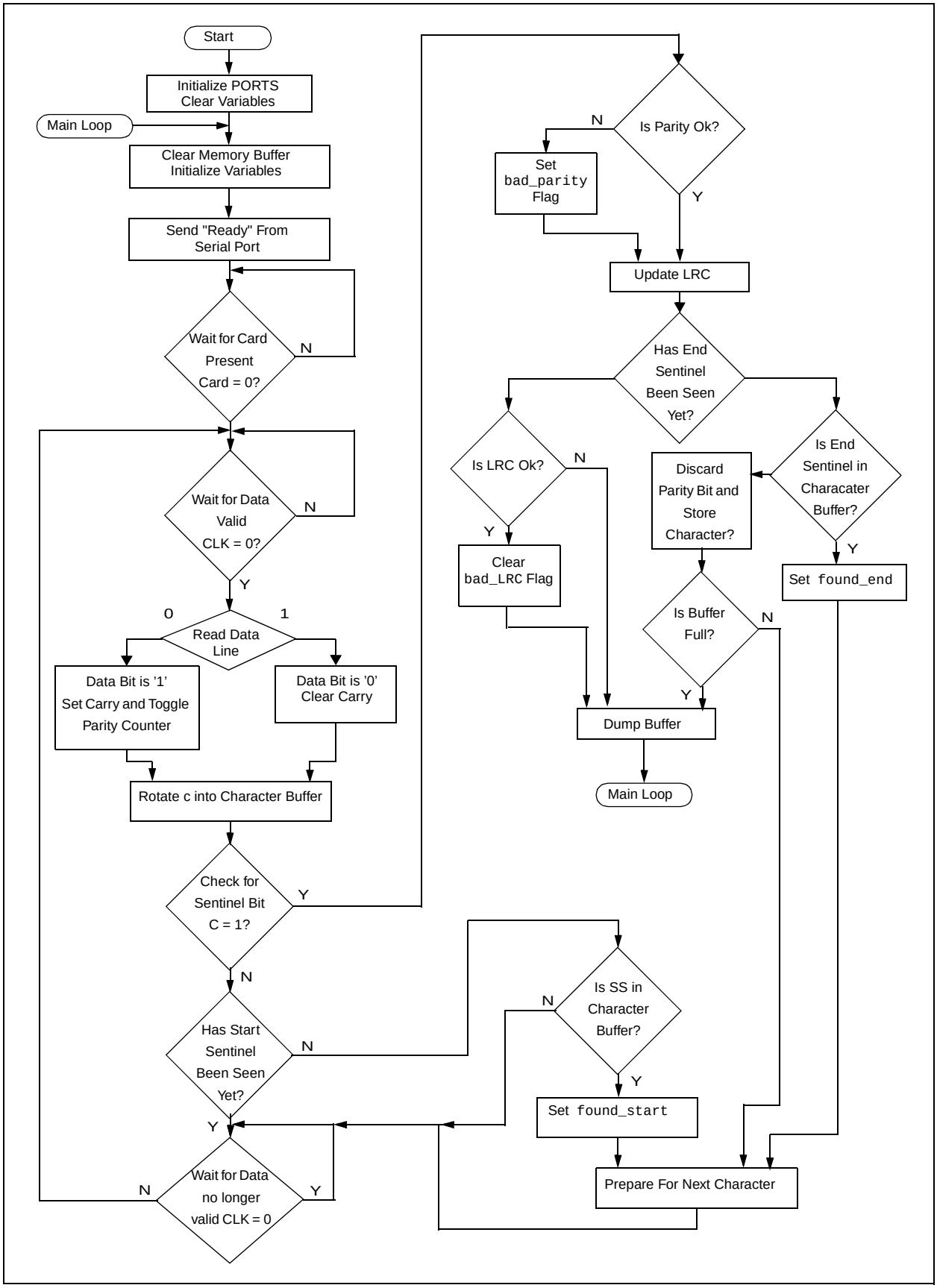


FIGURE 4: SIMPLIFIED FLOW DIAGRAM



APPENDIX A: SOURCE CODE

```
MPASM 01.50 Released         CARD509.ASM   9-29-1999  14:09:04         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ; Card Reader
                     00002 ;
                     00003 ; Written by    A M Errington
                     00004 ; Device        PIC 12C509 (8 pins)
                     00005 ; Clock speed   1Mhz Tcy=1us
                     00006 ; Resonator     4MHz internal RC
                     00007 ; Reset circuit Internal MCLR
                     00008 ; Watchdog      Disabled
                     00009 ;
                     00010 ; Uses Panasonic track 2 card reader or any typical magnetic
                     00011 ; card reader mechanism.  Sends data out serially at 1200bd.
                     00012 ;
                     00013 ; This software is Copyright 1998 Andrew M Errington
                     00014 ;
                     00015 ; This file is best viewed with hard tabs set to 4 character
                     00016 ; spacing.
                     00017
                     00018     TITLE "Card Reader"
                     00019
                     00020     LIST C=120, b=4
                     00021     LIST P=12C509
                     00022
                     00023     ERRORLEVEL -305         ; Suppress "Using default
                     00024                             ; destination of 1 (file)."
                     00025
                     00026 ; *****************************************************************
                     00027 ; General Equates
                     00028
                     00029 ; PIC12C509 standard registers
  00000000           00030 INDF        EQU 0x00
  00000001           00031 TMR0        EQU 0x01
  00000002           00032 PC          EQU 0x02
  00000003           00033 STATUS      EQU 0x03
  00000004           00034 FSR         EQU 0x04
  00000005           00035 OSCCAL      EQU 0x05
  00000006           00036 GPIO        EQU 0x06        ; lower 5 bits only
                     00037
                     00038
                     00039 ; I/O port bits
  00000000           00040 GP0         EQU 0x00
  00000001           00041 GP1         EQU 0x01
  00000002           00042 GP2         EQU 0x02        ; Shared with T0CKI
  00000003           00043 GP3         EQU 0x03        ; Always input, shared with MCLR, Vpp
  00000004           00044 GP4         EQU 0x04        ; Shared with OSC2
  00000005           00045 GP5         EQU 0x05        ; Shared with OSC1, clkin
                     00046
                     00047
                     00048 ; Status register bits
  00000000           00049 C           EQU 0x00        ; Carry flag
  00000001           00050 DC          EQU 0x01        ; Digit carry flag
  00000002           00051 Z           EQU 0x02        ; Zero flag
  00000003           00052 PD          EQU 0x03        ; Power down flag
  00000004           00053 TO          EQU 0x04        ; WDT timeout flag
```

```
MPASM 01.50 Released         CARD509.ASM   9-29-1999  14:09:04         PAGE  2
Card Reader

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

  00000005          00054 PA0         EQU 0x05        ; Program page select
  00000007          00055 GPWUF       EQU 0x07        ; GPIO reset bit
                    00056
                    00057
                    00058 ; Other control bits
  00000005          00059 RP0         EQU 0x05        ; Register page select
                    00060
                    00061
                    00062 ; Other useful constants
  00000000          00063 LSB         EQU 0x00
  00000007          00064 MSB         EQU 0x07
                    00065
                    00066
                    00067 ; Useful PIC macros
                    00068 ; The BTFSS and BTFSC instructions can be confusing to read.  It
                    00069 ; is easier to read the code with macros for IFSET and IFCLR
                    00070 ; meaning the opposite of SKPCLR and SKPSET respectively.
                    00071
                    00072 #DEFINE IFSET   BTFSC
                    00073 #DEFINE IFCLR   BTFSS
                    00074
                    00075 #DEFINE IFZ     SKPNZ
                    00076 #DEFINE IFNZ    SKPZ
                    00077 #DEFINE IFC     SKPNC
                    00078 #DEFINE IFNC    SKPC
                    00079
                    00080 #DEFINE SKPSET  BTFSS
                    00081 #DEFINE SKPCLR  BTFSC
                    00082
                    00083
                    00084 ; *****************************************************************
                    00085 ; Card reader constants
                    00086
  00000058          00087 start_code  EQU b'01011000'     ; Start Sentinel bit pattern,
                    00088                                 ; shifted up into the top 5 bits
                    00089
  000000F8          00090 end_code    EQU b'11111000'     ; End Sentinel bit pattern,
                    00091                                 ; shifted up into the top 5 bits
                    00092
                    00093
                    00094 ; PIC12C509 RAM location usage
                    00095
  00000007          00096 buf_ptr     EQU 0x07    ; card data buffer pointer (nibbles)
  00000008          00097 num_chr     EQU 0x08    ; Number of characters read from card
  00000009          00098 count       EQU 0x09    ; General 8 bit counter
  0000000A          00099 flag        EQU 0x0A    ; Control flags
  0000000B          00100 char_buf    EQU 0x0B    ; Character buffer, input and serial output
  0000000C          00101 parityLRC   EQU 0x0C    ; Parity/LRC workspace
  0000000D          00102 temp        EQU 0x0D    ; Temporary workspace
                    00103 ;           EQU 0x0E    ; unused
                    00104 ;           EQU 0x0F    ; unused
  00000010          00105 lo_mem      EQU 0x10    ; Memory buffer start address:
                    00106 ;           EQU 0x11    ; Track 2 of the magnetic card contains
```

```
MPASM 01.50 Released         CARD509.ASM   9-29-1999  14:09:04         PAGE  3
Card Reader

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00107 ;           EQU 0x12    ; at most 40 4-bit characters, including
                     00108 ;           EQU 0x13    ; the start sentinel, end sentinel and
                     00109 ;           EQU 0x14    ; LRC, so 20 bytes are reserved to store
                     00110 ;           EQU 0x15    ; all of them.  In fact only 37 nibbles
                     00111 ;           EQU 0x16    ; are used as the start and end sentinels
                     00112 ;           EQU 0x17    ; and the LRC are never stored in this
                     00113 ;           EQU 0x18    ; application
                     00114 ;           EQU 0x19
                     00115 ;           EQU 0x1A
                     00116 ;           EQU 0x1B
                     00117 ;           EQU 0x1C
                     00118 ;           EQU 0x1D
                     00119 ;           EQU 0x1E
                     00120 ;           EQU 0x1F
                     00121
                     00122 ; 0x20 to 0x2F are mapped to 0x00 to 0x1F, so the buffer
                     00123 ; continues from 0x30 onwards
                     00124
                     00125 ;           EQU 0x30
                     00126 ;           EQU 0x31
                     00127 ;           EQU 0x32
  00000033           00128 hi_mem      EQU 0x33    ; Memory buffer end
                     00129 ;           EQU 0x34    ; unused
                     00130 ;           EQU 0x35    ; unused
