Column #148, March 2008 by Jon Williams:

It's In the Cards

It's seems like at least once a year – and usually around this time of year – I remind myself (and you) that it's okay to experiment, in fact; experimenting for the sake of experimenting is absolutely worth doing and we should all make time in our schedules for experimenting that doesn't require or expect any specific results. Why? Well, we all get caught up in our dramas and the things that need to happen right this second, and oftentimes we spend more energy than required because we're not ready. This is where experimenting for the sake of experimenting comes in. My friend, Cliff, is a teacher and as an avid sports enthusiast he frequently uses sports analogies with his students (including me). One of his favorite reminders, one that I think is wholly appropriate here, is that Michael Jordan never practiced the [specific] game (which is not possible as the one cannot know what the opponent will bring to it), he practiced his skills so that he was ready for the game – and whatever an opponent showed up with.

We should do that, too, and we practice "for the game" by experimenting. I thought about this when I was recently asked if it was possible to interface a mag-stripe card reader to the SX using SX/B. "Sure, no problem!" was my enthusiastic response. "Okay, how?" The deafening silence was interrupted with my sheepish answer, "You know… I don't know – yet – but I will find out!"

If you've read more than a few of my columns you know that I frequently refer to myself as a very lucky guy. I'm healthy, happy, live in a great country, get to write for this very cool magazine, and I live in Los Angeles – one of the greatest cities in the world. I'm not knocking any other city, I'm just saying that Los Angeles works for me. I'm near the beach, the mountains, the desert, Hollywood boulevard is just minutes away and when I'm doing a project or just want to experiment… a quick drive down the Hollywood freeway to Van Nuys gets me to All Electronics (www.allelectronics.com). Yes, electronics heaven, and it's open seven days a week! For those of you not in Los Angeles, don't worry, you can of course order from All via the Internet.

While on an LED shopping spree I went looking for a mag-stripe reader because All Electronics (like Tanners and BG Micro in Texas) carries a lot of "recycled" parts. Guess what? I found a card reader and it cost me all of three bucks. So now it's time to play. I don't need the card reader for any projects – for now – but I might later and if that comes up, or I get another request for code, I'll be ready, and there's a good chance I'll have learned something useful along the way.

Experimental Connections

The great thing about experimenting is that we don't have to worry about PCB layout and soldering unless we come up with something really cool and want to make it permanent. That said, we do have to connect things, so a little prep work is in order.

Way back in the beginnings of this column (#8), Scott Edwards taught us to build cables using female crimp pin connectors and 22-guage standard wire. Parallax hosts online reprints of the column so if you don't have that on your shelf you can find it here:

http://www.parallax.com/Portals/0/Downloads/docs/cols/nv/vol1/col/nv8.pdf

Building your own cables with these connectors is a worthwhile skill, especially for prototyping and experimenting. The only thing that I'll add to Scott's excellent instruction is that for absolutely bullet-proof connections you should use just a touch of solder on the joint. Those crimp pins were designed for machines that can exert far more pressure than we can with a hand tool, so soldering keeps the connector form breaking if the wire is tugged.

Now… you have to be very careful when doing this, as too much solder can cause the socket (on female connectors) to become clogged and not fit on to a pin header. The easiest way to prevent clogging is to solder these connectors the same way we would solder SMD components on a PCB: put a drop of liquid flux on the crimped joint, put a tiny bit of solder on the tip of your iron, and then touch the iron to the crimped connection. The flux will clean everything and the solder will wick into the connection an make it permanent. The reason for the liquid flux on the joint is that applying the solder to the iron will boil off any flux in the solder. Clean the connector with a bit of 99% alcohol or flux remover and then protect it with heat shrink tubing or box connector designed for the crimp sockets.

Read the Card

Figure 148.1 shows the setup on my desk for experimenting with the card reader. The reader has a 7-pin connector with male post headers, so I made jumper wires with a female connector on one end and a male pin on the other; the female end goes to the reader, the male end gets plugged into the SX-Tech board – a nice, low-cost setup for experimenting with the SX28.

