



Column #138, October 2006 by Jon Williams:

Goin' With the Glow

I'm pretty sure that Scott Edwards had no idea what he was starting way back in the early 90's when he released his first serial LCD. Back then, the target customer was someone like me; a BASIC Stamp 1 user who wanted a nice display but didn't have a lot of spare I/O pins to support it. Well, as we've seen, serial LCD modules have become about as ubiquitous to electronics experimenters as the 555 timer chip.

There are, however, environments that can present serious challenges to LCDs, specifically those with extremes in ambient lighting and temperature, something industrial users face every day. Enter the VFD; the vacuum tube fluorescent display.

Even if you're not sure what I mean by VFD, it's a pretty good bet that you've seen them. VFDs are self-illuminating and very bright, have a striking blue-green color caste, and on close inspection you can see that the works are contained in a thick glass envelope. Before the explosion of LCDs, VFD displays were actually quite common in consumer devices like VCRs. In fact, when I worked in the irrigation industry we used a VFD designed for a VCR in a golf course sprinkler controller; the elements of the display matched our requirements well, and being a VFD meant that it work under the lighting and temperature extremes we'd encounter in outdoor use on a golf course.

The VFD that we're going to work with is the Noritake GU112X16G-7003. It's about the same size as a 2x16 serial LCD, albeit just a tad thicker. Connections are pretty easy: power (5 vdc, 350 mA max), a 38.4k serial connection, and ground. Do note, though, that the connections on the VFD are not compatible with standard servo extender cables, so if you'll

probably want to modify an off-the-shelf cable or roll your own. For our experiments we can plug the VFD right into the breadboard of a BOE or the PDB.

When you look over the detail specifications you'll see that it mentions jumpers for the selection of baud rates. The default setting is 38.4k, but it can go 115.2k if you intend to use it with an SX chip. But what the Noritake documentation refers to as "jumpers" are actually blank pads that you'll need to bridge with solder if you want to select the higher speed. In most applications this probably won't be necessary.

In practice, you can treat the GU112X16G-7003 very much like a serial LCD after meeting its initial setup requirements. A key difference, however is cursor positioning. You see, this display is inherently graphic, so the cursor positioning commands always refer to a bit when dealing with the X (horizontal) axis. This is nice as it gives us very precise positioning control, even when we're simply using the display for text. Again, that bit-level positioning is only on the X axis, the Y aspect of cursor positioning is always by row (0 or 1).

While we're on the subject of cursor positioning, let's discuss the display and what is versus what isn't seen. Figure 138.1 shows this display memory. Note that 112 horizontal pixels are visible and 400 are hidden – but can be written to when the display is set to "all screen." What you'll see in practice is that the "all screen" mode allows the display of partial characters when they are positioned between normal character boundaries. When in "display screen" mode, where only the visible area of the screen is used, characters pushed to one edge will wrap around to the other, and will do so as whole characters. You can see this in a little demo program I've included in the download package called GU112x16G_ScreenMode.BS2.

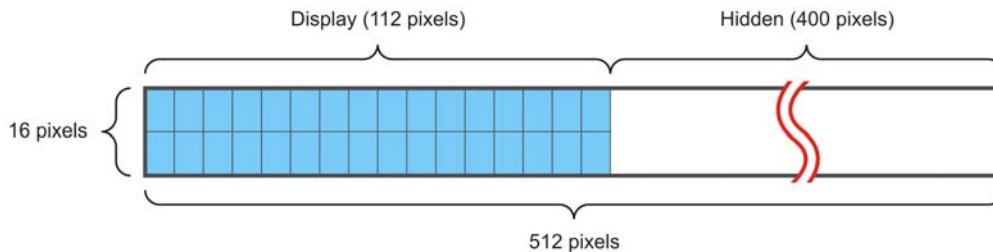


Figure 138.1: GU112X16G-7003 Memory Map

Mix It Up

Where the GU112X16G-7003 differs from VFDs that emulate LCDs is that we can mix text and graphics in the same display. Now, there is a bit of work involved for us to do this,

having mostly to do with the organization of bit images in the display. And this is where our focus will be this month: getting a graphic of our choice into the display. There are times when we as technical types are called on to be a bit artistic, so tools we might not consider in our day-to-day programming chores are going to be pulled out to get the job done.

Let's say, for example, that a customer gives us a logo that they'd like displayed in the GU112X16G while the program is waiting for some sort of user entry, perhaps from an RFID reader. What do we do? Well, the first thing we'll want to do is see if the image is viable as a 112x16 monochromatic bitmap.

You don't need to be a graphics wizard to do this stuff, but it will be helpful if you have some comfort with a program like Photoshop, Paint Shop Pro, or The Gimp. Any of these programs (and there are many others) will let us scale the image to see what it will look like in a 112x16 display. That's step 1. The next part is overlaying a blank map of the display pixels that we can fill in or clear to create the actual pixel map for the GU112X16G. For this I use a graphic called noritake_blank.bmp – this is actually about 8x the display size, making the things a lot easier to change pixels with the fill tool. The idea to pull your raw graphic into the project as a layer below the blank pixel map, scale it to fit the desired size of the pixel map, then set the pixel map transparency down enough so that you can see your image through it. After that it's a matter of filling in and clearing pixels in the map layer to best match the target graphic.