                     00131 ;           EQU 0x36    ; unused
                     00132 ;           EQU 0x37    ; unused
                     00133 ;           EQU 0x38    ; unused
                     00134 ;           EQU 0x39    ; unused
                     00135 ;           EQU 0x3A    ; unused
                     00136 ;           EQU 0x3B    ; unused
                     00137 ;           EQU 0x3C    ; unused
                     00138 ;           EQU 0x3D    ; unused
                     00139 ;           EQU 0x3E    ; unused
                     00140 ;           EQU 0x3F    ; unused
                     00141
                     00142
                     00143 ; Derived constants
                     00144
                     00145 ; buf_sz is the actual number of nibbles available in the
                     00146 ; buffer.  If the buffer continues into Bank 1 care must be
                     00147 ; taken to correct for the discontinuity in address space.
                     00148 ; Here the assembler signals an error if portions of the buffer
                     00149 ; are in the wrong banks.
                     00150
                     00151     if lo_mem > 0x1F
                     00152
                     00153 ERROR "Buffer start address (lo_mem) must be in Bank 0"
                     00154
                     00155     endif
                     00156
                     00157     if hi_mem > 0x1F && hi_mem < 0x30
                     00158
                     00159 ERROR "Buffer end address (hi_mem) must be in upper half of Bank 1"
```

```
MPASM 01.50 Released         CARD509.ASM   9-29-1999  14:09:04         PAGE  4
Card Reader

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00160
                     00161     endif
                     00162
                     00163     if hi_mem <= 0x1F
                     00164
                     00165 buf_sz      EQU ((hi_mem - lo_mem) + 1) * 2
                     00166
                     00167     else
                     00168
  00000028           00169 buf_sz      EQU ((hi_mem - lo_mem) - .15) * 2
                     00170
                     00171     endif
                     00172
                     00173
                     00174 ; Flag register bit meanings
                     00175
                     00176 ; bit   7  6  5  4  3  2  1  0 -> found start sentinel
                     00177 ;       |  |  |  |  |  |  +-----> found end sentinel
                     00178 ;       |  |  |  |  |  +--------> bad parity
                     00179 ;       |  |  |  |  +-----------> bad LRC
                     00180 ;       |  |  |  +--------------> reached end of buffer
                     00181 ;       |  |  +-----------------> R/^W flag for buffer operations
                     00182 ;       |  +--------------------> unused
                     00183 ;       +-----------------------> unused
                     00184
  00000000           00185 found_start EQU 0
  00000001           00186 found_end   EQU 1
  00000002           00187 bad_parity  EQU 2
  00000003           00188 bad_LRC     EQU 3
  00000004           00189 buf_end     EQU 4
  00000005           00190 read_buf    EQU 5
                     00191
                     00192
                     00193 ; ParityLRC register bit meanings
                     00194
                     00195 ; bit   7  6  5  4  3  2  1  0 -> unused
                     00196 ;       |  |  |  |  |  |  +-----> unused
                     00197 ;       |  |  |  |  |  +--------> unused
                     00198 ;       |  |  |  |  +-----------> LRC bit 0
                     00199 ;       |  |  |  +--------------> LRC bit 1
                     00200 ;       |  |  +-----------------> LRC bit 2
                     00201 ;       |  +--------------------> LRC bit 3
                     00202 ;       +-----------------------> parity bit
                     00203
                     00204 ; Note: Later code relies on these bits remaining in this position
                     00205
                     00206
                     00207 ; I/O pin declarations
                     00208
                     00209 ; The card reader connects to +5V and 0V, and has three signal
                     00210 ; lines connected to the following I/O pins:
                     00211
  00000000           00212 ser_out     EQU GP0     ; serial TxD pin to host
```

```
MPASM 01.50 Released         CARD509.ASM   9-29-1999  14:09:04         PAGE  5
Card Reader

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

  00000001           00213 card        EQU GP1     ; ^CLD signal (low when card present)
  00000002           00214 clock       EQU GP2     ; ^RCL signal (low when data valid)
  00000003           00215 signal      EQU GP3     ; ^RDT signal from magstripe
                     00216
                     00217 ; GP4 and GP5 are still available for I/O or for a crystal if
                     00218 ; required.
                     00219
                     00220
                     00221 ; Compilation options
                     00222
                     00223 ; The invert_tx option changes the sense of the ser_out line
                     00224
  00000000           00225 invert_tx   EQU 0       ; 0 = Idle (logical '1') is 0V
                     00226
                     00227
                     00228 ; *************************************************************
                     00229 ; Program code starts here
                     00230 ; *************************************************************
                     00231
                     00232 ; The PIC12C509 reset vector jumps to top of memory, then the
                     00233 ; program counter rolls to 0x00 after loading the RC osc.
                     00234 ; calibration value into W
                     00235
0000                 00236         ORG     0x00
0000 0025            00237         MOVWF   OSCCAL
                     00238
0001 0A31            00239         GOTO    start
                     00240
                     00241
                     00242 ; *************************************************************
                     00243 ; Subroutines
                     00244 ;
                     00245
                     00246
                     00247 ; *************************************************************
0002                 00248 send_char
                     00249
                     00250 ; Call send_char with an ASCII character code in W.  This is a
                     00251 ; simple serial output routine which sends the character out
                     00252 ; serially on an output pin at 1200 baud, 8 data bits, no parity,
                     00253 ; 1 stop bit.  Assume the PIC oscillator is running at 4MHz.
                     00254 ;
                     00255 ; The baud rate of 1200 baud was chosen as it will work with the
                     00256 ; 12C509 internal RC oscillator generating the timing.  Higher
                     00257 ; baud rates require tighter timing tolerance, and will therefore
                     00258 ; require a crystal.
                     00259 ;
                     00260 ; Normally serial communication (RS232) requires a negative
                     00261 ; voltage between -5V to -15V to represent a '1', and a positive
                     00262 ; voltage between +5V and +15V to represent a '0'.  Most PC
                     00263 ; serial ports will switch at +/-3V, and in fact will often work
                     00264 ; with 0V and 5V, so it is possible to use a PIC I/O pin, set
                     00265 ; high for a logic '0' and low for a logic '1'.  A 1k resistor