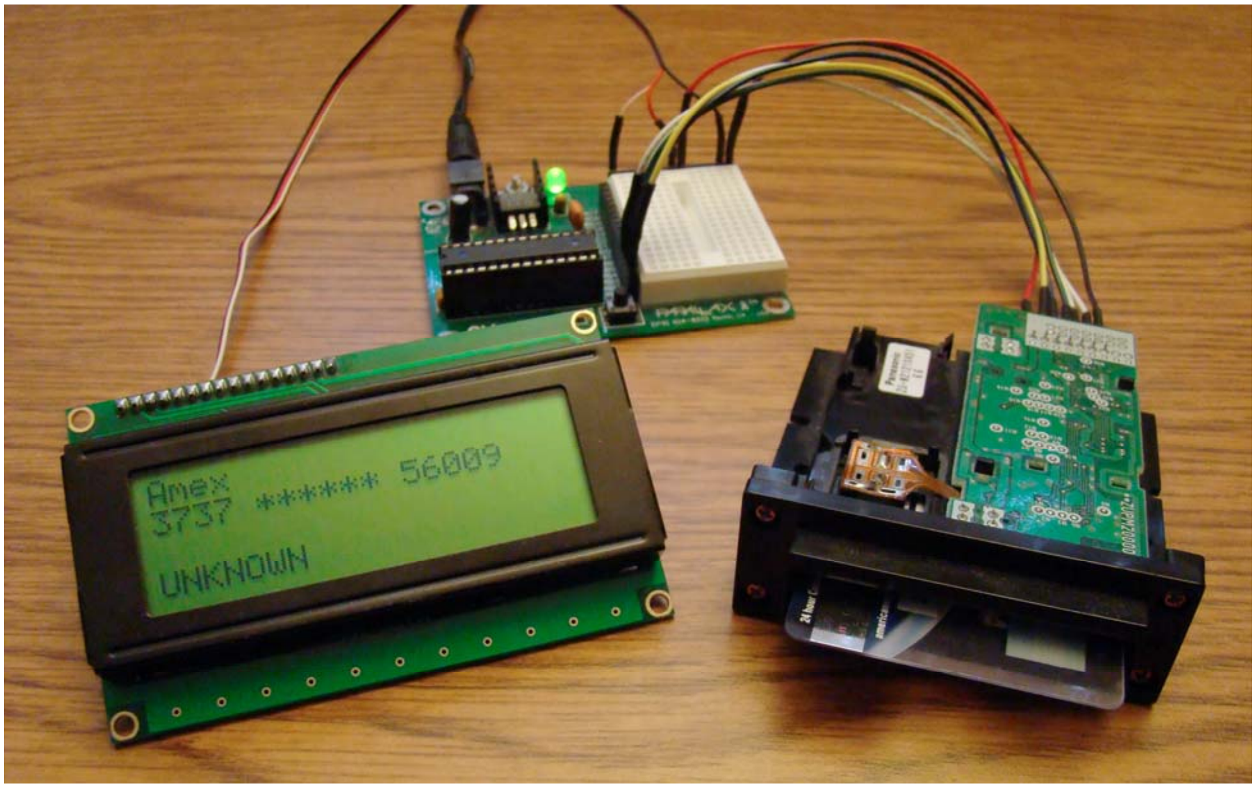

Figure 148.1: Card Reader Experiment Setup

For output I'm using a 4x20 serial LCD. Since there are no male post headers on the SX-Tech board I modified the LCD cable to give it male pins on one end; this lets me plug it into power and any I/O point on the SX that I desire.

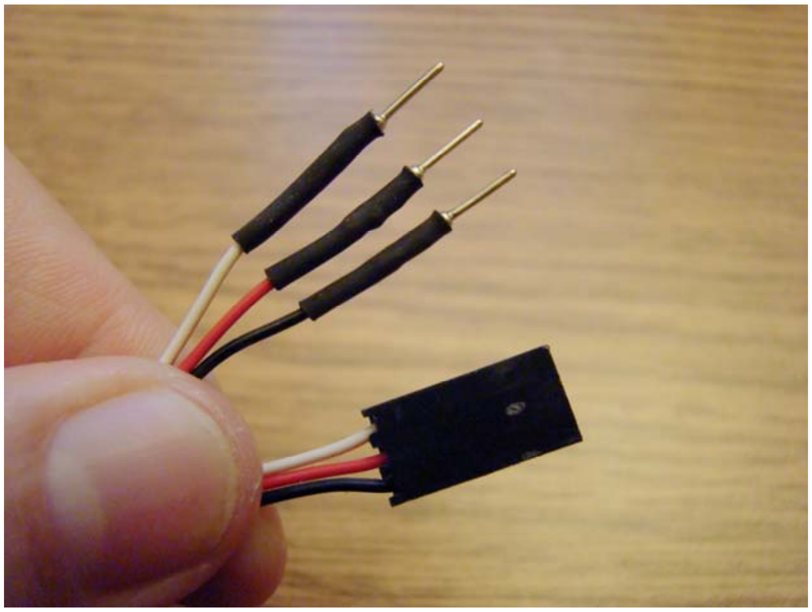

Figure 148.2: LCD/Servo Cable Modification

Figure 148.2 shows how I modified the standard LCD/Servo cable from Parallax to work with a solderless breadboard. Okay, we're ready to code.

Figure 148.3 shows the connections between the reader and the SX. What you're probably wondering is where the pull-ups are – as they are clearly not visible in Figure 1. For the experiment I'm using the SX's internal (weak) pull-ups. I think this is okay to do because I'm using such short connections. If we decide to install a card reader into a project where the connections are more that a foot long or so, we should use external pull-ups.

