Column #70, February 2001 by Jon Williams:

Let There Be LEDs

I really enjoy Dallas. The city is clean, the people are friendly and aside from some occasionally wacky weather (it snowed last weekend; today it's in the 60 and I'm as sick as a dog as I write this…), it's just a great city to live in. And at the risk of repeating myself, one of the best things about Dallas – in my opinion – is Tanner Electronics. Tanner's, a family-owned and operated surplus store, is to the electronics enthusiast what Willie Wonka's factory is to a chocolate addict. Jim Tanner and his family always have a smile and time for a customer.

While roaming the aisles on a recent parts run to Tanner's, I came across a two-inch tall, 5x7 common-cathode LED matrix (the kind used in big LED signs). I didn't have any particular use of it at the time, but it looked interesting and for two bucks, I thought it would be worth playing with. I was right.

I like this particular matrix because it's a set of raw LEDs and, other than its row/column wiring, there is no internal logic. What this means is that with a little bit of programming, I can display whatever I want: characters, graphics, anything. If you're only interested in displaying numbers and letters, there are arrays available with built-in ASCII decoders.

To control 35 LEDs individually would take a lot of I/O, not to mention power-supply current. And my LED is pre-wired as a 5x7 array, so it expects to be multiplexed. For review, multiplexing is the process of illuminating a portion of a display (a single digit or column, for example) for a short time, then another portion, then another and so on until we get back to the first element. This process takes place rapidly so our eye doesn't perceive any display flicker. The biggest advantage to multiplexing is that it cuts down the number of I/O lines required to drive a given display. In our project, we will be able to control 35 LEDs with just 12 lines.

Multiplexing can be accomplished with the Stamp, and yet, the process consumes a lot of horsepower and leaves us little time to do anything else. So what do we do? We pick up Maxim's MAX7219 LED display driver, that's what we do.

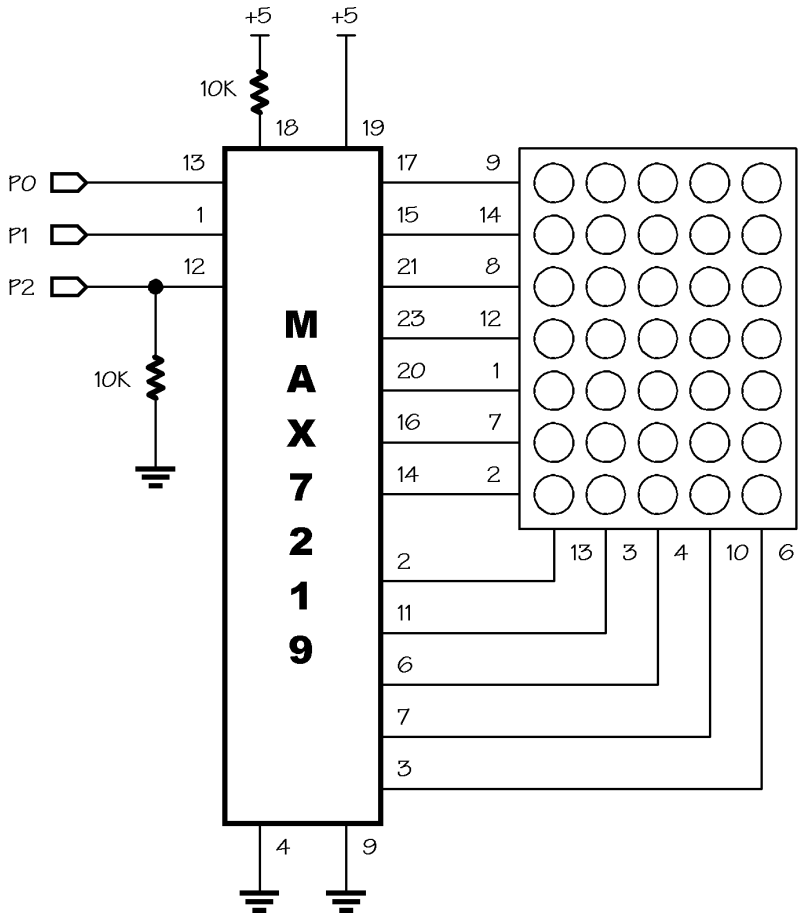
Scott Edwards introduced the MAX7219 to Stamp users way back in December of 1995. In that project, Scott used the MAX7219 in its traditional role: to control several seven-segment (numeric) LED displays. This is easy to do with the MAX7219 because it contains decimal (and hex) decoding logic to properly display digits on seven-segment LEDs. What we'll do this month is disable the decimal (BCD) decoding and take full manual control of the chip. By doing this, we have complete control of up to 64 individual LEDs – all with just 16 lines from the MAX7219. From the Stamp's point-of-view, we only need three lines for the display since the MAX7219 communicates with the MAX7219 with SHIFTOUT over a standard three-wire interface.