```

```
MPASM 01.50 Released         CARD509.ASM   9-29-1999  14:09:04         PAGE  6
Card Reader

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00266 ; placed in series with the Tx line will limit the current, and
                     00267 ; This is probably acceptable for experimental purposes.  For
                     00268 ; robustness, however, it may be desirable to include level shift
                     00269 ; IC, such as the MAX232A from Maxim.  The invert_tx compilation
                     00270 ; option can be used to alter the sense of transmitted bits if
                     00271 ; necessary.
                     00272 ;
                     00273 ; At 1200 baud each bit cell is just over 833us in length.  At
                     00274 ; power up the serial output line is set to its idle state (logic
                     00275 ; '1').  At the start of transmission it is taken to logic '0'
                     00276 ; for one bit time to generate the start bit.  Next, the 8 bits
                     00277 ; of character data are shifted out, LSB first, by rolling them
                     00278 ; down into the carry.  The program sets or clears the serial
                     00279 ; line pin according to whether the carry represents a logic '0'
                     00280 ; or '1'.  Finally the line is held at logic '1' for at least one
                     00281 ; bit time for the stop bit.  The line then rests at this state
                     00282 ; (idle) until it is time to send the next byte
                     00283 ;
                     00284 ; Bit cell timing is done by counting clock cycles: 1 instruction
                     00285 ; is 1us, jumps and skips are 2us.
                     00286
0002 002B            00287         MOVWF   char_buf        ; Store the character code (in W)
                     00288                                 ; to character buffer
                     00289
0003 0C0A            00290         MOVLW   .10             ; Set the number of bits (including
0004 0029            00291         MOVWF   count           ; start and stop bits) in count
                     00292
0005 0403            00293         CLRC                    ; Clear carry because the start bit
                     00294                                 ; is a '0'
                     00295
0006                 00296 bit_loop
0006 0703            00297         IFNC                    ; serial pin logic '0'
                     00298     if invert_tx
                     00299          BCF    GPIO,ser_out
                     00300     else
0007 0506            00301          BSF    GPIO,ser_out
                     00302     endif
                     00303
0008 0603            00304         IFC                     ; serial pin logic '1'
                     00305     if invert_tx
                     00306          BSF    GPIO,ser_out
                     00307     else
0009 0406            00308          BCF    GPIO,ser_out
                     00309     endif
                     00310
000A 090F            00311         CALL    bit_delay       ; Make up the bit time to 833us
                     00312
000B 032B            00313         RRF     char_buf        ; Roll LSB of char_buf into carry,
                     00314                                 ; and the '1' from the bit_delay
                     00315                                 ; routine into the MSB
                     00316
000C 02E9            00317         DECFSZ  count           ; Loop until all bits have been
000D 0A06            00318          GOTO   bit_loop        ; shifted out.
```

```
MPASM 01.50 Released         CARD509.ASM   9-29-1999  14:09:04         PAGE  7
Card Reader

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00319
000E 0800            00320         RETLW   0
                     00321
                     00322
000F                 00323 bit_delay
                     00324
                     00325 ; The bit length should be 833us in total (=833 Tcy).  This
                     00326 ; routine and the bit loop routine take 21 Tcy, leaving 812 Tcy
                     00327 ; to waste.  The delay loop is 4 cycles long, so loop for 203
                     00328 ; times.  This is done by loading a counter with 255 - 203 = 52
                     00329 ; and incrementing it every time around the loop.  When the
                     00330 ; counter reaches 255 it overflows and sets the carry flag.  As
                     00331 ; a side effect this routine returns to the bit loop just before
                     00332 ; the RRF instruction with carry set, which will roll a '1' into
                     00333 ; char_buf, which is then used as the stop bit when it rolls out
                     00334 ; again after being shifted 8 times.
                     00335
000F 0C34            00336         MOVLW   .52             ; Initialise temp
0010 002D            00337         MOVWF   temp
                     00338
0011 0C01            00339         MOVLW   .1              ; Put 1 in W for incrementing temp
                     00340
0012 0A13            00341         GOTO    $+1             ; Waste 2 cycles
0013 0000            00342         NOP                     ; Waste 1 cycle
                     00343
                     00344
0014                 00345 delay_loop
0014 01ED            00346         ADDWF   temp            ; Increment temp        1
0015 0703            00347         IFNC                    ; Did it overflow?      1
0016 0A14            00348          GOTO   delay_loop      ; No: go round again    2
                     00349                                 ;                      ---
                     00350                                 ;               time = 4 Tcy
                     00351
0017 0800            00352         RETLW   0               ; Yes: return
                     00353
                     00354
                     00355 ; *************************************************************
0018                 00356 get_put_char
                     00357
                     00358 ; This subroutine deals with buffer operations, either storing a
                     00359 ; character from char_buf to the buffer or fetching it from the
                     00360 ; buffer.  The routine uses buf_ptr (the logical buffer address)
                     00361 ; to calculate the physical address for the character.
                     00362 ;
                     00363 ; The 4 bit character will be stored at the current "memory
                     00364 ; location" in the buffer.  The buffer is a large chunk of RAM
                     00365 ; from 0x10 to 0x1F and 0x30 to 0x33.  Two "memory locations"
                     00366 ; are contained in each byte, one in each nibble.
                     00367 ;
                     00368 ; The variable buf_ptr points to the next free logical memory
                     00369 ; location, and the constants lo_mem and hi_mem record the
                     00370 ; physical start and end locations of the RAM block.  The
                     00371 ; constant buf_sz holds the number of nibbles (or "memory
```

```
MPASM 01.50 Released         CARD509.ASM   9-29-1999  14:09:04         PAGE  8
Card Reader

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00372 ; locations") that can be filled.
                     00373 ;
                     00374 ; Note there is a discontinuity in the address space, which must
                     00375 ; be dealt with when mapping the logical memory location to the
                     00376 ; physical memory address
                     00377 ;
