High-Resolution Sprite-Oriented Color Graphics

You don't need Logo to use sprites for animation with the illusion of depth.

Steve Ciarcia
POB 582
Glastonbury, CT 06033

A funny thing happened on my way to writing this article. Very rarely do I ever know what BYTE's monthly theme is when I am planning a project. The editors tell me, but I am always working on so many hardware projects simultaneously that I can't keep track. And I sometimes juggle my project schedule at the last minute.

This time, three weeks before my deadline, I told Senior Editor Gregg Williams that I was designing a sprite-graphics interface for August. He reminded me that the theme of the issue was Logo and that my project was a perfect enhancement to a Logo package produced by Terrapin Inc. of Cambridge, Massachusetts.

"What's Logo?" I thought to myself, but not wishing to appear completely ignorant, I took his word for it and sent my wire-wrapped prototype board to Leigh Klotz Jr. and Patrick Sobalvarro at Terrapin. It took them less than a week to devise ways to control my sprite-graphics interface using the Logo language.

Their help came just at the right time. Since I was struggling with using assembly language to draw the pictures necessary for this article, I gratefully accepted a copy of the Terrapin MIT Logo language from them, along with the Logo routines they wrote to manipulate sprites. Using Terrapin's software, I quickly came to understand why Logo and a sprite-graphics interface are a natural combination.

The key component is the TMS9918A Video Display Processor.

But you don't have to have Logo to use the sprite-graphics board. You can approach this project either as a versatile color graphics interface that you can mold to fit your requirements or as a sprite-graphics system for use with Terrapin MIT Logo. In either case, you will not be disappointed.

The TMS9918A VDP

The key component in this month's project is an integrated circuit from Texas Instruments, the TMS9918A Video Display Processor (VDP). This chip offers features that are not, to my knowledge, found in any other graphics system. A summary of its capabilities is shown in table 1.

The TMS9918A VDP is intended to be interfaced to a host microprocessor through an 8-bit bidirectional data bus and three control lines. The VDP's output is a composite color video signal, which can be fed directly into a video monitor or, with the addition of an RF (radio-frequency) modulator, to the antenna terminals of a television set.

Up to 16K bytes of dynamic RAM (random-access read/write memory) can be attached directly to the VDP. This VRAM (video RAM), which contains the data that defines the graphics image to be displayed, is automatically refreshed by the VDP. The VRAM needs no direct connection to the host computer.

The host processor interacts with the 9918A by reading from or writing to its registers or the VRAM. The interpretation of the data flow is controlled by the states of the three control lines. The timing of register and VRAM updates is asynchronous with the video output; thus the host processor can communicate with the VDP at any time.

Certain figures and diagrams pertaining to the TMS9918A are reprinted courtesy of Texas Instruments Inc.

1. display resolution of 256 by 192 pixels
2. 16 colors, including black and transparent
3. supports 16K bytes of separate video memory
4. real-time interrupt capability
5. 32 sprites for simulation of three-dimensional effects
6. composite video output
7. four display modes:
   a. graphics I (256 by 192 dots—limited color)
   b. graphics II (256 by 192 dots—extended color)
   c. text mode (24 lines of 40 user-defined characters)
   d. multicolor mode (64 by 48 low-resolution positions)
8. external video and sync inputs
9. automatic, transparent dynamic RAM refresh

Table 1: *Characteristics of the Texas Instruments TMS9918A Video Display Processor integrated circuit.*

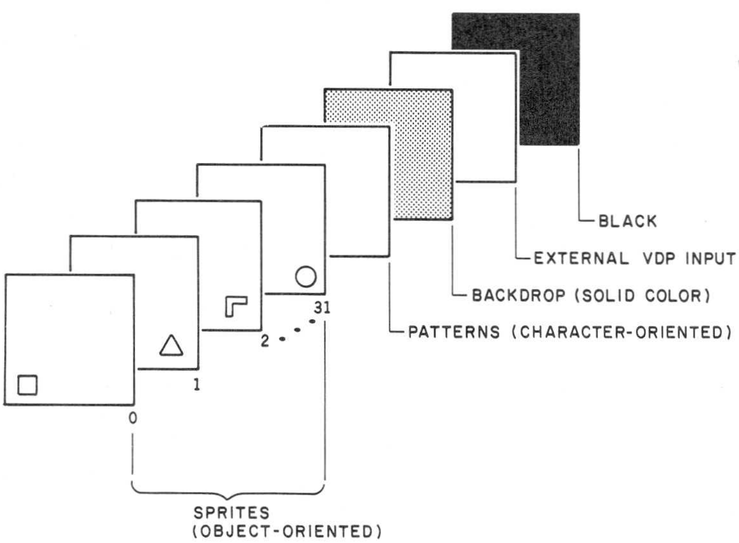


Figure 1: *The TMS9918A's screen image can be envisioned as a set of overlapping display planes sandwiched together. Image objects in planes figuratively closer to the viewer (the top layers of the sandwich) seem to be in front of objects on planes further away (the bottom layers of the sandwich). The top 32 sprite planes are in front of the pattern plane, the backdrop plane, and the external VDP (video) plane, which can contain a video image from almost any compatible external source. The 9918A combines the multiple image sources to form a single composite image.*

Distinctive Architecture

The TMS9918A VDP displays an image on the screen that can be best envisioned as a set of overlapping display planes sandwiched together, as shown in figure 1. This distinctive graphics architecture makes possible the simulation of depth relationships between animated objects in the display without the use of complex hidden-line algorithms.