Okay, you've got the idea so let's just jump right into it. There's only one program this month and it's packed. Of course, you're free to selectively remove portions of the main code block that you don't care about. If you don't have a standard PC joystick, you'll need to modify the code so that it doesn't get stuck in the "Crosshair" section. Ready? Let's go.

Project Hardware

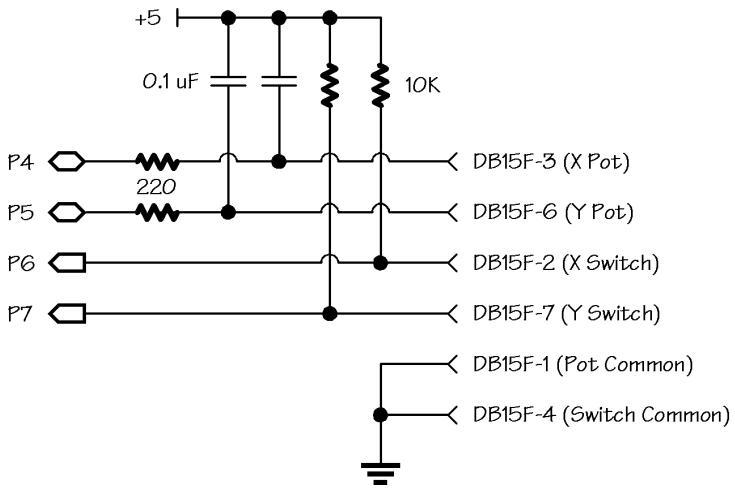
Yeah, there are a few connections, but the hardware this month is actually pretty simple (see Figure 70.1). The trickiest part is keeping track of each wire and where it's going. Since this is an experiment, I assembled mine on the new Parallax INEX-1000 board. It's a nice product in that it has a full-sized solder less breadboard, a beefy five-volt supply and lots of little extras. To be honest, I did have to do some soldering to make an adaptor for the LED array, but that was a pretty simple process and only took 15 minutes or so.

Figure 70.1: Maxim 7219 hookup with dot-matrix LED



The Stamp sends data to the MAX7219 through a standard three-wire, synchronous serial interface. After sending 16 bits (register and data) to the MAX7219, it's latched into the device by blipping (low-high-low) the load line. You may be wondering why the load line is pulled down to ground through a 10K resistor. Simply put, this resistor holds the load line low while the Stamp is resetting, preventing spurious (bad) data from getting into the MAX7219.

Figure 70.2: Standard PC joystick hookup



Another surplus part that I had on my bench and wanted to play with was a standard PC joystick. Electrically, a joystick is a pretty simple arrangement of two poteniometers and two normally-open switches. The pots have a common line, as do the switches. This presents no problem for the Stamp. We can use RCTIME to measure the pots and with pull-ups, we can easily monitor the state of the buttons. Figure 70.2 is the schematic for a joystick interface to the Stamp.

The Code, The Code

Okay, the hardware is all wired-up, so let's write some code. There's a lot neat stuff in the EEPROM Data section, but skip over that for the moment and look at the initialization code. The MAX7219 is a register-oriented device. To put new data in a register, we'll use SHIFTOUT to send the register number, then the data. After 16 bits have been shifted, the new data is latched into the MAX7219 by using PULSOUT to blip the Load line.