                     00378 ; To calculate the position in memory to store the current
                     00379 ; character, divide buf_ptr by 2 and add lo_mem to give the
                     00380 ; address in RAM.  Next, check if this exceeds the 0x1F address
                     00381 ; range by checking bit 5 of the resultant address, and if
                     00382 ; necessary force the address into the 0x3n address space by
                     00383 ; setting bit 4.  Check whether buf_ptr is odd or even by
                     00384 ; examining its LSB to see whether to store the character in the
                     00385 ; upper or lower nibble.
                     00386
                     00387 ; When buf_ptr is zero it is pointing at the first "memory
                     00388 ; location", which is the low nibble of the first byte.
                     00389 ;
                     00390 ; All RAM in the memory buffer is cleared at the beginning of
                     00391 ; the main loop, so it is not necessary to clear each "memory
                     00392 ; location" before storing anything there.
                     00393 ;
                     00394 ; Note that for a 'put' operation the character arrives here in
                     00395 ; the upper nibble of char_buf, and for a 'get' operation the
                     00396 ; character is returned in the lower nibble.
                     00397
0018 0307            00398         RRF     buf_ptr,W       ; load W with buf_ptr/2.  Carry
                     00399                                 ; flag is rolled in.
                     00400
0019 0024            00401         MOVWF   FSR             ; and use the FSR to point to it
                     00402                                 ; Upper bits of FSR are forced
                     00403                                 ; to 1 so junk in Carry flag
                     00404                                 ; doesn't matter.
                     00405
001A 0C10            00406         MOVLW   lo_mem          ; add the buffer start address
001B 01E4            00407         ADDWF   FSR             ; to get the physical address to
                     00408                                 ; store the character
                     00409
001C 06A4            00410         IFSET   FSR,RP0         ; Check for overflow into the
001D 0584            00411          BSF    FSR,4           ; second register page and set
                     00412                                 ; bit 4 to move into 0x3n address
                     00413                                 ; space if necessary
                     00414
001E 06AA            00415         IFSET   flag,read_buf   ; check whether this is a read
001F 0A26            00416          GOTO   get_char        ; or write operation
                     00417
0020                 00418 put_char
0020 020B            00419         MOVF    char_buf,W      ; Move the character (in high
                     00420                                 ; nibble) to W
                     00421
0021 0707            00422         IFCLR   buf_ptr,LSB     ; except if LSB of buf_ptr is '0'
0022 038B            00423          SWAPF  char_buf,W      ; then the destination is an even
                     00424                                 ; nibble, so swap the character
```

```
MPASM 01.50 Released         CARD509.ASM   9-29-1999  14:09:04         PAGE  9
Card Reader

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00425                                ; to the low nibble
                     00426
0023 0120            00427         IORWF   INDF          ; since the buffer was cleared
                     00428                                ; OR the character into place
                     00429
0024 0C29            00430         MOVLW   buf_sz + 1     ; set limit for 'put' operation
                     00431                                ; to size of buffer
                     00432
0025 0A2C            00433         GOTO    get_put_done
                     00434
0026                 00435 get_char
0026 0200            00436         MOVF    INDF,W         ; Fetch data from buffer to W
                     00437
0027 0607            00438         IFSET   buf_ptr,LSB    ; if LSB of buf_ptr is set the
0028 0380            00439          SWAPF  INDF,W         ; desired character is an odd
                     00440                                ; nibble, so swap the nibbles
                     00441
0029 0E0F            00442         ANDLW   0x0F           ; mask off upper nibble
                     00443
002A 002B            00444         MOVWF   char_buf       ; move it to the character buffer
                     00445
002B 0208            00446         MOVF    num_chr,W      ; set limit for 'get' operation
                     00447                                ; to number of characters read
                     00448                                ; from card
                     00449
002C                 00450 get_put_done
002C 02A7            00451         INCF    buf_ptr        ; increment memory pointer.
                     00452
002D 0187            00453         XORWF   buf_ptr,W      ; check if this was the last
                     00454                                ; nibble in the buffer by
                     00455                                ; comparing against W (either
                     00456                                ; buf_sz or num_chr)
                     00457
002E 0643            00458         IFZ                    ; if it was,
002F 058A            00459          BSF    flag,buf_end   ; then set a flag
                     00460
0030 0800            00461         RETLW   0
                     00462
                     00463
                     00464 ; End of subroutines
                     00465
                     00466
                     00467 ; *************************************************************
                     00468 ; Main Program starts here
                     00469 ; *************************************************************
                     00470
0031                 00471 start
0031 0063            00472         CLRF    STATUS
                     00473
0032 0CC0            00474         MOVLW   B'11000000'    ; Disable GPIO pull-ups and wake
                     00475                                ; up feature
0033 0002            00476         OPTION
                     00477
```



```
MPASM 01.50 Released         CARD509.ASM   9-29-1999  14:09:04         PAGE 10
Card Reader

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

0034 0C0E           00478         MOVLW   B'00001110'     ; Set GPIO <1:3> as inputs...
0035 0006           00479         TRIS    GPIO            ; Note: GP3 is always input
                    00480
0036 0066           00481         CLRF    GPIO            ; GPIO outputs all 0
                    00482
                    00483     if invert_tx
                    00484          BSF    GPIO,ser_out    ; except for invert_tx condition
                    00485     endif
                    00486
                    00487                                 ; Clear RAM from 0x07 to 0x0F
                    00488                                 ; for neatness
                    00489
0037 0C07           00490         MOVLW   0x07            ; Load start address (0x07) into
0038 0024           00491         MOVWF   FSR             ; the FSR
                    00492
0039                00493 clrloop
0039 0060           00494         CLRF    INDF            ; Clear the RAM location FSR is
                    00495                                 ; pointing to
