SD=8516 Stellar Basic

Stellar BASIC

Stellar BASIC started life as a typical microcomputer TinyBASIC and has grown to include several features of a more complete microcomputer BASIC such as DIM, READ/DATA, strings (ex. A$) and string operations. It is implemented in SD-8516 Machine Language, and comes standard in the VC-3 Kernal for the SD-8516.

If you have never programmed in BASIC before, please see Chapter 3 of the SD-8516 User's Guide for an introduction to BASIC. Stellar BASIC is a typical microcomputer TinyBASIC, so if you have programmed in BASIC before, you may find *this* tutorial moves at a pace you will find acceptable.

It's fast!

Stellar BASIC is a kind of 8 bit TinyBASIC, with a few improvements under the hood. It can execute over 3,000 lines per second at 4 MHz! Although of course, graphics-heavy statements can slow this down to 800 lines/second. Check out our Practical Speed Tests.

Now, compared to a C64 running at 1mhz, why is it so fast? You would expect it to top out at ~800 lines per second since it's four times faster. First, the SD-8516 does not experience the register pressure of a 6502. With more registers available to the programmer, the code size is smaller, and therefore faster.

Secondly, Commodore BASIC 2.0 is inefficient. It treats all numbers as floats, which is expensive and slows down the interpreter. Stellar BASIC is more advanced because it has 26 dedicated INT registers (A-Z). When you use A to Z, it is a native INT (native 16 bit register width).

Terminal Control

There are some basic terminal interface commands you may wish to know. These will allow you to make character mode games; see next chapter for graphics mode.

- STRXY X, Y, A$ – print string at XY on the screen
- CHARXY X, Y, C – puts character C (0-255) on the screen at location (X,Y).
- CHARFG X,Y,FG – changes the foreground color of a single character on the screen
- CHARBG X,Y,BG – changes the background color of a single character on the screen
- CHARAT$(X,Y) – returns the character at X, Y

As an example of these commands, the game ROBOTS uses CHARXY to draw the map.

Graphics

MODE

One of the interesting commands you will come to use often is the MODE command.

Command	Result
MODE 1 MODE 40	Reset into 40×25 TEXT mode.
MODE 2 MODE 80	Reset into 80×25 TEXT mode.
MODE 3	Reset into MODE 3: 320x200x16 graphics mode (4 bits per pixel)
MODE 6	Reset into MODE 6: SuperTerminal 80×25 “DYNATERM 8800” mode

Mode 80 is good! And, you might like the different color scheme it defaults to (more on that below). It's useful if you want to play some text adventure games, as many people today are much more used to 80 column displays than 40. I must say, I always found games like Pirate Adventure easily playable on 40 columns, but there's something to a game like Trinity played in 80 columns. Now, you can finally take your pick.

Mode 1 and Mode 2 are aliases for MODE 40 and MODE 80. MODE 3 is not a text mode, but enables GRAPHICS MODE.

Note that the terminal functions cease operation in GRAPHICS MODE. If you escape out of a program, you must type MODE 40<ENTER> or a similar command to return to TEXT mode, but you will not see your characters echo on screen.

So, while you can still type commands in mode 3, it is recommended that you use it within a program.

MODE 6

Mode 6 is a new mode in OS version 0.6.26, and is now fully supported by Stellar BASIC v1.0. However, the DEFCHAR and DRAWCHAR functions do not work with it because it relies on the DYNATERM 8800 TVC (text-to-video chip) which stores it's own font data. It is, however, a beautiful mode to work in. You can enter it by typing MODE 6 (and return to mode 1 or 2 any time).