When it's first powered-up, the MAX7219 display is cleared, the column scan limit is set to one, the brightness to minimum and BCD decoding off. For our program, we need to set the scan limit to five (number of columns in the LED matrix), we'll turn up the brightness just a bit and we'll make sure the display is turned on.

There's a bit of an inconsistency in the MAX7219. To set the scan limit to five columns, we'll put a four in the scan limit register ($0B). After that though, columns are addressed as one through five. We'll need to remember that when writing our code.

We send the initialization data to the MAX7219 by using an LOOKUP table that's embedded in a loop. On every other pass through the loop we'll blip the Load line to latch the register address and data. We can tell when it's time to blip the Load line by looking at Bit0 of the loop counter. When this bit is set, we've shifted 16 bits and it's time for Load.

Now that the display is initialized, we can start sending data. To display a character or graphic, we have to send data (one byte) to each of the five columns. Now go back and look at the EEPROM Data section of the listing. In this section you'll see that I've defined character maps for space, "B", "S" and "2." Notice that the definitions seem to have been turned on their sides. It's easier to read the map if you rotate the magazine or your computer monitor 90 degrees counter-clockwise. Okay, I'm kidding…don't tilt your monitor; just follow along.

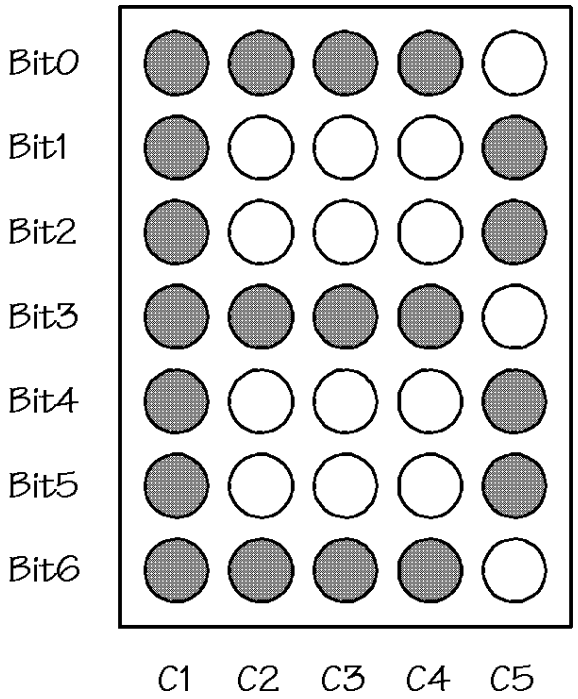
These definitions may seem odd, especially if you've ever designed custom characters for an LCD module. The reason for the peculiar definition is the nature of the MAX7219's column scan. It works like this: Column one (left-most) is enabled (made low) and the row data is output on the segment lines. It's held briefly. Column one is turned off and the process is repeated for column two. And so on to the scan limit. When the scan limit is reached, the process starts again at the first column.

Our job then, is to define the sequential column data for each character. Figure 70.3 shows the letter "B" LEDs lit in the array. A lit LED is equal to a one in the data byte. We just need keep in mind that the top row of the array corresponds to Bit0 of the data byte.

Moving on, the first part of the demo is called Flash_Characters. It does just that: flashes the characters "B", "S" and "2". It works by reading the definition address from a LOOKUP table. Once the character definition address is known, ShowChar displays the character. This subroutine takes the character address, reads the definition data from EEPROM (five bytes for each character) and sends it to the MAX7219.

A neat side effect of the having our definitions stored sequentially is that we can loop through the addresses and call ShowChar to create a crawling (horizontal) window. The MAX7219 is simply presenting the data we send it. If our address is offset from the beginning of a character definition, the character will be offset in the display.

Figure 70.3: Lighting the letter "B"



With our definitions adjacent to each other in memory, the display will behave accordingly. This is the reason for the zero between each set of character definition bytes. It creates a blank column when the display is crawling.