                    00496
003A 02A4           00497         INCF    FSR             ; Increment FSR to next location
                    00498
003B 0204           00499         MOVF    FSR,W           ; Check if FSR is pointing past
003C 0FD0           00500         XORLW   0x10 | 0xC0     ; its end point.  Remember MSBs
                    00501                                 ; of FSR read '11'
                    00502
003D 0743           00503         IFNZ                    ; If counter was not 0x10
003E 0A39           00504          GOTO   clrloop         ; then loop again
                    00505
                    00506
                    00507 ; Main program loop is here
                    00508
003F                00509 main_loop
                    00510                                 ; First clear memory buffer in
                    00511                                 ; the same way as above.
                    00512
003F 0C10           00513         MOVLW   lo_mem          ; Fetch buffer start address
0040 0024           00514         MOVWF   FSR
                    00515
0041                00516 clr_buf_loop
0041 0060           00517         CLRF    INDF
                    00518
0042 02A4           00519         INCF    FSR
                    00520
0043 06A4           00521         IFSET   FSR,RP0         ; If FSR points to register page 1
0044 0584           00522          BSF    FSR,4           ; set bit 4 to move into 0x3n
                    00523                                 ; address space
                    00524
0045 0204           00525         MOVF    FSR,W           ; Check for buffer end address.
0046 0FF4           00526         XORLW   (hi_mem + 1) | 0xC0
                    00527
0047 0743           00528         IFNZ                    ; If not end then loop around
0048 0A41           00529          GOTO   clr_buf_loop
                    00530
```

```
MPASM 01.50 Released         CARD509.ASM   9-29-1999  14:09:04         PAGE 11
Card Reader

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

0049 0067           00531         CLRF    buf_ptr         ; Initialise buffer pointer to 0
                    00532
004A 0C28           00533         MOVLW   buf_sz          ; Initialise the number of
004B 0028           00534         MOVWF   num_chr         ; characters read to the maximum
                    00535                                 ; in case of overflow later
                    00536
004C 0C58           00537         MOVLW   start_code      ; Initialise the LRC to the start
004D 002C           00538         MOVWF   parityLRC       ; sentinel code.
                    00539
004E 006A           00540         CLRF    flag            ; Initialise control flags to
004F 056A           00541         BSF     flag,bad_LRC    ; zero then set the bad_LRC
                    00542                                 ; flag.  Assume the LRC is bad
                    00543                                 ; until the check at the end.
                    00544
0050 0C52           00545         MOVLW   'R'             ; Send "Ready" from serial port
0051 0902           00546         CALL    send_char
                    00547
0052 0C65           00548         MOVLW   'e'
0053 0902           00549         CALL    send_char
                    00550
0054 0C61           00551         MOVLW   'a'
0055 0902           00552         CALL    send_char
                    00553
0056 0C64           00554         MOVLW   'd'
0057 0902           00555         CALL    send_char
                    00556
0058 0C79           00557         MOVLW   'y'
0059 0902           00558         CALL    send_char
                    00559
                    00560
005A 0C0D           00561         MOVLW   .13             ; Send CR LF from serial port
005B 0902           00562         CALL    send_char
                    00563
005C 0C0A           00564         MOVLW   .10
005D 0902           00565         CALL    send_char
                    00566
005E 006B           00567         CLRF    char_buf        ; Clear character input buffer
                    00568
005F                00569 wait_card
005F 0626           00570         IFSET   GPIO,card       ; Check ^CARD line
0060 0A5F           00571          GOTO   wait_card       ; if it's high then keep waiting
                    00572
                    00573
                    00574 ; ^CARD is low, so a card has started passing through the reader
                    00575
0061                00576 wt_clk_lo
0061 0646           00577         IFSET   GPIO,clock      ; Check ^CLK line
0062 0A61           00578          GOTO   wt_clk_lo       ; If it's high then keep waiting
                    00579
                    00580
                    00581 ; ^CLK is low, so valid data is on the ^DATA pin.  If ^DATA is
                    00582 ; low the data bit on the card is a '1', so set carry and toggle
                    00583 ; the parity bit counter.  If ^DATA is high the data bit on the
```

```
MPASM 01.50 Released         CARD509.ASM   9-29-1999  14:09:04         PAGE 12
Card Reader

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00584 ; card is a '0', so clear the carry.  Roll the carry flag into
                     00585 ; the character buffer.
                     00586 ;
                     00587 ; All this data processing must be done as quickly as possible,
                     00588 ; but fortunately the card is being swiped by a human, so from
                     00589 ; the micro's point of view it is all happening very slowly.
                     00590
0063                 00591 chk_data
0063 0666            00592         IFSET   GPIO,signal     ; Check ^DATA
0064 0A69            00593          GOTO   data_0          ; If it's high, data bit is '0'
                     00594
0065                 00595 data_1
0065 0503             00596        BSF     STATUS,C        ; Otherwise it's low so data bit
                     00597                                 ; is '1', so set carry flag
                     00598
0066 0C80            00599         MOVLW   0x80            ; and toggle parity bit in
0067 01AC            00600         XORWF   parityLRC       ; parityLRC register
                     00601
0068 0703            00602         BTFSS   STATUS,C        ; Use that fact that carry is
                     00603                                 ; set to skip the next line.
                     00604
0069                 00605 data_0
0069 0403            00606         BCF     STATUS,C        ; bit is '0', so clear carry
                     00607
006A                 00608 store_bit
006A 032B            00609         RRF     char_buf        ; shift data bit in carry flag
                     00610                                 ; into the input buffer, and
                     00611                                 ; shift LSB out into carry flag.
                     00612
                     00613 ; If the start sentinel code has not yet been seen the LSB will
                     00614 ; have been '0', so carry will be '0'.  If the start sentinel
                     00615 ; code has been seen then there will have been a sentinel bit
                     00616 ; set in char_buf which falls out after shifting 5 bits (one
                     00617 ; character) in.
                     00618
006B 0603            00619         IFC                     ; So, check the carry flag
006C 0A7C            00620          GOTO   got_char
                     00621
                     00622
                     00623 ; Otherwise, here a bit has just been read.  Check if the start
                     00624 ; sentinel code has ever been seen.  If it has then a sentinel