POKE for Graphics

Here's a little expose on how to use POKE for drawing. Let's begin with a simple graphics demo. As you can see we are using POKE to write directly to the framebuffer:

```
10 REM MODE 3 POKE DEMO 1 BY APPLEDOG 2026-03-01
20 PRINT "MODE 3 DEMO (C) 1985 STELLAR DYNAMICS"
30 INPUT "PRESS ENTER TO BEGIN MODE 3 DEMO", A$
40 MODE 3
```

```
 50 REM MODE 3 FRAMEBUFFER STARTS AT $0000 IN BANK 2
 60 BANK 2
 70 REM DRAW HORIZONTAL BARS OF ALL 16 COLORS
 80 REM FILL SCREEN WITH 16 COLOR COLUMNS
 90 FOR Y = 0 to 199
100 FOR C = 0 TO 15
110 REM PACK TWO PIXELS OF SAME COLOR: (C << 4) | C
120 LET V = C * 16 + C
130 REM DRAW 10 BYTES WIDE (20 PIXELS) PER COLOR
140 FOR X = 0 TO 9
150 LET A = Y * 160 + C * 10 + X
160 POKE A, V
170 NEXT X
180 NEXT C
190 NEXT Y
200 REM DRAW DIAGONAL LINE OF WHITE PIXELS
210 FOR I = 0 TO 99
220 POKE I * 160 + I, 2
230 NEXT I
240 REM WAIT FOR KEYPRESS
250 INPUT A$ : REM PRESS A KEY TO END DEMO
260 MODE 1
```

The problem with this is that it's a bit slow. In fact, it executes in 6 seconds at 100mhz.

The reason why it takes so long is because Mode 3 uses a packed pixel format – one byte holds two pixels (each pixel is four bits). In fact, if we wrote this program normally it would take over 12 seconds because we would have to PEEK the byte, modify it, and write it out again. The 6 seconds it took is much faster because we used an optimization that draws two pixels at once.

If you'd like to use something a bit faster, you can try the 2d accelerated graphics commands PIXEL and PEXEL.

PIXEL and PEXEL

Inside MODE 3 you can use the PIXEL command. This command is exactly like a POKE except that it parses the color variable into packed pixel format automatically. It is therefore much faster than using POKE to write directly to Video RAM.

To try the PIXEL command, enter mode 3 and type:

```
PIXEL X, Y, COLOR
```

X and Y are the locations on the screen to plot, and COLOR is a palette number from 0 to 15. If a larger number is used only the lower four bits will be used to calculate the actual color.

Together with PIXEL is PEXEL, which reads a pixel color.

```
    LET A = PEXEL(5,10)
```

Here, 5 is the X location and 10 is the Y location on the screen of the pixel you are reading.

```
    10 MODE 3
    20 FOR C = 0 TO 15
    30 FOR X = 0 TO 19
    40 FOR Y = 0 TO 199
    50 PIXEL C * 20 + X, Y, C
    60 NEXT Y
    70 NEXT X
    80 NEXT C
    90 REM DIAGONAL LINE IN WHITE
   100 FOR I = 0 TO 199
   110 PIXEL I, I, 3
   120 NEXT I
   130 INPUT "PRESS ENTER"; A$
   140 MODE 1
```

If you time this program you will find that it is even slower than the POKE version. This is because we're writing out the full 64,000 pixels– the POKE version only wrote 32,000. If you weigh it pound for pound, comparably, PIXEL is about 30% faster than using POKE to draw.

160x200 CGA Mode

Regarding PIXEL vs POKE for drawing, the interesting thing about POKE is that it allows you to simulate a 160×200 graphics mode with just one write. Just use POKE to draw (COLOR*16)+COLOR and it will look like 160×200. This enables you to replicate CGA 160x200x16 mode, and similar modes like those found on a C64/128 (medium bitmap mode), Tandy 1000/2000, PC Jr. and so on. Many games (like Jumpman!) were written using these modes. Other games include like King's Quest, Maniac Mansion, Ghostbusters, California Games, and Bruce Lee. It's a very interesting mode!

On the C64 for example, all of these games also use 160×200 mode:

- Pitfall II, the lost caverns
- Flight Simulator 2.0
- Dragon's Lair
- Elevator Action
- Ghosts 'n Goblins
- Phantasie and Phantasie II
- River Raid
- Robotron 2048
- Rygar
- Smash TV

Some versions of infocom games (like ZORK) used 160×200 mode for display as well.

It's still slow to POKE, but if you're just drawing a scene or some letters or art, you can easily get away with POKE in 160×200 mode.