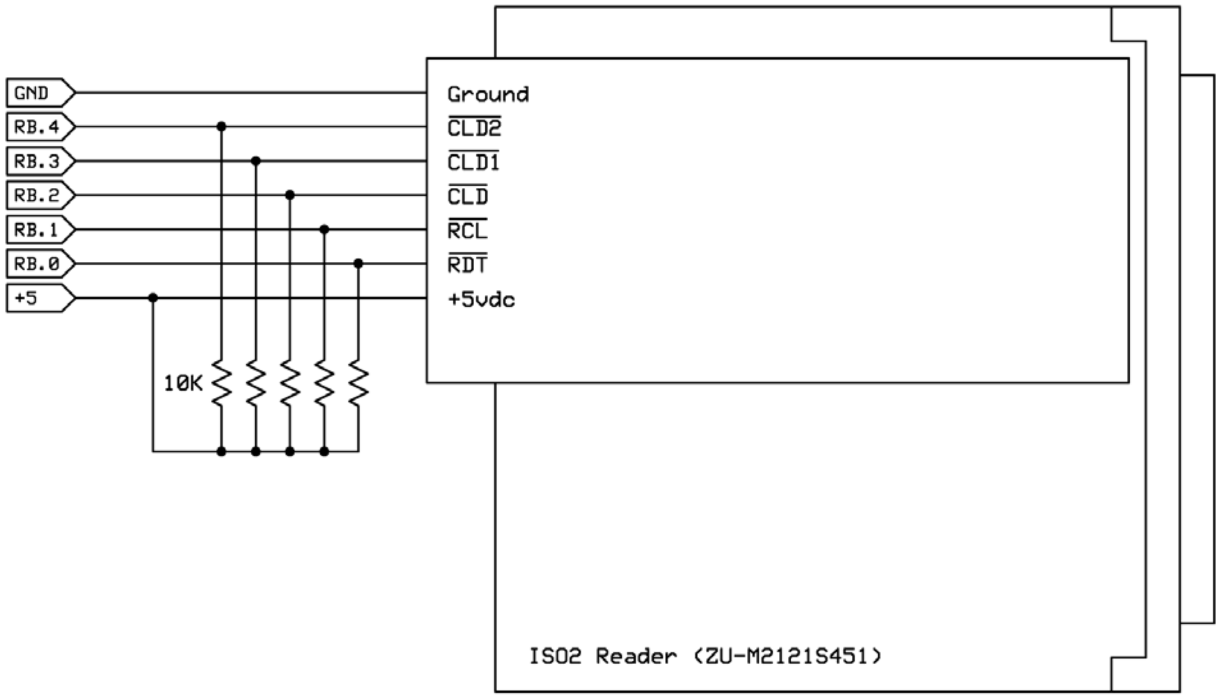

Figure 148.3: Card Reader/SX Microcontroller Connections

The SX/B compiler makes enabling the pull-ups on any give pin very simple – all we have to do is add the word **PULLUP** to the end of a **PIN** declaration

```
CrdData         PIN     RB.0 INPUT PULLUP
CrdClock        PIN     RB.1 INPUT PULLUP
CrdMotion       PIN     RB.2 INPUT PULLUP
CrdDetect       PIN     RB.3 INPUT PULLUP
CrdEndStop      PIN     RB.4 INPUT PULLUP
```

With the connections out of the way we can look at the signals from the reader (all are active-low):

/RDT	This is the data line; will be read when a clock pulse is detected
/RCL	Clock line (this comes from the reader)
/CLD	Card is in motion; goes high if card stops
/CLD1	Card has been inserted
/CLD2	Card has hit the end stop

With this particular reader the user is responsible inserting the card to the stop point; the data read takes place on the insertion (not on the retraction as I originally assumed). Because there is a physical stop for card we are limited in how much information can be pulled from track 2, but it is enough to read the full account number cards like Visa, MasterCard, etc. Track 2 contains 40 5-bit (four bits plus parity) characters, but as we're limited by the mechanical stop we'll keep the code simple by using a 16-byte array to capture the card number – this works fine for standard credit cards.
If we break down the process for reading card with this device it ought to go something like this:

1. Prompt the user
2. Wait for card insertion
3. Look for the start sentinel character
4. Read card characters from track until end stop
   a. monitor for slide error
5. Convert data to ASCII and display

Steps 1 and 2 are pretty easy; using a serial LCD is like using a general-purpose terminal – all we have to do is put the serial connection into an idle state and give the LCD a bit of time to handle internal initialization. Once that's done we can clear the LCD and then display an appropriate prompt; you know, something really snappy like, "Insert Card."

```
Start:
  TX = Idle
  DELAY_MS 100

Lcd_Setup:
  TX_BYTE LcdBLoff
  TX_BYTE LcdOn1

Main:
  CLR_LCD
  TX_STR "Insert Card"
  DO UNTIL CrdDetect = HasCard
  LOOP
```

Once a card has been detected (/CLD1 goes low) we can get to the meat of the matter – reading the card number. The interesting thing about this experiment is that the reader is a synchronous serial device (i.e., it has clock and data lines), but it provides the clock instead of accepting a clock – so this precludes the use of **SHIFTIN**. Yes, we're going to have to manually code the serial input from the reader but as you'll see, it's not difficult (and this code may be useful later).