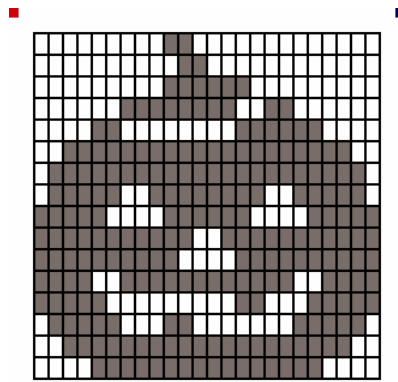


Figure 138.2: Pumpkin Pixel Map

The second part is critical if you want to get an accurate representation of the original image. The reason for this is something called Pixel Aspect Ratio; this is the ratio between the height and width of a pixel. On our computer monitors, this ratio is 1:1 as the pixels are square. But

the GU112X16G has rectangular pixels, with an aspect closer to 3:2, that is, the pixels are taller than they are wide. What you'll find if you attempt to map square pixels is that your image will be too skinny. The pumpkin we're about to use as a demo, for example, looked more like a yellow squash when mapped as square pixels. Figure 138.2 shows my completed pumpkin graphic after tweaking for the PAR and using the blank template with properly shaped pixels. As you can see, the image is square but due to the PAR, it is 24 pixels wide by 16 pixels tall.

Now that we have an image that can be mapped accurately into the display, let's look at how it's stored in the display RAM – this will affect our choice of program storage and downloading design. Figure 138.3 shows the arrangements of bits in one column of a graphic. Note that only eight bits are shown; if the graphic uses two rows (16 bits tall), then the organization is the same: bits run top to bottom, but the data stream to the display starts on the top row. For a 16-bit tall graphic, then, every other byte is a column on the top row of the display, and the alternate bytes are for the bottom row.

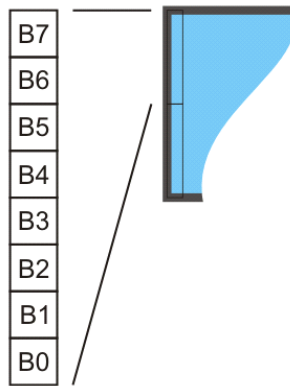


Figure 138.3: Bit Arrangement in a Graphic

It sure would have been easier if the bits were mapped left-to-right instead of bottom-to-top, but the organization used by the GU112X16G probably makes the display internals a little more efficient since each row is exactly eight bits tall. Still, it presents a challenge for us to encode into our listing easily. After staring at the data organization for what seemed like an eternity, I came up with a workable scheme that lets us write DATA statements top-to-bottom to correspond with the image being displayed left-to-right.

What I did is rotated the graphic 90 degrees counterclockwise, and then flipped it around its horizontal axis (note the registration marks in the corners of my demo graphics). Now I can map each horizontal line as an 8- or 16-bit value for the graphic. Figure 138.4 shows the pumpkin graphic after it has been rotated and flipped head to toe. As you can see I also divided the graphic into four-column (nib-sized) chunks; I found this a bit easier on my eyes when transposing the pixels to 1's (filled) and 0's (not filled).

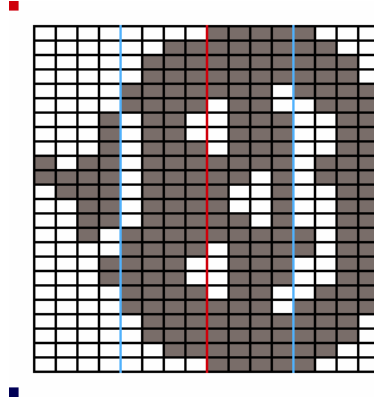


Figure 138.4: The Pumpkin Graphic Rotated and Flipped

With the graphic encoding into a DATA table the final step is to create a routine that will display the image at the current cursor position. To make life simpler with the GU112X16G, I coded what I felt were the most useful features into subroutines that are easy to call. Since there are two possibilities for graphic heights, there are separate routines, but they work about the same. Let's look at displaying a 16-bit tall image.

```
DL_Graphic2:
  SEROUT Sout, Baud,
    [$1F, $28, $66, $11,
    width.BYTE0, width.BYTE1, 2, 0, 1]

  FOR idx = 0 TO width*2-1 STEP 2
    READ (addr + idx), Word temp
    SEROUT Sout, Baud, [tmpHi, tmpLo]
  NEXT
  RETURN
```

As you can see, it's not terribly complicated outside the command sequence required by the VFD. The first four bytes of the serial command specify the real-time bit image display. This is followed by the width of the graphic (low byte, then high byte), the height in rows (two in

Column #138: Goin' With the Glow

this case), and a one for fixed image as called out in the spec. What follows this command is the column data for the image.

I chose to use the Word modifier in the DATA statements so that I have a single string of bits. When we do this, the BASIC Stamp compiler stores the information low-byte/high byte, so it's now out of order from the requirement of the display. What this means is that we can't read and transmit a byte at a time. What we'll do, of course, is read it back a word at a time and then send the bytes in the correct order (first high, then low). Since we're reading word-sized values from the table we need to toss STEP 2 into the FOR-NEXT loop to maintain proper byte alignment for READ. It may look odd, at first, to multiply the width by two (for two bytes for column) and then use a step size of two but, again, we have to do this because READ works on byte boundaries.