LINE

```
10 MODE 3
20 FOR C = 0 TO 15
30 FOR X = 0 TO 19
40 LINE C * 20 + X, 0, C * 20 + X, 199, C
50 NEXT X
60 NEXT C
70 LINE 0, 0, 199, 199, 3
80 INPUT "PRESS ENTER"; A$
90 MODE 1
```

Line drawing is much faster. It takes 0.14 seconds at 100mhz. That is an incredible speed-up over the previous record. But there is an even faster command than this – the RECT drawing command.

RECT

Yes! What's even better than a LINE? A rectangle!

```
10 MODE 3
20 FOR C = 0 TO 15
30 RECT C * 20, 0, C * 20 + 19, 199, C, 1
40 NEXT C
50 LINE 0, 0, 199, 199, 3
60 INPUT "PRESS ENTER"; A$
70 MODE 1
```

This program is even shorter than the others, and ever so much faster! Clocking in at well under a tenth of a second, it is the fastest version of this demo yet.

CIRCLE

Topping off our graphical journey is the CIRCLE command. It draws a circle or an ellipse on the screen.

```
10 MODE 3
20 CLS
30 LINE 0,0,100,100,3
40 CIRCLE 100,100,36,30,9
50 INPUT A$
```

```
    60 MODE 1
```

The format is:

```
  CIRCLE x1,y1, xr, yr, color
```

Now you can create truly stunning art on the SD-8516!

DRAWCHAR

Some may feel the loss of characters on the mode 3 screen deep in their hearts and long for the days of alphanumeric bliss. For those of you, we have DRAWCHAR.

```
   10 MODE 3
   20 CLS
   30 DRAWCHAR 8, 8, 72, 15, 1
   40 DRAWCHAR 16, 16, 69, 14, 2
   50 DRAWCHAR 24, 24, 76, 13, 3
   60 DRAWCHAR 32, 32, 76, 15, 13
   70 DRAWCHAR 40, 40, 79, 3, 12
   80 DRAWCHAR 32, 72, 87, 10, 6
   90 DRAWCHAR 40, 64, 79, 9, 7
  100 DRAWCHAR 48, 56, 82, 11, 14
  110 DRAWCHAR 56, 48, 76, 7, 9
  120 DRAWCHAR 64, 40, 68, 6, 10
  130 INPUT A$
  140 MODE 1
```

The command is:

```
  DRAWCHAR X, Y, CHARCODE, FG, BG
```

It says “HELLO WORLD”, but in mode 3! You can omit the BG if you want and it will only draw the character. Good for preserving game backgrounds.

DEFCHAR

Life wouldn't be fun if you couldn't define your own characters for use in games! Try the DEFCHAR function.

```
  DEFCHAR char_code, b0, b1, b2, b3, b4, b5, b6, b7
```

This allows you to define your own PETSCII graphics characters *(almost none are included by default). Here's the DEFCHAR function in action:

```
10 MODE 3
20 CLS
30 DEFCHAR 128, $3C,$42,$A5,$81,$A5,$99,$42,$3C
40 DRAWCHAR 100, 100, 128, 15
50 INPUT A$
60 MODE 1
```

This draws a happyface in the middle of the screen in Mode 3.

VSTOP

You may be concerned that, during the execution of a clear and redraw cycle, your screen will show tearing since the video chip updates independently of framebuffer memory. The solution is to turn off the video update cycle.

```
VSTOP
```

This will stop the video chip from updating the screen. The same image as before will be displayed.

```
VSTEP
```

When you wish the screen to update, a maximum of 120FPS, you may issue a VSTEP command. This will redraw the screen.

```
VSTART
```

VSTART will resume regular video chip scans of the framebuffer to video memory.

Mode 4

Mode 4 is a 256×224 “Famous Console” inspired mode. Here's a demo program:

```
10 MODE 4
20 FOR C = 0 TO 15
30 RECT C * 16, 0, C * 16 + 15, 223, C, 1
40 NEXT C
50 LINE 0, 0, 223, 223, 3
60 INPUT "PRESS ENTER"; A$
70 MODE 1
```

As you can see, it's pretty much more of the same, but you may prefer the aspect ratio for classic 8bit and 16bit console games.