If you connect a logic analyzer to the clock and data lines you'll see several clock pulses before any data bits show up. This is for good reason: it allows the external device to get in sync with the clock pulses and search for what is called the start sentinel. This is a special character that precedes the card number. Let’s have a look at the code that finds the start sentinel.

```
Find_Start_Sentinel:
  flags = %00000000
  char = 0
  DO
    DO WHILE CrdClock = 1
    LOOP
    char = char >> 1
    char.4 = ~CrdData
    DO WHILE CrdClock = 0
    LOOP
  LOOP UNTIL char = %01011
```

After clearing the error flags and buffer for the output the program enters a loop that waits for the leading edge of a clock pulse. Since the data bits are provided LSB first we prep the buffer value by shifting it to the right. Remember that a shift will cause the end bit to get a zero, so we don’t have to worry about stray bits polluting the output value. Then we sample the data line, moving the inverted value to bit 4 of the buffer byte. After the clock pulse clears we can check the value of char for %01011 which is the start sentinel that indicates we have alignment with the card data.

So now you can see why the clock pulses show up first; it allows a routine line this to lock onto the data stream for synchronization. Since the bits in char are shifted right every time through the loop, this is code acting like a sliding window on the data bits from the card. Once we detect the sentinel value we can read the card number which, like the sentinel, is made up of 5-bit values.

```
Read_Card_Number:
  FOR idx = 0 TO 15
    char = 0
    FOR bCount = 1 TO 5
      DO WHILE CrdClock = 1
        IF CrdMotion = IsStopped THEN
          slideErr = 1
          GOTO Process_Errors
        ENDIF
      LOOP
      char = char >> 1
      char.4 = ~CrdData
      DO WHILE CrdClock = 0
      LOOP
    NEXT
    buf(idx) = char
  NEXT
```

As you can see, the loop that reads the card data looks a lot like the code we used to find the sentinel; the difference, of course, is that we expect that we can read 16 packets of five bits each. When we have a character it is moved to an array called buf().

Note that in between clock pulses we do a quick check of the /CLD line which is low so long as the card is in motion. The reason for this check is to detect a partial insertion followed by a partial retraction; the card would have to stop to be retracted, and as such would throw off the synchronization of the data read. If a stop is detected we flag the error and abort the read loop.

Once we have the data captured we can verify it with a parity check. Since track 2 holds numbers and a few separator characters, four bits is all that are needed for the data, the fifth bit is used for [odd] parity.

```
Check_Parity:
  FOR idx = 0 TO 15
    char = buf(idx)
    bCount = 0
    FOR pCheck = 1 TO 4
      bCount = bCount + char.0
      char = char >> 1
    NEXT
    IF bCount.0 = char.0 THEN
      parityErr = 1
    ENDIF
  NEXT
```

The parity check loop pulls each byte from the buffer, counts the number of 1's in the lower four bits and compares the result to bit 5 – if there are an odd number of 1's in the data, bit 5 should be a 0; an even number of 1's in the data means we should find a 1 in bit 5. If a parity error is detected we'll set a flag and abort the loop.

Unless you hesitate when sliding the card, or have a defective card, you probably won't see the error processing code, but it's important to include it for robust applications:

```
Process_Errors:
  IF slideErr = 1 THEN
    CLR_LCD
    TX_STR "Slide Error"
    GOTO Remove_Card
  ENDIF
  IF parityErr = 1 THEN
    CLR_LCD
    TX_STR "Read Error"
    GOTO Remove_Card
  ENDIF
```

Nothing mysterious here; we simply prompt the user and have them try again.

Okay, now that we have the data and are sure it's good we can convert it to ASCII values to be displayed. This is really easy: we simply strip the parity bit and then add "0" (48 decimal) to convert to the appropriate ASCII codes.

```
Convert_To_Ascii:
  FOR idx = 0 TO 15
    char = buf(idx) & $0F
    buf(idx) = char + "0"
  NEXT
```