An interesting thing happens when we put a 16-bit tall graphic onto the second row: only the top half of the image is displayed. Well, this can actually come in handy, and when you run the demo program you'll see that we can make the pumpkin appear to jump up from the bottom of the display by starting it on row 1, then immediately displaying it in the same column on row 0.

The graphics demo program has another image, an 8-bit tall bat. Creating and encoding the image is identical to what we did with the pumpkin, though we only have eight bits on a line. Since the image is just one row tall we can simplify the FOR-NEXT portion of the download loop as shown here.

```
DL_Graphic1:
  SEROUT Sout, Baud,
    [$1F, $28, $66, $11,
    width.BYTE0, width.BYTE1, 1, 0, 1]

  FOR idx = 0 TO width-1
    READ (addr + idx), tmpLo
    SEROUT Sout, Baud, [tmpLo]
  NEXT
  RETURN
```

Okay, let's put the graphics to use. Our main demo program (GU112x16G_Halloween.BS2) starts by resetting the VFD and changing the write mode to XOR. Remember that any bit XOR'd with itself will be zero, so writing a graphic on top of itself in this mode will cause the second write sequence to erase the first. We want to do this because the beginning of the program is going to "fly" the bat graphic across the display. Within a loop we'll display the bat graphic twice in the same position with a short delay in between; the delay creates the flight timing.

```

Main:
  GOSUB Reset_VFD
  mode = VFD_XOR
  GOSUB Set_Mode

Fly_Bat:
  addr = Bat
  width = 22
  FOR col = 0 TO 112 STEP 22
    FOR row = 1 TO 0
      GOSUB Put_Crsr
      GOSUB DL_Graphic1
      PAUSE 100
      GOSUB DL_Graphic1
      col = col + 22
    NEXT
  NEXT

  mode = VFD_NORM
  GOSUB Set_Mode

```

After the bat flies off the screen we can display our message with simple cursor positioning and text writing.

```

col = 30
row = 0
GOSUB Put_Crsr
SEROUT Sout, Baud, ["HAPPY"]
row = 1
GOSUB Put_Crsr
SEROUT Sout, Baud, ["HALLOWEEN!"]

```

This is easy stuff – again, remember that our column positioning resolution is one pixel.

Next comes the pumpkin display. We can make it “pop up” by putting the row value into a loop, using a short delay in between to control movement timing.

```

addr = Pumpkin
width = 24
FOR row = 1 TO 0
  col = 0
  GOSUB Put_Crsr
  GOSUB DL_Graphic2
  PAUSE 50
NEXT

```

Column #138: Goin' With the Glow

The final part of the graphic demo program gives a bit of insight into what are called “user windows.” The GU112X16G has a base (default) window which encompasses the entire display, and it allows us to define up to four user windows that have individual control. What the user windows don't have is their own coordinate system; position values in the user window are taken from the underlying base window.

So, let's define a user window in the upper right-hand corner of the display.

```
SEROUT Sout, Baud,
      [$1F, $28, $77, $02, 1, 1,
      90, 0, 0, 0,
      22, 0, 1, 0]
```

Zoiks, Batman, that's complicated. Yes, I agree – but if we can get a handle on it there are benefits that allow us to generate sophisticated display actions. The first line in the output data specifies the user definition for window #1. The next line says that the upper-left corner of the window is at column 90, and on row 0. The second line says that the window is 22 pixels wide and one row (eight pixels) deep.

The window is now defined, but we have to select it before doing anything with it. This part is pretty easy.

```
SEROUT Sout, Baud, [$1F, $28, $77, $01, 1]
```

Finally, let's paint the bat image into the window and alternately clear it to make the bat blink. This is different from using the XOR mode as before, and it demonstrates that the Display Clear command of the VFD affects only the current window.

```
FOR flashes = 1 TO 5
  GOSUB Clear_Window
  PAUSE 50
  col = 90
  row = 0
  GOSUB Put_Crsr
  addr = Bat
  width = 22
  GOSUB DL_Graphic1
  PAUSE 150
NEXT
```

Notice that we actually start by clearing the window and then painting in the bat graphic. This sequence leaves the bat graphic in place at the end of the flashing loop. Figure 138.5 shows the final display with mixed text and graphics.



Figure 138.5: The Final Display

Custom Characters

One of the things we all love about character LCDs is the ability to define custom characters. Well, the GU112X16G gives us that capability as well – in fact, up to 16 characters. The difference is that we will actually map out custom characters onto the address of an existing character in the display. Now, the coolest thing, in my opinion, is that we can alternately select between “standard” version of the character and our custom version of the character. We can even have both versions of a character in the display at the same time, as the enabling/disabling of custom characters only affects subsequent writes; anything already in the display is not affected. How cool is that?

The GU112X16G supports multiple character fonts, but to keep things easy we’ll stick the standard 5x7 bit font that will be used for most of our applications. After mapping the new character (like we did with the pumpkin and bat) we will once again rotate and flip the bit pattern so that the bits can be properly mapped into a DATA table (I’ve included my reference images in the download package). Figure 138.6 shows a custom digit “1” and its rotated and flipped version for mapping into DATA statements.