Mode 8

Mode 8 is a 128×128 “Ersatz-8” inspired mode.

```
10 MODE 8
20 FOR C = 0 TO 15
30 RECT C * 8, 0, C * 8 + 7, 127, C, 1
40 NEXT C
50 LINE 0, 0, 127, 127, 3
60 INPUT "PRESS ENTER"; A$
70 MODE 1
```

As you can see, it's pretty much more of the same, but you may prefer this aspect ratio for classic “ersatz retro” games.

Music

Music and sound effects are an important part of any game. For many of us it is impossible to forget the iconic video game themes of yesteryear. Stellar BASIC honors this tradition with the PLAY command.

Ode to Joy

```
PLAY "T120 O3 L4 E E F G G F E D C C D E E D2"
```

The above PLAY statement plays a theme from “Ode to Joy”. The structure of the PLAY statement is easy enough to understand: Tempo 120, Octave 3, Note Length 4 (Quarter notes), and then the music: E E F G, G F E D, and on, notes played in order.

Jingle Bells

```
PLAY "T200 L4 O3 MN E E L2 E L4 E E L2 E L4 E G L3 C L8 D L1 E"
PLAY "L4 F F L3 F L8 F L4 F E L2 E L8 E E L4 E D D E L2 D G"
PLAY "L4 E E L2 E L4 E E L2 E L4 E G L3 C L8 D L1 E"
PLAY "L4 F F L3 F L8 F L4 F E L2 E L8 E F L4 G G F D L2 C"
```

Here, we see the MN command and L commands used to great effect. MN means normal, and L# sets the length of the next notes. However, you can also write it like this:

```
PLAY "T200 L4 O3 MN"
PLAY "E E E2 E E E2 E G C3 D8 E1"
PLAY "F F F F8 F8 F E E E8 E8 E D D E D2 G2"
PLAY "E E E2 E E E2 E G C3 D8 E1"
PLAY "F F F F8 F8 F E E E8 F8 G G F D C2"
```

Here, instead of using a lot of L commands, we just add the length to the end of the note for any note that is not the default length.

FURELISE.BAS

Let's take a closer look at the PLAY command by writing our first music program, FURELISE.BAS:

```
10 REM FUR ELISE
20 PLAY "XV1 W1 T140 L8 O5"
30 PLAY "XV2 W1 T140 L8 O2"
```

The commands here are easy to understand. This is the voice setup. The first command in each PLAY line is XV0 or XV1. This sets the commands to use voice 0 or voice 1.

```
XV1            This sets the PLAY command to use voice 1.
XV2            This sets the PLAY command to use voice 2.
```

Next, we have the W1 command. This sets waveform 1. The SD-450 gate register values are:

- 0 = off
- 1 = square
- 2 = triangle
- 3 = sawtooth
- 4 = sine
- 5 = pulse
- 6 = noise

Now, 1 (square) is the default, so you don't need to type it. But we included it here for edification.

```
W1             Set waveform to 1 (square).
W2             Set waveform to 2 (triangle).
```

Next, we set tempo to 120, a standard value. Well, let's spice things up a little and choose 140 this time.

```
T120           Set tempo to 120
T140           Set tempo to 140
```

Finally, we have the default note length, 8, which means eighth note, and the starting octave:

```
L8 O5          Set defaults to quarter note and octave 5 (mid-
```

```
range/upper).
  L8 O2          Set defaults to quarter note and octave 2 (lower).
```

Next lets look at few bars of notes:

```
    40 PLAY "XV0 E D# E D# E O4 B O5 D C XW"
    50 PLAY "XV1 R4. R4. XW"
```

There are two things going on here. One is the main line of notes for voice 1 containing the first 8 notes and an XW command. The second PLAY here contains information for voice 2. It starts with two rests and then an XW command. The other thing to note about the first PLAY is that it contains an O4 command which changes the octave to 4.

```
  O4             Set default octave to 4
```

This is just like above, You can set the octave any time to change it. But, what is XW?

```
  XW             Wait for all other active voices to reach XW and then
continue
```

XW is the synchronize voice command. XW ensures that when the next lines of PLAY execute they will be perfectly synchronized with each other. There are other ways to synchronize music, such as using PLAY STOP, loading the music, then PLAY START; but XW is the way to do it inside a PLAY command.

```
  PLAY STOP      Pause the player.
  PLAY START     Start the player.
```

When the player is stopped, you can add music and it won't start until you do PLAY START. But for our example we have plenty of time to enter the music before the first synchronization point is reached.