                     00625 ; bit will drop out after each character, which is dealt with by
                     00626 ; the code above.  If not then it is necessary to check for the
                     00627 ; start sentinel code after reading each bit.
                     00628
006D 060A            00629         IFSET   flag,found_start; Has the start code been seen?
006E 0A79            00630          GOTO   wt_clk_hi       ; Yes, so wait for ^CLK to go
                     00631                                 ; high again and get next bit.
                     00632
                     00633                                 ; No, so check for the start
                     00634                                 ; sentinel code now
                     00635
006F 0CF8            00636         MOVLW   B'11111000'     ; The start code is five bits
```

```
MPASM 01.50 Released         CARD509.ASM   9-29-1999  14:09:04         PAGE 13
Card Reader

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00637                                ; long, so mask off the low 3
0070 016B            00638         ANDWF   char_buf       ; bits in the buffer (which are
                     00639                                ; probably '0' anyway)
0071 0C58            00640         MOVLW   start_code     ; and compare start_code
0072 018B            00641         XORWF   char_buf,W     ; to the buffer
                     00642
0073 0743            00643         IFNZ                   ; Is it the start code?
0074 0A79            00644          GOTO   wt_clk_hi      ; No, so wait for ^CLK to go high
                     00645                                ; again and get the next bit.
                     00646
0075 050A            00647         BSF     flag,found_start; Yes, so set a flag
                     00648
                     00649
                     00650 ; Prepare for next character
                     00651
0076                 00652 next_char
0076 04EC            00653         BCF     parityLRC,MSB  ; clear the parity flag,
                     00654
0077 006B            00655         CLRF    char_buf       ; clear the input buffer,
                     00656
0078 058B            00657         BSF     char_buf,4     ; and set a sentinel bit
                     00658
                     00659
                     00660 ; Now wait for the next data bit
                     00661
0079                 00662 wt_clk_hi
0079 0746            00663         IFCLR   GPIO,clock     ; Check ^CLK line
007A 0A79            00664          GOTO   wt_clk_hi      ; Keep waiting whilst it's low
                     00665
007B 0A61            00666         GOTO    wt_clk_lo      ; Then go and wait for it to be
                     00667                                ; low again and get another bit
                     00668
                     00669
007C                 00670 got_char
                     00671
                     00672 ; Jump here when a sentinel bit has dropped out of the input
                     00673 ; buffer.  There is now a character in the top five bits of
                     00674 ; char_buf, the low three bits will be '0', and carry will be '1'
                     00675 ; The MSB of char_buf is the parity bit for the character, which
                     00676 ; can now be discarded, and bits [6..3] form the character itself:
                     00677 ;
                     00678 ; char_buf 7  6  5  4  3  2  1  0
                     00679 ;          P  D3 D2 D1 D0 0  0  0
                     00680 ;
                     00681 ; First, check the parity of the character just read.  The
                     00682 ; characters on the card are encoded with odd parity, and before
                     00683 ; each character is read the parity bit in parityLRC is cleared.
                     00684 ; This bit is toggled every time a '1' is read for the current
                     00685 ; character, which means that if the character was read correctly
                     00686 ; this bit will be '1'.
                     00687
                     00688
007C 07EC            00689         IFCLR   parityLRC,MSB  ; If parity bit is '0'
```



```
MPASM 01.50 Released         CARD509.ASM   9-29-1999  14:09:04         PAGE 14
Card Reader

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

007D 054A           00690          BSF    flag,bad_parity ; set the parity error flag
                    00691
                    00692
                    00693 ; Now XOR char_buf with the parityLRC register to update the LRC
                    00694 ; state.  The LRC portion of the parityLRC register was
                    00695 ; initialised with the code for the start sentinel.  Every time
                    00696 ; a character is read it is XORed with the parityLRC register.
                    00697 ; Any bits in the char_buf register which are set will toggle
                    00698 ; bits in the LRC portion of parityLRC.  If there have been no
                    00699 ; bit errors after reading all the characters and the LRC from
                    00700 ; the card, then the LRC bits in the parityLRC register will all
                    00701 ; be zero.
                    00702
007E 020B           00703          MOVF    char_buf,W      ; Copy char_buf to W
007F 01AC           00704          XORWF   parityLRC       ; XOR with the parityLRC register
                    00705                                  ; to update the LRC calculation
                    00706
                    00707                                  ; If the end sentinel has not
0080 072A           00708          IFCLR   flag,found_end  ; yet been seen then this is
0081 0A89           00709           GOTO   not_LRC         ; not the LRC, so store it
                    00710
0082 020C           00711          MOVF    parityLRC,W     ; Otherwise it was the LRC, so
0083 0E78           00712          ANDLW   b'01111000'     ; get the LRC check from the
                    00713                                  ; parityLRC register, and mask
                    00714                                  ; off the parity flag
                    00715
0084 0643           00716          IFZ                     ; If it is zero then the LRC was
0085 046A           00717           BCF    flag,bad_LRC    ; okay so clear the bad_LRC flag
                    00718
0086 0207           00719          MOVF    buf_ptr,W       ; Copy the value of buffer pointer
0087 0028           00720          MOVWF   num_chr         ; to num_chr
                    00721
0088 0A95           00722          GOTO    dump_buffer     ; and dump it out
                    00723
                    00724
0089                00725 not_LRC
0089 0CF8           00726          MOVLW   end_code        ; Is this the end sentinel?
008A 018B           00727          XORWF   char_buf,W
                    00728
008B 0643           00729          IFZ                     ; If so, the next character is
008C 052A           00730           BSF    flag,found_end  ; the LRC, so set a flag
                    00731
008D 0643           00732          IFZ
008E 0A76           00733           GOTO   next_char       ; and don't bother storing it
                    00734
008F 036B           00735          RLF     char_buf        ; discard parity by shifting it
                    00736                                  ; out, leaving the 4 bit
                    00737                                  ; character in the upper nibble
                    00738
0090 0CF0           00739          MOVLW   0xF0            ; mask off the lower nibble
0091 016B           00740          ANDWF   char_buf