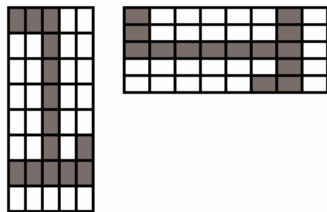


Figure 138.6: Custom Character “1” Original (left); Rotated and Flipped (right)

The GU112X16G allows us to download more than one character at a time, but I think it's simpler to encapsulate the character downloading sequence into a subroutine like this:

```
DL_Char:
  SEROUT Sout, Baud,
    [$1B, $26, $01, char, char, 5]
  FOR idx = 0 TO 4
    READ (addr + idx), tmpLo
    SEROUT Sout, Baud, [tmpLo]
  NEXT
  RETURN
```

Again, this is setup so that we point to the character data with `addr`, and at that location will be five bytes that define the new character shape. The variable called `char` holds the ASCII code of the character that we are redefining.

For the demo I decided to change the decimal digits to an OCR-type font. The new characters are downloaded with a simple loop.

```
FOR char = "0" TO "9"
  addr = Digit0 + ((char-"0") * 5)
  GOSUB DL_Char
NEXT
```

Now to use the custom characters, we have to enable them – that's a pretty simple command sequence.

```
Enable_Custom:
  SEROUT Sout, Baud, [$1B, $25, 1]
  RETURN
```

There's a similar subroutine for disabling the custom characters.

When the demo is running it doesn't look like it's doing anything but, in fact, it's constantly updating the display, switching between the internal digit maps and our custom character maps.

```
DO
  col = 0
  FOR row = 0 TO 1
    GOSUB Put_Crsr
    IF (row = 1) THEN
      GOSUB Enable_Custom
    ELSE
      GOSUB Disable_Custom
    ENDIF
    SEROUT Sout, Baud, ["0123456789"]
  NEXT
  PAUSE 250
LOOP
```

While the column remains fixed at zero we loop through rows zero and one, writing the string of characters. When we're on the top row, the internal characters are used (custom characters are disabled); when we're on the bottom row. Look at the display; it contains both versions (standard and custom) of the decimal digits at the same time.

Check the Specs

I will admit that getting the full specifications for the GU112X16G display was a bit of a challenge. I had to make five calls to five different Noritake engineers. Finally, in the last guy I detected a bit of Japanese accent so I wowed him with my staggering Japanese vocabulary (yes, I'm being facetious – I know enough to explain to a native speaker that I don't quite understand what they're saying). This made him laugh and he sent the full specification. I've included this in the download package as well, so please have a look because there are many more features of the GU112X16G that are worth exploring.

And my old pals at Parallax have started releasing more app notes on the GU112X16G, so do check their web site for anything new. Between what we've done here, the specs, and the Parallax docs and app notes, you should be able to push the GU112X16G to do anything it is capable of doing.

Until next time, have a safe and happy Halloween -- and Happy Stamping!

Project Code

```

' =====
'
' File..... GU112x16G.BS2
' Purpose... Noritake GU112x16G-7003 Template
' Author.... Jon Williams, EFX-TEK (www.efx-tek.com)
' E-mail.... jwilliams@efx-tek.com
' Started...
' Updated... 11 AUG 2006
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
' =====
' -----[ I/O Definitions ]-----
Sout          PIN      15          ' serial out to VFD
' -----[ Constants ]-----
#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
  T9600      CON      84
  T19K2      CON      32
  T38K4      CON       6
#CASE BS2SX, BS2P
  T9600      CON     240
  T19K2      CON     110
  T38K4      CON      45
#ENDSELECT

Baud          CON      T38K4

VFD_BS       CON      $08          ' backspace
VFD_HT       CON      $09          ' horizontal tab (1 char)
VFD_LF       CON      $0A          ' linefeed
VFD_HOME     CON      $0B
VFD_CLR     CON      $0C          ' clear the screen
VFD_CR       CON      $0D

VFD_NORM     CON      $00          ' write modes
VFD_OR       CON      $01
VFD_AND      CON      $02
VFD_XOR      CON      $03

' -----[ Variables ]-----
mode         VAR      Nib          ' display write mode

```

```

xSize      VAR    Nib      ' font font magnification
ySize      VAR    Nib
level      VAR    Nib      ' brightness level

col        VAR    Word     ' cursor location
row        VAR    Nib
width      VAR    Word     ' width of downloaded graphic

char       VAR    Byte
idx        VAR    Word     ' loop control
addr       VAR    Word     ' ee address
temp       VAR    Word
tmpLo      VAR    temp.LOWBYTE
tmpHi      VAR    temp.HIGHBYTE

tNorm      VAR    Byte     ' on timing (x14 ms)
tBlank     VAR    Byte     ' off timing (x14 ms)
tRev       VAR    tBlank   ' reverse timing (x14 ms)
nReps      VAR    Byte     ' number of reps

' -----[ Initialization ]-----
Reset:
HIGH Sout      ' put in idle state
PAUSE 250      ' let display settle

' -----[ Program Code ]-----
Main:
GOSUB Reset_VFD
FOR level = 1 TO 8
  GOSUB Set_Brightness
  SEROUT Sout, Baud, [VFD_HOME, "BRIGHTNESS: ", DEC level]
  PAUSE 500
NEXT

SEROUT Sout, Baud, [VFD_CLR,
                  "  NORITAKE", VFD_CR, VFD_LF,
                  "    VFD"]

tNorm = 40
tBlank = 40
nReps = 3
GOSUB Flash_Display
PAUSE ((tNorm * 15) + (tBlank * 15)) * nReps

tNorm = 5
tRev = 5
nReps = 20
GOSUB Reverse_Display
PAUSE ((tNorm * 15) + (tRev * 15)) * nReps