The final step is to display the card number – but let's make this a little interesting and show what kind of card was used. I found information on card type codes here:

http://money.howstuffworks.com/credit-card1.htm

To be candid, it's not perfect because my ATM card shows up as MasterCard, and my Borders book store card reads as a Discover card. The card number is still read correctly, it's just that the algorithm used to identify the card type is somewhat simplistic.

```
Get_Card_Type:
  CLR_LCD
  char = buf(0)
  IF char = "3" THEN
    char = buf(1)
    IF char = "7" THEN
      TX_STR "Amex"
      GOTO Display_Amex
    ELSEIF char = "8" THEN
      TX_STR "Diners"
    ELSE
      TX_STR "???"
    ENDIF
  ELSEIF char = "4" THEN
    TX_STR "Visa"
  ELSEIF char = "5" THEN
    TX_STR "M/C"
  ELSEIF char = "6" THEN
    TX_STR "Discover"
  ELSE
    TX_STR "???"
  ENDIF
```

The first character of the card defines its type, and in the case of a leading “3” the second character is checked for “7” (American Express) or “8” (Diners Club). When an Amex type card is detected the card number is formatted `XXXX XXXXXX XXXXX`, otherwise it will be formatted `XXXX XXXX XXXX XXXX`.

```
Display_Standard:
  TX_BYTE CR
  FOR idx = 0 TO 3
    TX_BYTE buf(idx)
  NEXT
  TX_BYTE " "
  FOR idx = 4 TO 7
  '{$IFDEF Secret_Display}
    TX_BYTE "*"
  '{$ELSE}
    TX_BYTE buf(idx)
  '{$ENDIF}
  NEXT
  TX_BYTE " "

  FOR idx = 8 TO 11
  '{$IFDEF Secret_Display}
    TX_BYTE "*"
  '{$ELSE}
    TX_BYTE buf(idx)
  '{$ENDIF}
  NEXT
  TX_BYTE " "
  FOR idx = 12 TO 15
    TX_BYTE buf(idx)
  NEXT
  TX_BYTE " "
  GOTO Check_Password


Display_Amex:
  TX_BYTE CR
  FOR idx = 0 TO 3
    TX_BYTE buf(idx)
  NEXT
  TX_BYTE " "
  FOR idx = 4 TO 9
  '{$IFDEF Secret_Display}
    TX_BYTE "*"
  '{$ELSE}
    TX_BYTE buf(idx)
  '{$ENDIF}
  NEXT
  TX_BYTE " "
  FOR idx = 10 TO 14
    TX_BYTE buf(idx)
  NEXT
```

These routines work fine with a 4x20 display; if you want to use a 16-column LCD then you’ll need to remove the spaces between groups (in fact, a single loop can be used to display the card number). I have included a conditional compilation section that allows the display to be modified so that the full card number is not displayed, somewhat like what is printed on charge card receipts.

We’ve done it – we took a three-dollar recycled ISO2 card reader and put it to use; and learned a couple neat things long the way. Before we wrap up, let’s create one more bit of code. Let’s say we want to compare the card number read against a known value – perhaps allowing us to use our card as an electronic ID (like my bank does when I want to do a transaction with a teller).

```
FUNC CHECK_CARD
  tmpW1 = __WPARAM12
  tmpB1 = __PARAM3
  tmpB2 = __PARAM4

  IF tmpB2 > 0 THEN
    goodCard = Yes
  ELSE
    goodCard = No
  ENDIF

  DO WHILE tmpB2 > 0
    READINC tmpW1, tmpB3
    tmpB4 = __RAM(tmpB1)
    IF tmpB4 <> tmpB3 THEN
      goodCard = No
      EXIT
    ENDIF
    INC tmpB1
    DEC tmpB2
  LOOP
  __PARAM1 = goodCard
  ENDFUNC
```

This function loops through the number of characters to check comparing a byte from the string with a byte from the card buffer. Since the card value is stored in an array we're using the __RAM() pointer to get to it – this will let us modify the code later to compare two cards that might be stored in different arrays.

Note the end of the code:

```
  __PARAM1 = goodCard
```

We can do this because the SX/B compiler uses the *__PARAMx* variables to move things back and forth. We could have done this:

```
  tmpB1 = goodCard
  RETURN tmpB1
```

If we look at the compiled code we'll find that *tmpB1* is moved to *__PARAM1* – so we can just skip that and make the code a little faster.

With this function we will pass a known good card number, either as an inline string or as a label that holds the card number as a z-string; we also need to send the pointer to the buffer that holds the card data and the number of charters we want to compare. We could, for example, use the function like this:

```
Check_Password:
  CHECK_CARD "0000000000000000", buf, 16
  TX_BYTE LcdLine4
  IF goodCard = Yes THEN
    TX_STR "JON WILLIAMS"
  ELSE
    TX_STR "UNKNOWN"
  ENDIF
```

Okay, that's it – go have some fun. It doesn't have to be with a card reader, it can be with anything. The point here is to try something new, and along the way learn something new that we can apply later. You know why Michael Jordan made so many clutch shots during his career? Because he practiced; he practiced a lot, and this made him one of the best players in the history of the game.

So go do it: become one of the best "players" in your game. Until next time – Happy Stamping.