That explains the behavior of Crawl_Characters. It simply loops through the definition bytes and passes the address to ShowChar for display. You can change the crawl direction by reversing the direction of the loop. If you do this, though, the "2" will appear first. Finally, the crawl speed is controlled by PAUSE 250. You can speed it up by making the PAUSE value smaller.

Scrolling characters into the display vertically is a little bit trickier, but thanks to the Stamp's data types and ability to overlay variables, it can be done with just a bit of code. It helps here to visualize two characters stacked on top of each other, and then extend the bits in our definition graphic to Bit15 (the size of a Word). By loading character data in the low and high bytes of vScroll, we can scroll characters into the display, bottom to top, by using the right shift (>>) operator.

Take a look at the code in Scroll_Characters. A loop is used to scroll through four characters (B, S, 2 and space). LOOKUP tables are used to define the starting addresses of the scrolling characters (notice the offset in the tables). An inner loop controllers the scrolling by shifting our buffer, vScroll, to the right eight times. For each iteration through the scroll loop, we grab the column data for each of the two characters from EEPROM, shift it to the proper "window" and send it to the MAX7219.

The next part of the demo, Show_Graph, draws a simple bar graph on the LED matrix using values stored in a DATA table. The code for drawing a bar is similar to the scrolling characters code. In this case, we'll scroll a bunch of ones (lit LEDs) into a blank column. The number of ones scrolled in is based on the value for the column. Here's the code:

```
vScroll = $FF80 >> (grVal / GraphY)
```

The value, $FF80 stuffs a bunch of ones into upper end of vScroll. The shift operator moves the bits. The larger the data value (in grVal), the greater the number of shifts, resulting in a "taller" bar in the graph. The value, GraphY, is a pre-defined constant that determines the value of each lit LED. You can change the value of GraphMax (GraphY is GraphMax divided by seven) in the constants section to accommodate larger or smaller data values. And, if you put the graph values in an array, you could create a scrolling display.

The final section of the program, Crosshair, reads a PC joystick and creates a display on the fly. When the joystick is centered, a crosshair is drawn on the LED matrix. If you move the joystick left or right, the vertical line moves left or right accordingly. If you move the joystick up or down, the horizontal line follows.

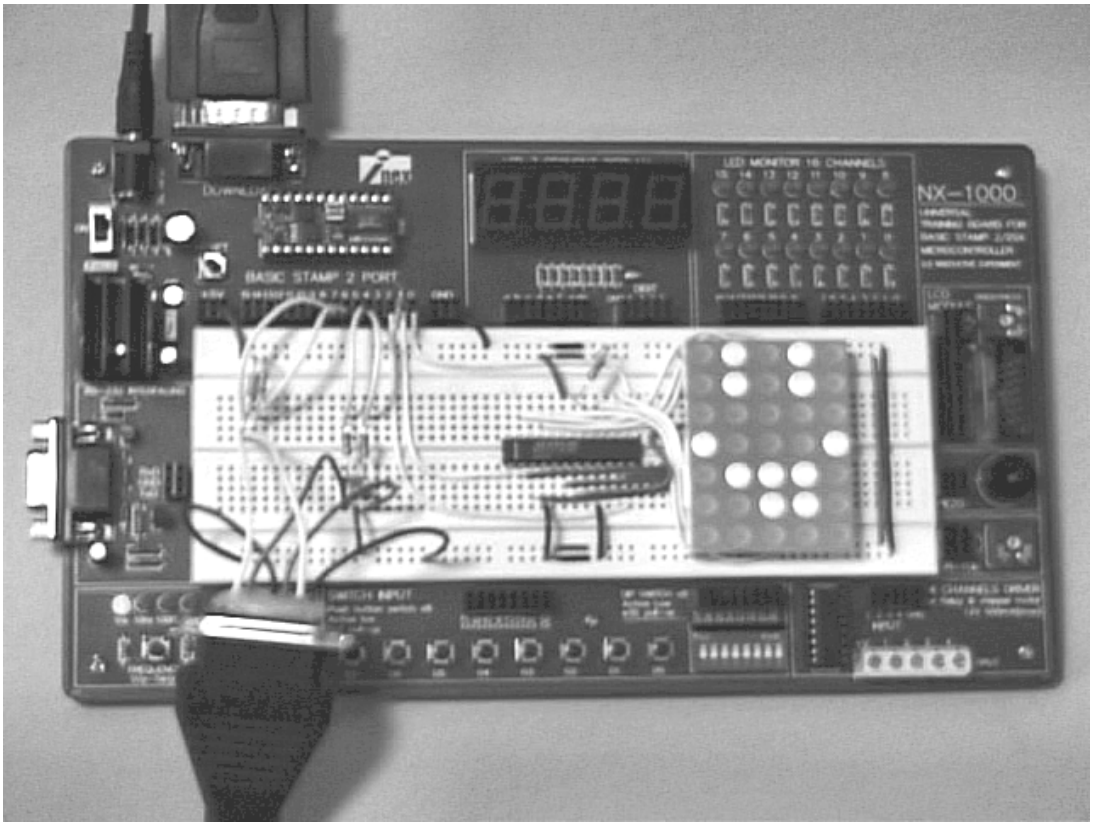
Pressing the X-axis button causes a smirky smiley face to be displayed. Pressing the Y-axis button starts the program all over. Okay, let's look at the crosshair display.