```

Column #138: Goin' With the Glow

```
GOTO Main
END

' -----[ Subroutines ]-----

Reset_VFD:
  SEROUT Sout, Baud, [$1B, $40]           ' reset the display
  PAUSE 10
  RETURN

' -----

' Clears currently selected window

Clear_Window:
  SEROUT Sout, Baud, [VFD_CLR]
  RETURN

' -----

Put_Crsr:
  SEROUT Sout, Baud, [$1F, $24, col.BYTE0, col.BYTE1, row, 0]
  RETURN

' -----

Set_Mode:
  SEROUT Sout, Baud, [$1F, $77, mode]
  RETURN

' -----

' Sets font magnification for subsequent writes

Set_Size:
  SEROUT Sout, Baud, [$1F, $28, $67, $40, xSize, ySize]
  RETURN

' -----

' Download custom 5x7 character
' -- address of character data (5 bytes) in "addr"
' -- character # in "char"

DL_Char:
  SEROUT Sout, Baud, [$1B, $26, $01, char, char, 5]
  FOR idx = 0 TO 4
    READ (addr + idx), tmpLo
    SEROUT Sout, Baud, [tmpLo]
  NEXT
```

```

RETURN

' -----
' Enable custom characters in VFD

Enable_Custom:
SEROUT Sout, Baud, [$1B, $25, 1]           ' use custom characters
RETURN

' -----
' Disable custom characters in VFD

Disable_Custom:
SEROUT Sout, Baud, [$1B, $25, 0]         ' use internal characters
RETURN

' -----
' Download custom graphic
' -- move cursor to left-edge column before calling
' -- put data location in "addr"
' -- put width of graphic in "width"
' -- assumes graphic uses one row (eight bits tall)

DL_Graphic1:
SEROUT Sout, Baud, [$1F, $28, $66, $11,
                    width.BYTE0, width.BYTE1, 1, 0, 1]

FOR idx = 0 TO width-1
    READ (addr + idx), tmpLo
    SEROUT Sout, Baud, [tmpLo]
NEXT
RETURN

' -----
' Download custom graphic
' -- move cursor to left-edge column before calling
' -- put data location in "addr"
' -- put width of graphic in "width"
' -- assumes graphic uses two rows (16 bits tall)

DL_Graphic2:
SEROUT Sout, Baud, [$1F, $28, $66, $11,
                    width.BYTE0, width.BYTE1, 2, 0, 1]

FOR idx = 0 TO width*2-1 STEP 2
    READ (addr + idx), Word temp
    SEROUT Sout, Baud, [tmpHi, tmpLo]

```

Column #138: Goin' With the Glow

```
NEXT
RETURN

' -----

Set_Brightness:
  SEROUT Sout, Baud, [$1F, $58, level]          ' level = 1 to 8
  RETURN

' -----

Flash_Display:
  SEROUT Sout, Baud, [$1F, $28, $61, $11, 1, tNorm, tBlank, nReps]
  RETURN

' -----

Reverse_Display:
  SEROUT Sout, Baud, [$1F, $28, $61, $11, 2, tNorm, tRev, nReps]
  RETURN

' -----
```

```
' =====
'
'   File..... GU112x16G_Custom.BS2
'   Purpose... Noritake GU112x16G-7003 Demo
'   Author.... Jon Williams, EFX-TEK (www.efx-tek.com)
'   E-mail.... jwilliams@efx-tek.com
'   Started...
'   Updated... 11 AUG 2006
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =====

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

Sout          PIN      15          ' serial out to VFD

' -----[ Constants ]-----

#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
  T9600        CON      84
  T19K2        CON      32
  T38K4        CON       6
#CASE BS2SX, BS2P
  T9600        CON     240
  T19K2        CON     110
```

The Nuts and Volts of BASIC Stamps 2006

```

T38K4      CON      45
#ENDSELECT

Baud       CON      T38K4

VFD_BS     CON      $08      ' backspace
VFD_HT     CON      $09      ' horizontal tab (1 char)
VFD_LF     CON      $0A      ' linefeed
VFD_HOME   CON      $0B
VFD_CLR    CON      $0C      ' clear the screen
VFD_CR     CON      $0D

' ----- [ Variables ] -----

col        VAR      Word
row        VAR      Nib
width     VAR      Word

char       VAR      Byte
idx        VAR      Word
addr       VAR      Word
temp       VAR      Word
tmpLo      VAR      temp.LOWBYTE
tmpHi      VAR      temp.HIGHBYTE

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

Digit0     DATA    %01111100      'custom character maps
           DATA    %10000010
           DATA    %10000010
           DATA    %10000010
           DATA    %01111100

Digit1     DATA    %10000010
           DATA    %10000010
           DATA    %11111110
           DATA    %00000010
           DATA    %00000110

Digit2     DATA    %10001110
           DATA    %10010010
           DATA    %10010010
           DATA    %10010010
           DATA    %01100010

Digit3     DATA    %10000010
           DATA    %10010010
           DATA    %10010010
           DATA    %10010010
           DATA    %01101100

