



VideoStamp

This is a reprint from AWC's Stamp Project of the Month column. You can find a different project each month on the Web at AWC's homepage: <http://www.al-williams.com/awce.htm>. This article is copyright 2000 by Al Williams. All Rights Reserved.

The world is a different place then when I started in the computer business. Back then the user interface meant an 029 card punch. The latest convenience was a chadless paper tape. Today's users are spoiled. Look around in any modern game arcade. Do you think the player would spend a quarter to play Atari's Battlezone? (Battlezone was an old vector -graphic tank game; see <http://www.gamearchive.com/video/manufacturer/atari/vector/html/battlezone.html>).

While the Stamp is a handy little device for doing processing, its I/O is geared at control, not user interfaces. That's good because 99% of the time, you want the Stamp controlling devices, not talking to users. But what about that 1% of the time?

In previous projects I've shown you how to hookup a Hitachi LCD, several kinds of keypads, and serial LCDs. Compared to even an average PC program, though, these look like Battlezone sitting next to the latest and greatest Mortal Kombat machine.

This month I'll show you how to use a BOB -II (from Decade Engineering) to create NTSC video from the Stamp. Many modern TV's have video inputs, and if yours doesn't you can still use it with a VCR or a video modulator. Of course, in a real setting, you might use a normal video monitor that has NTSC inputs anyway.

The BOB can drive a monitor by itself, providing 28 columns by 11 rows of text (308 characters total). All of the usual upper and lower case letters are available, as well as numerals and many punctuation marks (although, oddly, there is no comma). The BOB also has several graphic characters, some of which are suitable for making bargraphs -- that's what I'll do this month, is graph some data on my big screen TV.

The BOB can also generate data to an existing video signal. So you could use the BOB as a low-cost video titler, or use it to mark data on to a security camera

video, and similar tasks. We have a client that uses one of these to display telemetry data over video coming from the nose of a vehicle.

With a little circuitry and ingenuity, you could use a BOB to send messages from your home automation system to your TV, for example. Or you could make one of those silly caller ID devices that overlay the caller's name and number on your TV.

All About Bob

The BOB is in a 30-pin SIMM socket. The whole package comes with a complete manual and a SIMM socket in a videotape box. The SIMM socket is not suitable for breadboarding, and neither is the BOB itself. You could solder a 30-pin header to a SIMM socket, of course. However, you only need a handful of the 30 pins in most cases, so just soldered wires directly to the socket. Be careful, the pins are delicate.

For most uses, the only connections you need are:

Pin 1: +8V to +16V unregulated

Pin 2: Ground

Pin 7: Data in (RS-232, TTL -level)

Pin 21: Ground (Character intensity)

Pin 22: Ground (Background intensity)

Pin 24: Connect to pin 26 (Video out to video buffer)

Pin 26: Connect to pin 24 (Video out to video buffer)

Pin 27: Ground (wire to one side of an RCA jack for video out)

Pin 28: Video output (wire to an RCA jack)

There are other connections you can make, including an external reset, baud rate selection, and video input (for genlock mode).

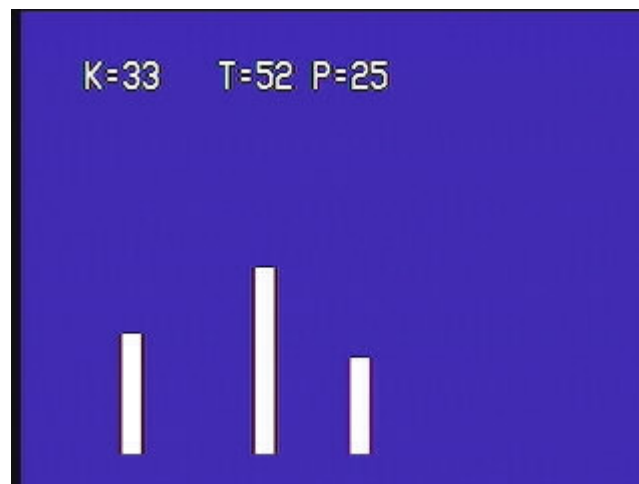
I used a dual RCA jack on a small board (these are available at Radio Shack or in my junk box) and wound up with 3 wires going to my breadboard: power, ground, and data. Notice that the power is not 5V regulated power. Since I kept moving the board between my lab and the big screen TV, I used a 9V battery for a while. I finally switched to a small transformer power supply once I wasn't moving back and forth so often.

Programming BOB

Programming the BOB with the Stamp is simple. You simply use SEROUT to send text to the BOB as you might expect. However, you can also send special commands that start with the { character. For example, {A clears the screen and resets the cursor. {Cxy sets the cursor to the specified X and Y position. The command must have 5 characters, so to go to column 2 row 2 (the columns and rows are zero-based) you'd write {C0202. In local mode (but not genlock mode) you can also set character and screen colors -- although some of the colors are not very bright, and the manual mentions that the color varies from unit to unit.

To send any arbitrary character, including the special graphics characters, you can send a {T followed by the hex codes indicated in the manual for each character. You end with an escape (\$1B). Obviously, that means you can't send the character for \$1B, but that's an uppercase P anyway, so that shouldn't be a problem.