Another way to do this is to do something like PLAY “W0 R1 XW” and this rest will give you enough time to load at least the first bar of music, then you can go on to load other bars in the background while the music plays.

That's it! It's a very simple system.

Program Listing

Here is the complete program listing.

```
    10 REM FUR ELISE
    20 PLAY "XV1 W1 T140 L8 O5"
    30 PLAY "XV2 W1 T140 L8 O2"
    40 PLAY "XV1 E D# E D# E O4 B O5 D C XW"
    50 PLAY "XV2 R4. R4. XW"
    60 PLAY "XV1 O4 A4. L8 C E A XW"
```

```
    70 PLAY "XV2 A4. O3 E A XW"
    80 PLAY "XV1 O4 B4. L8 E G# B XW"
    90 PLAY "XV2 O2 E4. O3 E G# XW"
    100 PLAY "XV1 O5 C4. L8 O4 E O5 E D# E D# E O4 B O5 D C XW"
    110 PLAY "XV2 O2 A4. O3 E R O2 R4. R4. XW"
    120 PLAY "XV1 O4 A4. L8 C E A XW"
    130 PLAY "XV2 O2 A4. O3 E A XW"
    140 PLAY "XV1 O4 B4. L8 E O5 C O4 B XW"
    150 PLAY "XV2 O2 E4. O3 E A XW"
    160 PLAY "XV1 O4 A4."
    170 PLAY "XV2 O2 A4."
```

MML COMMANDS

This is 'Music Markup Language', used by many different implementations of BASIC. Our version, of course, is the best one – but they are all very similar.

```
  A-G             - play note (optional sharp/flat, length override, dot)
  A#8 B-.         - Play A Sharp eighth note then B Flat with +1/2 length
(i.e. dotted)
  R               - rest
  O n             - set octave (0-7)
  > <             - octave up/down
  L n             - set default note length (1=whole, 2=half, 4=quarter...)
  T n             - tempo in BPM (32-255)
  V n             - volume (0-15)
  W n             - waveform (0=tri, 1=saw, 2=pulse, 3=noise)
  P n             - pulse width (0-4095; PWM only)
  MS / MN / ML    - staccato / normal / legato
  XV n            - switch voice (0-7)
  XC              - clear current voice queue
  XW              - wait (sync point — all active voices must reach XW)
```

Further Study

If you wish to learn more about how the MML system works under the hood, please refer to the Programmer's Reference Guide Appendix 5 Sound System.

Benchmarking your Code

You might want to know how fast your code runs. This simple benchmarking program gives you an insight on how to create a simple timer:

```
    10 BANK 1
    20 LET T=PEEK($EF06)
```

```
30 FOR I=1 TO 1000
40 LET A=I*3+7
50 LET B=A/2-1
60 LET C=A+B
70 LET S=S+C
80 IF C>100 THEN LET P=P+1
90 NEXT I
100 LET U=PEEK($EF06)
110 LET D=U-T
120 PRINT D
```

The memory location $EF06 is the second byte of the hardware clock, which ticks once every .256 seconds. it's a bit rough but if you need a simple timer you can do something like this. On my computer this returns 14, which indicates 7000/(14*.256) or 1,953 BASIC lines per second. Of course, we already have the WAIT command so this is not useful for waiting on time; but rather it can be used to measure approximately how long something took.

Dictionary

FREE()

This is like MEM in that it calculates the free space in bank 0 between the end of the BASIC program and VARTOP, however it returns it as an integer. Be careful because in BASIC integers are signed. So read the number like this; when it is negative, this shows how much space has been used (the zero page is reserved so it always shows -256 at start). When it is positive, then it shows how much space is left.

Also see: MEM

MEM

This command calculates the free space in bank 0 between the end of the BASIC program and VARTOP, which roughly estimates how much free space you have left in bank 0 for BASIC.

Also see: FREE()

PRINT

This command prints a number, expression, variable or string. Examples:

```
PRINT "HELLO WORLD!"
PRINT A$
PRINT ASC("*")
PRINT 5
```