```

Column #138: Goin' With the Glow

```
Digit4      DATA    %11111000
             DATA    %00001000
             DATA    %00001000
             DATA    %01111110
             DATA    %00001000

Digit5      DATA    %11110010
             DATA    %10010010
             DATA    %10010010
             DATA    %10010010
             DATA    %10001100

Digit6      DATA    %01111100
             DATA    %10010010
             DATA    %00010010
             DATA    %00010010
             DATA    %00001100

Digit7      DATA    %11000000
             DATA    %10000000
             DATA    %10001110
             DATA    %10010000
             DATA    %11100000

Digit8      DATA    %00011110
             DATA    %11110010
             DATA    %10010010
             DATA    %11110010
             DATA    %00011110

Digit9      DATA    %01100000
             DATA    %10010000
             DATA    %10010000
             DATA    %10010010
             DATA    %01111100

' -----[ Initialization ]-----

Reset:
HIGH Sout           ' put in idle state
PAUSE 250           ' let display settle
SEROUT Sout, Baud, [$1B, $40] ' reset the display
PAUSE 10

' -----[ Program Code ]-----

Main:
GOSUB Clear_VFD
FOR char = "0" TO "9"           ' loop through digit chars
  addr = Digit0 + ((char-"0") * 5) ' point to new char map
  GOSUB DL_Char                 ' download it
```

```

NEXT
DO
  col = 0                                ' left align
  FOR row = 0 TO 1                        ' set row (line)
    GOSUB Put_Crsr
    IF (row = 1) THEN                     ' if bottom row
      GOSUB Enable_Custom                 ' use custom digits
    ELSE                                   ' else
      GOSUB Disable_Custom               ' use standard digits
    ENDIF
    SEROUT Sout, Baud, ["0123456789"]    ' write digits
  NEXT
  PAUSE 250
LOOP

' -----[ Subroutines ]-----

Clear_VFD:
  SEROUT Sout, Baud, [VFD_CLR]
  RETURN

' -----

Put_Crsr:
  SEROUT Sout, Baud, [$1F, $24, col.BYTE0, col.BYTE1, row, 0]
  RETURN

' -----

' Download custom 5x7 character
' -- address of character data (5 bytes) in "addr"
' -- character # in "char"

DL_Char:
  SEROUT Sout, Baud, [$1B, $26, $01, char, char, 5]
  FOR idx = 0 TO 4
    READ (addr + idx), tmpLo
    SEROUT Sout, Baud, [tmpLo]
  NEXT
  RETURN

' -----

' Enable custom characters in VFD

Enable_Custom:
  SEROUT Sout, Baud, [$1B, $25, 1]        ' use custom characters
  RETURN

' -----

```

Column #138: Goin' With the Glow

```
' Disable custom characters in VFD
Disable_Custom:
  SEROUT Sout, Baud, [$1B, $25, 0]          ' use internal characters
  RETURN
' -----
```

```
' =====
'
' File..... GU112x16G_Font_Size.BS2
' Purpose... Noritake GU112x16G-7003 Demo
' Author.... Jon Williams, EFX-TEK (www.efx-tek.com)
' E-mail.... jwilliams@efx-tek.com
' Started...
' Updated... 11 AUG 2006
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
' =====
' -----[ I/O Definitions ]-----
Sout          PIN      15          ' serial out to VFD
' -----[ Constants ]-----
#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
  T9600      CON      84
  T19K2      CON      32
  T38K4      CON       6
#CASE BS2SX, BS2P
  T9600      CON     240
  T19K2      CON     110
  T38K4      CON     45
#ENDSELECT
Baud          CON      T38K4
VFD_BS       CON      $08          ' backspace
VFD_HT       CON      $09          ' horizontal tab (1 char)
VFD_LF       CON      $0A          ' linefeed
VFD_HOME     CON      $0B
VFD_CLR      CON      $0C          ' clear the screen
VFD_CR       CON      $0D
' -----[ Variables ]-----
```