Resources

http://money.howstuffworks.com/credit-card1.htm
http://www.cyberd.co.uk/support/technotes/isocards.htm

Parts List for Mag-Stripe Card Reader Experiment		
Part	**Part Number or Description**	**Source**
Card Reader	ZU-M2121S451	All Electronics MCR-12
Dev Board	SX-Tech Tool Kit	Parallax 45180
Power Supply	12v, 1A	Parallax 750-00007
Resonator	20 MHz	Parallax 250-02060
LCD	4x20 Serial	Parallax 27927
Cable	3-pin f/f	Parallax 805-00012
Crimp Pins	female, 0.025	Mouser 538-16-02-0102
Crimp Pins	male	Mouser 538-16-02-0114
Wire	#22, stranded	(various)

Code Listing

```
' =========================================================================
'
'   File...... Card_Reader.SXB
'   Purpose...
'   Author.... Jon Williams, EFX-TEK
'              Copyright (c) 2007 EFX-TEK
'              Some Rights Reserved
'              -- see http://creativecommons.org/licenses/by/3.0/
'   E-mail.... jwilliams@efx-tek.com
'   Started...
'   Updated...
'
' =========================================================================


' -------------------------------------------------------------------------
' Program Description
' -------------------------------------------------------------------------

' Reads first 16 characters after the start sentinel on track 2 of a
' mag stripe card -- ISO2 format (5-bit code [4 + parity])
'
' Helpful resources:
' -- http://money.howstuffworks.com/credit-card1.htm
' -- http://www.cyberd.co.uk/support/technotes/isocards.htm
'
' Code tested with #ZU-M2121S451 card reader from All Electronics
' -- www.allelectronics.com
' -- part #MCR-12


' -------------------------------------------------------------------------
' Conditional Compilation Symbols
' -------------------------------------------------------------------------

'{$DEFINE Secret_Display}                       ' hide some digits


' -------------------------------------------------------------------------
' Device Settings
' -------------------------------------------------------------------------

DEVICE          SX28, OSCXT2, TURBO, STACKX, OPTIONX, BOR42
FREQ            20_000_000
ID              "CardRead"
```

```
' -------------------------------------------------------------------------
' I/O Pins
' -------------------------------------------------------------------------

TX              PIN     RA.0 OUTPUT             ' to serial LCD

CrdData         PIN     RB.0 INPUT PULLUP       ' RDT\  (pin 2)
CrdClock        PIN     RB.1 INPUT PULLUP       ' RCL\  (pin 3)
CrdMotion       PIN     RB.2 INPUT PULLUP       ' CLD\  (pin 4)
CrdDetect       PIN     RB.3 INPUT PULLUP       ' CLD1\ (pin 5)
CrdEndStop      PIN     RB.4 INPUT PULLUP       ' CLD2\ (pin 6)

UnusedRA1       PIN     RA.1 INPUT PULLUP
UnusedRA2       PIN     RA.2 INPUT PULLUP
UnusedRA3       PIN     RA.3 INPUT PULLUP
UnusedRB5       PIN     RB.5 INPUT PULLUP
UnusedRB6       PIN     RB.6 INPUT PULLUP
UnusedRB7       PIN     RB.7 INPUT PULLUP
UnusedRCx       PIN     RC   INPUT PULLUP


' -------------------------------------------------------------------------
' Constants
' -------------------------------------------------------------------------

Yes             CON     1
No              CON     0

NoCard          CON     1
HasCard         CON     0

IsStopped       CON     1
IsMoving        CON     0

LF              CON     10                      ' line feed
FF              CON     12                      ' form feed; use as CLS
CR              CON     13                      ' carriage return

Baud            CON     "T19200"                ' or T2400, or T9600
Idle            CON     1                       ' for true mode

LcdBkSpc        CON     $08                     ' move cursor left
LcdRt           CON     $09                     ' move cursor right
LcdLF           CON     $0A                     ' move cursor down 1 line
LcdCls          CON     $0C                     ' clear LCD (need 5 ms delay)
LcdCR           CON     $0D                     ' move pos 0 of next line
LcdBLon         CON     $11                     ' backlight on
LcdBLoff        CON     $12                     ' backlight off
LcdOff          CON     $15                     ' LCD off
LcdOn1          CON     $16                     ' LCD on; no crsr, no blink
LcdOn2          CON     $17                     ' LCD on; no crsr, blink on
LcdOn3          CON     $18                     ' LCD on; crsr on, no blink
LcdOn4          CON     $19                     ' LCD on; crsr on, blink on
LcdLine1        CON     $80                     ' move to line 1, column 0
LcdLine2        CON     $94                     ' move to line 2, column 0
LcdLine3        CON     $A8                     ' move to line 3, column 0