                    00741
0092 0918           00742          CALL    get_put_char    ; and store the character
```

```
MPASM 01.50 Released         CARD509.ASM   9-29-1999  14:09:04         PAGE 15
Card Reader

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00743
0093 078A            00744         IFCLR   flag,buf_end    ; Is the buffer full?
0094 0A76            00745          GOTO   next_char       ; no, so get the next character
                     00746
                     00747                                 ; Otherwise, fall through...
                     00748
                     00749 ; Jump to dump_buffer when the buffer is full.  This routine
                     00750 ; loops through each "location" in the memory buffer, and sends
                     00751 ; the character at that location out serially on an output pin.
                     00752
0095                 00753 dump_buffer
0095 0067            00754         CLRF    buf_ptr         ; Load buffer pointer with 0
                     00755
0096 05AA            00756         BSF     flag,read_buf   ; Set the flag to read mode
0097 048A            00757         BCF     flag,buf_end    ; Clear the buf_end flag
                     00758
0098                 00759 loop_buffer
0098 0918            00760         CALL    get_put_char    ; Get character from buffer
                     00761
0099 0C30            00762         MOVLW   .48             ; convert to ASCII by adding 48
009A 01CB            00763         ADDWF   char_buf, W     ; and put the result in W
                     00764
009B 0902            00765         CALL    send_char       ; and send the character
                     00766
009C 078A            00767         IFCLR   flag,buf_end    ; have we emptied the buffer?
009D 0A98            00768          GOTO   loop_buffer     ; No, so loop around
                     00769
                     00770 ; After sending the contents of the buffer, two more characters
                     00771 ; are sent to indicate any errors.  If there was bad parity on
                     00772 ; any character a "P" is sent, and if there was a bad LRC an "L"
                     00773 ; is sent.  If either condition was okay we send a period "."
                     00774
009E 0C2E            00775         MOVLW   '.'             ; Load ASCII "." into W
                     00776
009F 064A            00777         IFSET   flag,bad_parity ; If parity was ever bad
00A0 0C50            00778          MOVLW  'P'             ; load ASCII "P" into W instead
                     00779
00A1 0902            00780         CALL    send_char       ; then send the character
                     00781
00A2 0C2E            00782         MOVLW   '.'             ; Load ASCII "." into W again
                     00783
00A3 066A            00784         IFSET   flag,bad_LRC    ; If LRC was bad
00A4 0C4C            00785          MOVLW  'L'             ; load ASCII 'L' into W instead
                     00786
00A5 0902            00787         CALL    send_char       ; and send the character
                     00788
00A6 0C0D            00789         MOVLW   .13             ; Send CR LF from serial port
00A7 0902            00790         CALL    send_char
                     00791
00A8 0C0A            00792         MOVLW   .10
00A9 0902            00793         CALL    send_char
                     00794
00AA 0A3F            00795         GOTO    main_loop       ; Back to the beginning and wait
```

```
MPASM 01.50 Released         CARD509.ASM   9-29-1999  14:09:04         PAGE 16
Card Reader

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00796                                  ; for another card.
                     00797
                     00798 ; The end is nigh...
                     00799
                     00800     END

SYMBOL TABLE
  LABEL                             VALUE

C                                 00000000
DC                                00000001
FSR                               00000004
GP0                               00000000
GP1                               00000001
GP2                               00000002
GP3                               00000003
GP4                               00000004
GP5                               00000005
GPIO                              00000006
GPWUF                             00000007
IFC                               SKPNC
IFCLR                             BTFSS
IFNC                              SKPC
IFNZ                              SKPZ
IFSET                             BTFSC
IFZ                               SKPNZ
INDF                              00000000
LSB                               00000000
MSB                               00000007
OSCCAL                            00000005
PA0                               00000005
PC                                00000002
PD                                00000003
RP0                               00000005
SKPCLR                            BTFSC
SKPSET                            BTFSS
STATUS                            00000003
TMR0                              00000001
TO                                00000004
Z                                 00000002
__12C509                          00000001
bad_LRC                           00000003
bad_parity                        00000002
bit_delay                         0000000F
bit_loop                          00000006
buf_end                           00000004
buf_ptr                           00000007
buf_sz                            00000028
card                              00000001
char_buf                          0000000B
chk_data                          00000063
clock                             00000002
clr_buf_loop                      00000041
clrloop                           00000039
count                             00000009
data_0                            00000069
data_1                            00000065
delay_loop                        00000014
dump_buffer                       00000095
end_code                          000000F8
flag                              0000000A
found_end                         00000001
```

```
MPASM 01.50 Released         CARD509.ASM   9-29-1999  14:09:04         PAGE 18
Card Reader

SYMBOL TABLE
  LABEL                             VALUE

found_start                         00000000
get_char                            00000026
get_put_char                        00000018
get_put_done                        0000002C
got_char                            0000007C
hi_mem                              00000033
invert_tx                           00000000
lo_mem                              00000010
loop_buffer                         00000098
main_loop                           0000003F
next_char                           00000076
not_LRC                             00000089
num_chr                             00000008
parityLRC                           0000000C
put_char                            00000020
read_buf                            00000005
send_char                           00000002
ser_out                             00000000
signal                              00000003
start                               00000031
start_code                          00000058
store_bit                           0000006A
temp                                0000000D
wait_card                           0000005F
wt_clk_hi                           00000079
wt_clk_lo                           00000061


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX----- ----------------

All other memory blocks unused.

Program Memory Words Used:   171
Program Memory Words Free:   853


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,    14 suppressed
```