The Nuts and Volts of BASIC Stamps 2006

```

xSize          VAR      Nib          ' font font magnification
ySize          VAR      Nib

' -----[ Initialization ]-----

Reset:
HIGH Sout          ' put in idle state
PAUSE 250          ' let display settle
SEROUT Sout, Baud, [$1B, $40] ' reset the display
PAUSE 10

' -----[ Program Code ]-----

Main:
FOR ySize = 1 TO 2 ' loop through y sizes
  FOR xSize = 1 TO 4 ' loop through x sizes
    GOSUB Set_Size ' set new write size
    SEROUT Sout, Baud, [VFD_CLR, "N&V"] ' clear and update display
    PAUSE 1000
  NEXT
NEXT

GOSUB Clear_VFD

Mixed_Demo:
xSize = 1 : ySize = 1 ' return to default size
GOSUB Set_Size
SEROUT Sout, Baud, ["N"]

xSize = 4 : ySize = 1 ' wide
GOSUB Set_Size
SEROUT Sout, Baud, ["&"]

xSize = 4 : ySize = 2 ' wide and tall
GOSUB Set_Size
SEROUT Sout, Baud, ["V"]

PAUSE 2000

GOTO Main

' -----[ Subroutines ]-----

Clear_VFD:
SEROUT Sout, Baud, [VFD_CLR]
RETURN

' -----

' Sets font magnification for subsequent writes

```

Column #138: Goin' With the Glow

```
Set_Size:
  SEROUT Sout, Baud, [$1F, $28, $67, $40, xSize, ySize]
  RETURN
'
```

```
' =====
'
'   File..... GU112x16G_Halloween.BS2
'   Purpose... Noritake GU112x16G-7003 Demo
'   Author.... Jon Williams, EFX-TEK (www.efx-tek.com)
'   E-mail.... jwilliams@efx-tek.com
'   Started...
'   Updated... 11 AUG 2006
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =====
' -----[ I/O Definitions ]-----
Sout          PIN      15          ' serial out to VFD
' -----[ Constants ]-----
#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
  T9600      CON      84
  T19K2      CON      32
  T38K4      CON      6
#CASE BS2SX, BS2P
  T9600      CON      240
  T19K2      CON      110
  T38K4      CON      45
#ENDSELECT

Baud          CON      T38K4

VFD_BS       CON      $08          ' backspace
VFD_HT       CON      $09          ' horizontal tab (1 char)
VFD_LF       CON      $0A          ' linefeed
VFD_HOME     CON      $0B
VFD_CLR      CON      $0C          ' clear the screen
VFD_CR       CON      $0D

VFD_NORM     CON      $00          ' write modes
VFD_OR       CON      $01
VFD_AND      CON      $02
VFD_XOR      CON      $03
```

The Nuts and Volts of BASIC Stamps 2006

```

' ----- [ Variables ] -----
mode          VAR      Nib      ' display write mode
col           VAR      Word     ' cursor location
row           VAR      Nib
width        VAR      Word     ' width of downloaded graphic

idx           VAR      Word     ' loop control
addr         VAR      Word     ' ee address
temp         VAR      Word
tmpLo        VAR      temp.LOWBYTE
tmpHi        VAR      temp.HIGHBYTE

tNorm        VAR      Byte     ' on timing (x14 ms)
tBlank       VAR      Byte     ' off timing (x14 ms)
nReps        VAR      Byte     ' number of reps

flashes      VAR      Nib      ' for bat flash

' ----- [ EEPROM Data ] -----
Pumpkin      DATA    Word    %0000000011111000 ' custom graphic (tall)
             DATA    Word    %0000001111111100
             DATA    Word    %0000011111111110
             DATA    Word    %0000011111111110
             DATA    Word    %0000111111101111
             DATA    Word    %0000111101100111
             DATA    Word    %0001011001110011
             DATA    Word    %0001011001110011
             DATA    Word    %0001011011100111
             DATA    Word    %1011011111110111
             DATA    Word    %1111011111010111
             DATA    Word    %0111011110010011
             DATA    Word    %0011011110010011
             DATA    Word    %0011011111010011
             DATA    Word    %0010111111111011
             DATA    Word    %0000111101111011
             DATA    Word    %0001111001110011
             DATA    Word    %0001111001110011
             DATA    Word    %0000111101100111
             DATA    Word    %0000111111101111
             DATA    Word    %0000011111111110
             DATA    Word    %0000001111111100
             DATA    Word    %0000000011111000

Bat          DATA    %00001100 ' custom graphic (short)
             DATA    %00011000
             DATA    %00110000