The first thing we do is grab the buttons. Since the two inputs are pulled-up, they'll read as one when open and zero when closed. The buttons are connected to the upper two inputs of the input nibble, so the data is shifted right twice and inverted to give it positive logic. The upper bits of the shifted data are masked out so our final value has a range of zero to three.

The buttons value is used in a BRANCH table to control the flow of this part of the program. If the X button is pressed, the value will be one and the code will jump to the

label called ShowSmile. The code here is very straightforward. It simply points at a custom character definition and displays it (you can see the smiley in Figure 70.4). After a short delay, the program loops back to the button check.

Figure 70.4: Smiley display



If no buttons are pressed, the code drops through to the crosshair display at ShowTarget. This code uses the subroutine, ReadJoyStick, to grab the pot values from the joystick using RCTIME. If you decide to port the code to one of the faster Stamps (BS2sx or BS2p), you'll need double-check the JoyXMax and JoyYMax values. You should probably do this anyway since my joystick is very old and may use non-standard values.

Okay, we know where the stick is pointing by reading the two pots, now we need to convert this into row and column data for the crosshair display. This is easy, we simply need to divide the current reading by the respective value for each row or column

With the y-axis value, we'll use a LOOKUP table to get the column data – the lit dot corresponding to the up-down position of the joystick. By using the same value all the way across the display, a horizontal line is drawn. When we get to the column that corresponds to the x-axis value, we fill it with ones, creating a vertical line. It's deceptively easy.

You may be wondering about the LED matrix being connected directly to the MAX7219 without any current limiters (resistors) in between. The current to the LEDs is actually controlled by the MAX7219 and is set by the 10K resistor on Pin18. At 10K, the current to each segment is about 40 mA. With seven LEDs in a column lit, this ads up to about 280 mA (total current draw is also affected by the Intesity register value). This is why we need a separate power supply for the MAX7219 – there's no possible way for the Stamp regulator to supply this kind of current.

Taking It Further

If you're using a BS2sx, BS2e or BS2p you can take advantage of the extra EEPROM space and define an entire custom character set. Using simple math, it would be very easy to convert the characters in a text string to EEPROM addresses for your characters. With a little effort, you could extend this month's demo into a single-character, scrolling message board. Are you up to it? Of course you are – and have fun doing it.