LcdLine4        CON     $BC                     ' move to line 4, column 0

LcdCC0          CON     $F8                     ' define custom char 0
LcdCC1          CON     $F9                     ' define custom char 1
LcdCC2          CON     $FA                     ' define custom char 2
LcdCC3          CON     $FB                     ' define custom char 3
LcdCC4          CON     $FC                     ' define custom char 4
LcdCC5          CON     $FD                     ' define custom char 5
LcdCC6          CON     $FE                     ' define custom char 6
LcdCC7          CON     $FF                     ' define custom char 7


' -------------------------------------------------------------------------
' Variables
' -------------------------------------------------------------------------

flags           VAR     Byte
 slideErr       VAR     flags.0
 parityErr      VAR     flags.1
 mechStopErr    VAR     flags.2
 goodCard       VAR     flags.7                 ' for card comparison

idx             VAR     Byte                    ' loop index
char            VAR     Byte                    ' character from char
bCount          VAR     Byte                    ' counter for bits in byte
pCheck          VAR     Byte                    ' for parity checking

buf             VAR     Byte (16)               ' buffer for card #

tmpB1           VAR     Byte                    ' for subs/funcs
tmpB2           VAR     Byte
tmpB3           VAR     Byte
tmpB4           VAR     Byte
tmpW1           VAR     Word
tmpW2           VAR     Word


' =========================================================================
  PROGRAM Start
' =========================================================================


' -------------------------------------------------------------------------
' Subroutine / Function Declarations
' -------------------------------------------------------------------------

DELAY_MS        SUB     1, 2                    ' wrapper for PAUSE

CLR_LCD         SUB     0                       ' clears LCD screen
TX_BYTE         SUB     1                       ' wrapper for SEROUT
TX_STR          SUB     2                       ' transmit a string

CHECK_CARD      FUNC    1, 4                    ' check card string


' -------------------------------------------------------------------------
' Program Code
' -------------------------------------------------------------------------

Start:
  TX = Idle
  DELAY_MS 100                                  ' let LCD self initialize


Lcd_Setup:
  TX_BYTE LcdBLoff                              ' backlight off
  TX_BYTE LcdOn1                                ' no cursor or blink


Main:
  CLR_LCD
  TX_STR "Insert Card"
  DO UNTIL CrdDetect = HasCard                  ' wait for card insertion
  LOOP


Find_Start_Sentinel:
  flags = %00000000                             ' assume no errors
  char = 0                                      ' clear character buffer
  DO
    DO WHILE CrdClock = 1                       ' wait for 1 -> 0 transition
    LOOP
    char = char >> 1                            ' prep for next bit
    char.4 = ~CrdData                           ' get bit, LSB first
    DO WHILE CrdClock = 0                       ' wait for end of clock bit
    LOOP
  LOOP UNTIL char = %01011                      ' hold for start sentinel


Read_Card_Number:
  FOR idx = 0 TO 15                             ' read card # (with sep for Amex)
    char = 0
    FOR bCount = 1 TO 5                         ' five bits per char (4 + parity)
      DO WHILE CrdClock = 1                     ' wait for bit
        IF CrdMotion = IsStopped THEN
          slideErr = 1
          GOTO Process_Errors
        ENDIF
      LOOP
      char = char >> 1
      char.4 = ~CrdData
      DO WHILE CrdClock = 0
        IF CrdMotion = IsStopped THEN
          slideErr = 1                         ' set flag on error
          GOTO Process_Errors
        ENDIF
      LOOP
    NEXT
    buf(idx) = char
  NEXT


Check_Parity:
  FOR idx = 0 TO 15                             ' loop through buffer
    char = buf(idx)                             ' pull a character
    bCount = 0                                  ' clear counter (for 1's)
    FOR pCheck = 1 TO 4                         ' count four bits
      bCount = bCount + char.0                  ' update 1's count
      char = char >> 1                          ' position next bit
    NEXT
    IF bCount.0 = char.0 THEN                   ' compare with parity bit
      parityErr = 1                             ' set flag on error
      GOTO Process_Errors
    ENDIF
  NEXT


Process_Errors:
  IF slideErr = 1 THEN