```

Column #138: Goin' With the Glow

```
DATA %01111100
DATA %11111000
DATA %01110000
DATA %00110000
DATA %00110000
DATA %00111000
DATA %11111100
DATA %01111111
DATA %01111111
DATA %11111100
DATA %00111000
DATA %00110000
DATA %00110000
DATA %01110000
DATA %11111000
DATA %01111100
DATA %00110000
DATA %00001100
DATA %00001100

' -----[ Initialization ]-----

Reset:
HIGH Scut           ' put in idle state
PAUSE 250           ' let display settle

' -----[ Program Code ]-----

Main:
GOSUB Reset_VFD
mode = VFD_XOR
GOSUB Set_Mode

Fly_Bat:
addr = Bat
width = 22
FOR col = 0 TO 112 STEP 22           ' fly bat across display
  FOR row = 1 TO 0
    GOSUB Put_Crsr                   ' position bat
    GOSUB DL_Graphic1                ' draw bat
    PAUSE 100
    GOSUB DL_Graphic1                ' erase bat with XOR
    col = col + 22
  NEXT
NEXT

mode = VFD_NORM
GOSUB Set_Mode

Write_Message:
col = 30
```

```

row = 0
GOSUB Put_Crsr
SEROUT Sout, Baud, ["HAPPY"]
row = 1
GOSUB Put_Crsr
SEROUT Sout, Baud, ["HALLOWEEN!"]

Show_Pumpkin:                                ' pop-up pumpkin
  addr = Pumpkin
  width = 24
  FOR row = 1 TO 0
    col = 0
    GOSUB Put_Crsr
    GOSUB DL_Graphic2
    PAUSE 50
  NEXT

  SEROUT Sout, Baud, [$1F, $28, $77, $02, 1, 1, ' define window #1
                                90, 0, 0, 0, ' top left position
                                22, 0, 1, 0] ' 22 bits by 1 row

  SEROUT Sout, Baud, [$1F, $28, $77, $01, 1] ' select window #1

Flash_Bat:
  FOR flashes = 1 TO 5
    GOSUB Clear_Window
    PAUSE 50
    col = 90
    row = 0
    GOSUB Put_Crsr
    addr = Bat
    width = 22
    GOSUB DL_Graphic1
    PAUSE 150
  NEXT

  PAUSE 3000
  GOTO Main

' -----[ Subroutines ]-----

Reset_VFD:
  SEROUT Sout, Baud, [$1B, $40] ' reset the display
  PAUSE 10
  RETURN

' -----

' Clears currently selected window

Clear_Window:

```

Column #138: Goin' With the Glow

```
SEROUT Sout, Baud, [VFD_CLR]
RETURN

' -----

Put_Crsr:
SEROUT Sout, Baud, [$1F, $24, col.BYTE0, col.BYTE1, row, 0]
RETURN

' -----

Set_Mode:
SEROUT Sout, Baud, [$1F, $77, mode]
RETURN

' -----

' Download custom graphic
' -- move cursor to left-edge column before calling
' -- put data location in "addr"
' -- put width of graphic in "width"
' -- assumes graphic uses one row (eight bits tall)

DL_Graphic1:
SEROUT Sout, Baud, [$1F, $28, $66, $11,
                    width.BYTE0, width.BYTE1, 1, 0, 1]

FOR idx = 0 TO width-1
  READ (addr + idx), tmpLo
  SEROUT Sout, Baud, [tmpLo]
NEXT
RETURN

' -----

' Download custom graphic
' -- move cursor to left-edge column before calling
' -- put data location in "addr"
' -- put width of graphic in "width"
' -- assumes graphic uses two rows (16 bits tall)

DL_Graphic2:
SEROUT Sout, Baud, [$1F, $28, $66, $11,
                    width.BYTE0, width.BYTE1, 2, 0, 1]

FOR idx = 0 TO width*2-1 STEP 2
  READ (addr + idx), Word temp
  SEROUT Sout, Baud, [tmpHi, tmpLo]
NEXT
RETURN
```

```
Flash_Display:
  SEROUT Sout, Baud, [$1F, $28, $61, $11, 1, tNorm, tBlank, nReps]
  RETURN
```

```

=====
'
' File..... GU112x16G_ScreenMode.BS2
' Purpose... Noritake GU112x16G-7003 Template
' Author.... Jon Williams, EFX-TEK (www.efx-tek.com)
' E-mail.... jwilliams@efx-tek.com
' Started...
' Updated... 11 AUG 2006
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
=====
' -----[ I/O Definitions ]-----
Sout          PIN      15          ' serial out to VFD

' -----[ Constants ]-----
#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
  T9600        CON      84
  T19K2        CON      32
  T38K4        CON      6
#CASE BS2SX, BS2P
  T9600        CON      240
  T19K2        CON      110
  T38K4        CON      45
#ENDSELECT

Baud          CON      T38K4

VFD_BS        CON      $08          ' backspace
VFD_HT        CON      $09          ' horizontal tab (1 char)
VFD_LF        CON      $0A          ' linefeed
VFD_HOME      CON      $0B
VFD_CLR       CON      $0C          ' clear the screen
VFD_CR        CON      $0D

' -----[ Variables ]-----
```

Column #138: Goin' With the Glow

```
col          VAR      Word      ' cursor location
row          VAR      Nib       '
mode         VAR      Nib       ' screen mode

' -----[ Initialization ]-----

Reset:
HIGH Sout                    ' put in idle state
PAUSE 250                    ' let display settle
GOSUB Reset_VFD

' -----[ Program Code ]-----

Main:
FOR mode = 0 TO 1
  SEROUT Sout, Baud, [$1F, $28, $77, $10, mode]
  GOSUB Clear_Window
  FOR col = 0 TO 111
    row = 0
    GOSUB Put_Crsr
    SEROUT Sout, Baud, ["Nuts & Volts"]
    PAUSE 25
  NEXT
NEXT
GOTO Main

' -----[ Subroutines ]-----

Reset_VFD:
SEROUT Sout, Baud, [$1B, $40] ' reset the display
PAUSE 10
RETURN

' -----

' Clears currently selected window

Clear_Window:
SEROUT Sout, Baud, [VFD_CLR]
RETURN

' -----

Put_Crsr:
SEROUT Sout, Baud, [$1F, $24, col.BYTE0, col.BYTE1, row, 0]
RETURN

' -----
```