Until next time, Happy Stamping.

```
' Program Listing 70.1
' Nuts & Volts - February 2000

' ----[ Title ]---------------------------------------------------------
'
' File...... LEDARRAY.BS2
' Purpose... Uses the MAX7219 to drive a 5x7 LED array
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' Started... 06 JAN 2001
' Updated... 07 JAN 2001

' {$STAMP BS2}


' ----[ Program Description ]-------------------------------------------
'
' Demonstrates the use of Maxim's MAX7219 LED display driver in the
' non-decoded mode.  In this mode, the programmer is responsible for
' sending segment (row) data for each digit (column).
'
' In this application, the MAX7219 is connected to a common-cathode LED
' array.  The array is five columns wide by 7 rows tall (35 LEDs).  The
' digit outputs from the MAX7219 are connect to the columns; the segment
' control lines to the rows.
'
' MAX7219 --> LED Connections:
'
'   MAX7219.2  (0) --> Col 1 (left)
'   MAX7219.11 (1) --> Col 2
'   MAX7219.6  (2) --> Col 3
'   MAX7219.7  (3) --> Col 4
'   MAX7219.3  (4) --> Col 5
'
'   MAX7219.17 (g) --> Row 1 (top)
'   MAX7219.15 (f) --> Row 2
'   MAX7219.21 (e) --> Row 3
'   MAX7219.23 (d) --> Row 4
'   MAX7219.20 (c) --> Row 5
'   MAX7219.16 (b) --> Row 6
'   MAX7219.14 (a) --> Row 7


' ----[ Revision History ]----------------------------------------------
'
' 07 JAN 2001 - Version 1


' ----[ I/O Definitions ]------------------------------------------------
'
```

```
Clock           CON     0                       ' shift clock to MAX7219
DPin            CON     1                       ' shift data to MAX7219
Load            CON     2                       '

JoyX            CON     4                       ' x-axis pot reading
JoyY            CON     5                       ' y-axis pot reading
BtnX            VAR     In6                     ' x-axis button
BtnY            VAR     In7                     ' y-axis button
JoyBtns         VAR     InB


' ----[ Constants ]-----------------------------------------------------
'
Decode          CON     $09                     ' bcd decode
Intensity       CON     $0A                     ' brightness
Scan            CON     $0B                     ' scan (column) limit
ShutDn          CON     $0C                     ' shutdown (1 = on)
Test            CON     $0F                     ' display test mode

Yes             CON     1
No              CON     0

GraphMax        CON     255                     ' scale value for graph mode
GraphY          CON     GraphMax / 7            ' division value

JoyXMax         CON     6200                    ' pre-measure max value
JoyYMax         CON     5730                    ' pre-measured max value


' ----[ Variables ]-----------------------------------------------------
'
index           VAR     Nib                     ' loop counter
idxOdd          VAR     index.Bit0              ' is index odd? (1 = yes)
d7219           VAR     Byte                    ' data for MAX7219
char            VAR     Byte                    ' character ee address
col             VAR     Nib                     ' column value
row             VAR     Nib                     ' row value
eeAddr1         VAR     Byte                    ' ee pointer
eeAddr2         VAR     Byte
vScroll         VAR     Word                    ' scrolling data buffer
grVal           VAR     Byte                    ' graph value

joyXval         VAR     Word                    ' joystick x reading
joyYval         VAR     Word                    ' joystick y reading
xAxis           VAR     Nib                     ' target axis
yAxis           VAR     Nib
btns            VAR     Nib                     ' button status


' ----[ EEPROM Data ]---------------------------------------------------
'
```

```
Char_Space      DATA    %0000000
                DATA    %0000000
                DATA    %0000000
                DATA    %0000000
                DATA    %0000000
                DATA    0                       ' column between characters

Char_B          DATA    %1111111                ' xxxxxxx
                DATA    %1001001                ' x..x..x