The arbitrary characters of interest in this month's code are from \$72 to \$77. These characters are blocks of various heights that can stack up. \$77 fills the entire vertical character area. \$72 fills 1/6 of that area. With a little math, you can make nice bargraphs using these characters:



The Code

The program shown below will drive the BOB on pinout 7 (you can change that, of course). The main program just makes up some numbers and displays them using the plotting routines below. You set the minimum and maximum values you want to be able to plot, and the column that the plots should appear at. If you call bobplotlbl, the top row will get the value printed (the 33, 52, and 25 in the picture). If you want other labels (like K=, T=, and P=) your main program will have to print them. If you just want the histogram bars, call bobplot.

The code works by converting the input value into a proportional number within the range of the display (0 to 59). Then the code loops through rows 1 to 10 of the indicated column (row 0 is reserved for the labels). This corresponds to a range of 0 to 59 since each of the bargraph characters increase in value by 1/6 of a character cell.

For each row, one of three conditions applies. If the value of the bottom of the row is greater than the value, then the entire row should be blank. If the value of the top of the row is less than the adjusted plot value, then the entire row should be white (\$77). However, one character will be right at the edge -- the lower part of the cell will be greater than the adjusted value, but the upper part will be less than the value. This is where the program has to select the correct bargraph character. If the current row is in variable `cy` and the adjusted value is `val`, the code looks like this:

```
$72+val-(10-cy)*6
```

In the screenshot, you'll notice that although 52 is more than half of 59, the bar is only about halfway up. That's because the value is adjusted by the minimum and maximum values you specify (in this case, a span of 100). So 52 is about half of 100, so the bar's height is roughly 30 units.

A Few Cautions

I found two parts about programming BOB a bit tricky. First, the `{A}` clears the screen and returns the cursor to the home position. However, it doesn't reset any color commands, so keep that in mind. When I first got the BOB, I quickly hooked it up and wrote "Hello World" or something to my TV. Everything worked fine, and I used a baud rate constant of 84 -- 9600 baud on a BS2.

Later I hooked up the circuit I used for this month, and it simply wouldn't work. I tried everything. Finally I went back to printing a "Hello" message and that didn't work either! To make a long story short, the baud rate had mutually shifted to the point that I had to use an 82 or 83 to get the two devices to work. Keep in mind that the BS2 and the BOB use a ceramic resonator for timing and there is some bit rate error in the BS2 (and probably the BOB, too). Apparently the heat here (did I mention it has been a hot Texas summer?) and the tolerance of the various components conspired to make the bit error on both sides add up until communications were not reliable when the BS2 was set at baud mode 84.

Another issue that shouldn't have surprised me is that the BOB doesn't erase anything it doesn't have to. So if you are counting down at a certain cursor location, you might see an 11, followed by a 10. Then you'll see a 90! That's because the 9 takes one position and the 0 is left over from the 10. That means you should either use a fixed number of digits, or write enough spaces after the

number to take care of the worst case. Very smart code could output a vary
number of blanks depending on the value just written.

ing

Goodbye BOB

The BOB module is very powerful and performs a function you aren't likely to design yourself. For all of its power, however, its interface is simple ASCII sent on one pin with a SEROUT command! You probably won't use a BOB on every Stamp project you build -- no more than you'd use an A/D or an LCD on every project. But when you need to drive a video output, it's the way to go.

Stamp Code

```
' bob graphics
' bob has 28 col x 11 rows the bar graph characters split
' each character into 6 pieces so if the Y-axis is 10 rows
' the full scale is 0-59
bob con 7 ' data pin
baud con 83 ' baud rate (see text)
cx var byte ' x & y char pos
cy var byte
' plot variables
minval var word ' minimum range
maxval var word ' max range
val var word ' value to plot
plotcol var byte
plottemp var byte
i var word
minval=0
maxval=100
gosub bobcls
cx=1
gosub bobsetxy
serout bob,baud,["K="] ' label
cx=8
gosub bobsetxy
serout bob,baud,["T="] ' label
cx=13
gosub bobsetxy
serout bob,baud,["P="] ' label
for i=1 to 100
plotcol=10
val=i
gosub bobplotlbl
plotcol=15
val=i/2
gosub bobplotlbl
plotcol=3
val=33
gosub bobplotlbl
pause 500
next
end
```

```

' Plot to BOB with value label
bobplotlbl:
cx=plotcol
cy=0
gosub bobsetxy
' Adjust the space below to make sure your number is always clear
' but without wiping other text or use DEC2, DEC3, etc.
serout bob,baud,[dec val," "]
' Just plot bar
bobplot:
val=(val-minval)*60/maxval
cx=plotcol
for cy=1 to 10
gosub bobsetxy
plottemp=(10-cy)*6
if plottemp > val then bobspace
if plottemp+6 <= val then bobsolid
' first plot position
serout bob,baud,["{T", $72+val-plottemp, $1B]
goto bobnext
bobspace:
serout bob,baud,[" "]
goto bobnext
bobsolid:
serout bob,baud,["{T", $77, $1B]
bobnext:
next
return
' Clear BOB
bobcls:
serout bob,baud,[$1B,"{A"}]
pause 100
return
' Set BOB cursor location
bobsetxy:
serout bob,baud,["{C",dec2 cx,dec2 cy]
return

```