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-786-7200 Fax: 480-786-7277
Technical Support: 480-786-7627
Web Address: http://www.microchip.com

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508-480-9990 Fax: 508-480-8575

Chicago
Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
Microchip Technology Inc.
4570 Westgrove Drive, Suite 160
Addison, TX 75248
Tel: 972-818-7423 Fax: 972-818-2924

Dayton
Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Detroit
Microchip Technology Inc.
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
Microchip Technology Inc.
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

AMERICAS (continued)

Toronto
Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

Hong Kong
Microchip Asia Pacific
Unit 2101, Tower 2
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

Beijing
Microchip Technology, Beijing
Unit 915, 6 Chaoyangmen Bei Dajie
Dong Erhuan Road, Dongcheng District
New China Hong Kong Manhattan Building
Beijing 100027 PRC
Tel: 86-10-85282100 Fax: 86-10-85282104

India
Microchip Technology Inc.
India Liaison Office
No. 6, Legacy, Convent Road
Bangalore 560 025, India
Tel: 91-80-229-0061 Fax: 91-80-229-0062

Japan
Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa 222-0033 Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Shanghai
Microchip Technology
RM 406 Shanghai Golden Bridge Bldg.
2077 Yan'an Road West, Hong Qiao District
Shanghai, PRC 200335
Tel: 86-21-6275-5700 Fax: 86 21-6275-5060

ASIA/PACIFIC (continued)

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan, R.O.C
Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5858 Fax: 44-118 921-5835

Denmark
Microchip Technology Denmark ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Arizona Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 München, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

11/15/99

DNV Certification, Inc. USA — ANSI-RAB QMS ACCREDITED — DNV — DNV MSC The Netherlands Accredited by the RvA
ISO 9001 / QS-9000 REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.