                DATA    %1001001                ' x..x..x
                DATA    %1001001                ' x..x..x
                DATA    %0110110                ' .xx.xx.
                DATA    0

Char_S          DATA    %0100110                ' .x..xx.
                DATA    %1001001                ' x..x..x
                DATA    %1001001                ' x..x..x
                DATA    %1001001                ' x..x..x
                DATA    %0110010                ' .xx..x.
                DATA    0

Char_2          DATA    %1000010                ' x....x.
                DATA    %1100001                ' xx....x
                DATA    %1010001                ' x.x...x
                DATA    %1001001                ' x..x..x
                DATA    %1000110                ' x...xx.

Pad             DATA    0,0,0,0,0
                DATA    0

GrData          DATA    130,220,255,150,75      ' graph values

Smile           DATA    %0001000                ' ...x...
                DATA    %0010011                ' ..x..xx
                DATA    %0110000                ' .xx....
                DATA    %0110011                ' .xx..xx
                DATA    %0001000                ' ...x...


' ----[ Initialization ]-----------------------------------------------
'
Initialize:
  DirL = %111                                   ' clock, data and load pins

  FOR index = 0 TO 5
    LOOKUP index,[Scan,4,Intensity,7,ShutDn,1],d7219
    SHIFTOUT Dpin,Clock,MSBFirst,[d7219]
    IF idxOdd = No THEN NoLoad
    PULSOUT Load,3                              ' load parameter
  NoLoad:
  NEXT
```

```
' ----[ Main Code ]----------------------------------------------------------
'
Main:

Flash_Characters:                           ' on screen, one at a time
  FOR char = 0 TO 3
    LOOKUP char,[Char_B,Char_S,Char_2,Char_Space],eeAddr1
    GOSUB ShowChar
    PAUSE 1000
  NEXT


Crawl_Characters:                           ' crawl on (horizontally)
  FOR eeAddr1 = Char_Space TO Pad
    GOSUB ShowChar
    PAUSE 250
  NEXT
  PAUSE 1000


Scroll_Characters:                          ' scroll on (vertically)
  FOR char = 0 TO 3
    LOOKUP char,[Char_Space,Char_B,Char_S,Char_2],eeAddr1
    LOOKUP char,[Char_B,Char_S,Char_2,Char_Space],eeAddr2
    FOR row = 1 TO 8
      FOR col = 1 TO 5
        READ (eeAddr1 + col - 1),vScroll.LowByte
        READ (eeAddr2 + col - 1),vScroll.HighByte
        d7219 = vScroll >> (row - 1) ' get "frame"
        SHIFTOUT Dpin,Clock,MSBFirst,[col,d7219]
        PULSOUT Load,3
      NEXT
      PAUSE 200
    NEXT

  NEXT
  PAUSE 1000


Show_Graph:
  FOR col = 1 TO 5                          ' five colums wide
    READ (GrData + col - 1),grVal           ' get stored data
    vScroll = $FF80 >> (grVal / GraphY)     ' draw bar
    SHIFTOUT Dpin,Clock,MSBFirst,[col,vScroll.LowByte]
    PULSOUT Load,3
  NEXT
  PAUSE 1000
```

```
Crosshair:
  btns = ~JoyBtns >> 2 & %0011              ' read buttons (1=down)
  BRANCH btns,[ShowTarget,ShowSmile,Main,Main]

ShowTarget:
  GOSUB ReadJoyStick

  xAxis = joyXval/(JoyXMax / 5) MAX 4       ' crosshair column
  yAxis = joyYval/(JoyYMax / 7) MAX 6       ' crosshair row

  FOR col = 1 TO 5
    LOOKUP yAxis,[$01,$02,$04,$08,$10,$20,$40],d7219
    IF ((col-1) <> xAxis) THEN DrawColumn   ' draw crosshair
    d7219 = $7F
  DrawColumn:
    SHIFTOUT Dpin,Clock,MSBFirst,[col,d7219]
    PULSOUT Load,3
  NEXT

  GOTO CrossHair

ShowSmile:
  eeAddr1 = Smile                           ' point to definition
  GOSUB ShowChar                            ' show it
  PAUSE 1000
  GOTO CrossHair

  END


' ----[ Subroutines ]-----------------------------------------------------
'
ShowChar:
  FOR col = 1 TO 5                          ' character is 5 columns wide
    READ (eeAddr1 + col - 1),d7219          ' read column data from
EEPROM
    SHIFTOUT Dpin,Clock,MSBFirst,[col,d7219]
    PULSOUT Load,3
  NEXT
  RETURN


ReadJoyStick:
  HIGH JoyX                                 ' discharge RC caps
  HIGH JoyY
  PAUSE 5
  RCTIME JoyX,1,joyXval                     ' read x axis
  RCTIME JoyY,1,joyYval                     ' read y axis
  RETURN
```