    CLR_LCD
    TX_STR "Slide Error"
    GOTO Remove_Card
  ENDIF
  IF parityErr = 1 THEN
    CLR_LCD
    TX_STR "Read Error"
    GOTO Remove_Card
  ENDIF


Convert_To_Ascii:
  FOR idx = 0 TO 15                             ' loop through buffer
    char = buf(idx) & $0F                       ' strip parity bit
    buf(idx) = char + "0"                       ' convert to ASCII
  NEXT


Get_Card_Type:
  CLR_LCD
  char = buf(0)                                 ' get card type char
  IF char = "3" THEN                            ' Amex or Diners
    char = buf(1)
    IF char = "7" THEN                          ' get card sub-type
      TX_STR "Amex"
      GOTO Display_Amex
    ELSEIF char = "8" THEN
```

```
      TX_STR "Diners"
    ELSE
      TX_STR "???"
    ENDIF
  ELSEIF char = "4" THEN
    TX_STR "Visa"
  ELSEIF char = "5" THEN
    TX_STR "M/C"
  ELSEIF char = "6" THEN
    TX_STR "Discover"
  ELSE
    TX_STR "???"
  ENDIF


Display_Standard:
  TX_BYTE CR
  FOR idx = 0 TO 3
    TX_BYTE buf(idx)
  NEXT
  TX_BYTE " "
  FOR idx = 4 TO 7
  '{$IFDEF Secret_Display}
    TX_BYTE "*"
  '{$ELSE}
    TX_BYTE buf(idx)
  '{$ENDIF}
  NEXT
  TX_BYTE " "
  FOR idx = 8 TO 11
  '{$IFDEF Secret_Display}
    TX_BYTE "*"
  '{$ELSE}
    TX_BYTE buf(idx)
  '{$ENDIF}
  NEXT
  TX_BYTE " "
  FOR idx = 12 TO 15
    TX_BYTE buf(idx)
  NEXT
  TX_BYTE " "
  GOTO Check_Password


Display_Amex:
  TX_BYTE CR
  FOR idx = 0 TO 3
    TX_BYTE buf(idx)
  NEXT
  TX_BYTE " "
  FOR idx = 4 TO 9
  '{$IFDEF Secret_Display}
    TX_BYTE "*"
  '{$ELSE}
    TX_BYTE buf(idx)
  '{$ENDIF}
  NEXT
  TX_BYTE " "
  FOR idx = 10 TO 14
    TX_BYTE buf(idx)
  NEXT


Check_Password:
  CHECK_CARD Passcode1, buf, 16                 ' check for me
  TX_BYTE LcdLine4
  IF goodCard = Yes THEN
    TX_STR "JON WILLIAMS"
  ELSE
    TX_STR "UNKNOWN"
  ENDIF
```

```
Remove_Card:
  DO WHILE CrdDetect = HasCard                  ' wait for card removal
  LOOP
  GOTO Main


' -------------------------------------------------------------------------
' Subroutine / Function Code
' -------------------------------------------------------------------------

' Use: DELAY_MS duration
' -- "duration" in milliseconds
' -- wrapper for PAUSE

SUB DELAY_MS
  IF __PARAMCNT = 1 THEN
    tmpW1 = __PARAM1
  ELSE
    tmpW1 = __WPARAM12
  ENDIF
  PAUSE tmpW1
  ENDSUB

' -------------------------------------------------------------------------

' Use: CLR_LCD
' -- clears LCD with proper delay before next possible transmission

SUB CLR_LCD
  TX_BYTE LcdCls
  DELAY_MS 2
  ENDSUB

' -------------------------------------------------------------------------

' Use: TX_BYTE byteVal
' -- transmit "byteVal" at "Baud" on pin "TX"

SUB TX_BYTE
  SEROUT TX, Baud, __PARAM1
  ENDSUB

' -------------------------------------------------------------------------

' Use: TX_STR [string | label]
' -- "string" is an embedded string constant
' -- "label" is DATA statement label for stored z-String

SUB TX_STR
  tmpW1 = __WPARAM12                            ' get address of string
  DO
    READINC tmpW1, tmpB1                        ' read a character
    IF tmpB1 = 0 THEN EXIT                      ' if 0, string complete
    TX_BYTE tmpB1                               ' send character
  LOOP
  ENDSUB

' -------------------------------------------------------------------------

' Use: result = CHECK_CARD [String | Label], pntr, length
' -- compares card array to inline string or z-string at Label
' -- "pntr" is a RAM address of card data
' -- "length" is the number of characters to compare

FUNC CHECK_CARD
  tmpW1 = __WPARAM12                            ' pointer to test string
  tmpB1 = __PARAM3                              ' RAM pointer to card data
  tmpB2 = __PARAM4                              ' length of comparison
  IF tmpB2 > 0 THEN
    goodCard = Yes                             ' assume good
  ELSE
    goodCard = No                              ' abort on zero length
  ENDIF

  DO WHILE tmpB2 > 0
    READINC tmpW1, tmpB3                       ' get char from string
    tmpB4 = __RAM(tmpB1)                       ' get comparison character
    IF tmpB4 <> tmpB3 THEN                     ' if mismatch
      goodCard = No                            '   flag bad
      EXIT                                     '   and abort
    ENDIF
    INC tmpB1
    DEC tmpB2
  LOOP
  __PARAM1 = goodCard                          ' 1 = good, 0 = bad
  ENDFUNC


' -------------------------------------------------------------------------
' User Data
' -------------------------------------------------------------------------

Passcode1:
  DATA  "1111222233334444", 0                  ' replace with valid card#
```