Image objects in planes figuratively closer to the viewer (the top layers of the sandwich) have higher priorities of visibility than the planes further away (the bottom layers of the sandwich). When the objects on two different planes attempt to occupy the same spot on the screen, the object on the higher-priority plane will be seen by the viewer. For an object on one of the lower-priority planes to be visible, all planes in front of the object's plane (the higher-priority planes) must be transparent at that point.

The top 32 planes are designated for the display of special graphics objects called sprites, which I'll explain shortly. Behind the sprite planes is the pattern plane. The pattern plane is used for text and graphics generated in one of four color-display modes. This pattern plane works like a conventional single-plane, spriteless graphics system. The resolution varies depending on the display mode selected.

Behind the pattern plane is the backdrop plane. Its area is larger than the other planes so that it can form a border around them. The backdrop is always either 1 of 15 solid colors or transparent.

The last, rearmost plane is called the external VDP plane, which can allow one 9918A chip to overlay its display over the output of a second 9918A. But the external VDP plane could contain a video image from almost any compatible external source such as a TV camera, a videotape recorder, or another computer display, as long as the external source is synchronized to the 9918A's Clock and $\overline{\text{Reset}}$/Sync inputs. It might also be necessary to adjust the signal voltage levels.

The four image sources (sprites, pattern plane, backdrop, and external input) can be combined to create a single composite image in the 9918A. In most applications, however, the 9918A's external VDP input is not used, and the image is formed from the pattern, backdrop, and sprite planes.

What Are Sprites?

A sprite is a graphics object of a specified pattern appearing on its plane in a position determined by a single coordinate pair specifying the

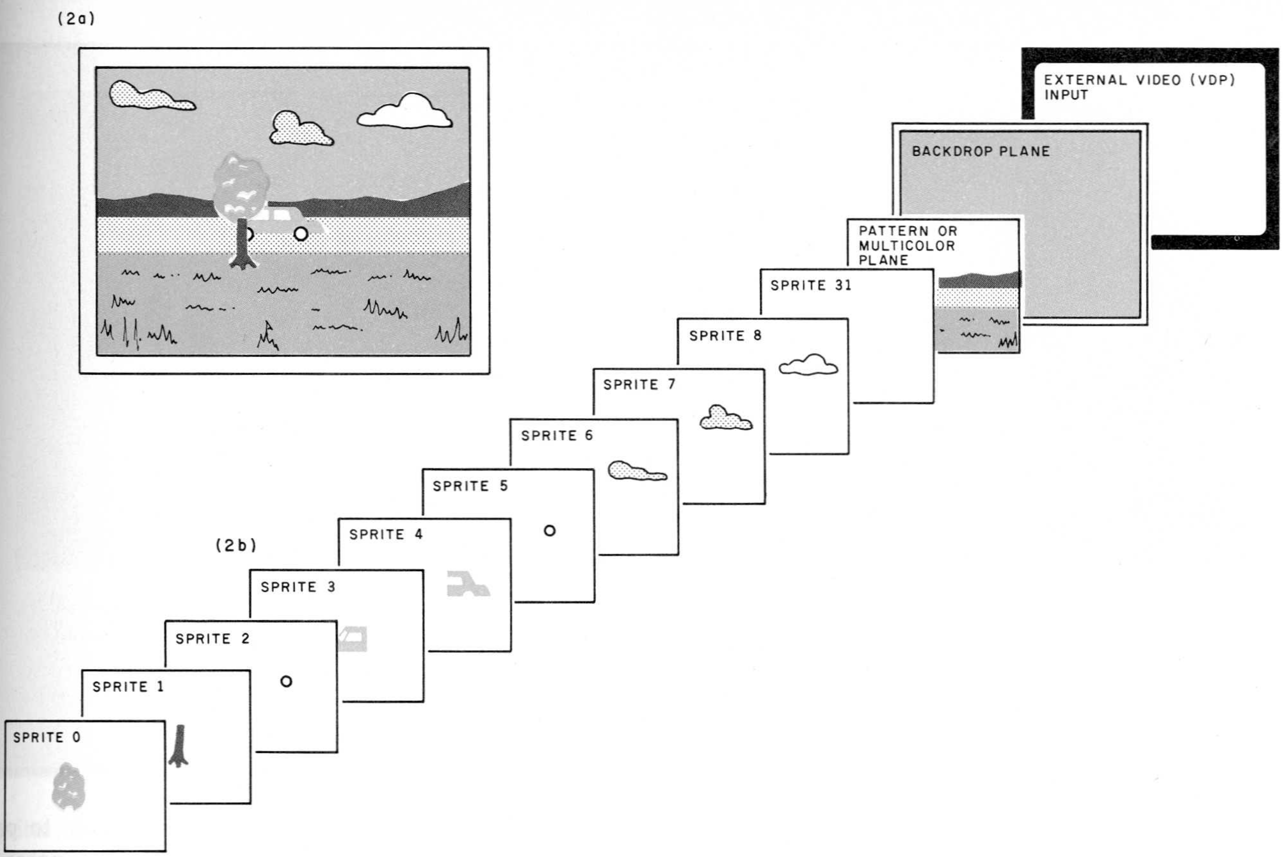


Figure 2: *A possible application for sprites: displaying a graphics image of an automobile driving along a road through hilly country, past a field containing grass and a single tree, under a sky populated by clouds.*

The background, comprising the hills, grass, road, and sky, is "painted" on the pattern plane. Sprites 0 and 1 are set up with patterns representing the tree's foliage and trunk. The sections of the car are drawn using sprites 2 through 5. Finally, three clouds are drawn using sprites 6 through 8. Each of the sprites can be made to move smoothly across the screen by continuously changing a 2-byte address pointer in the sprite-attribute table.

As sprites 2 through 5 (the car sprites) are moved past the position occupied by sprites 0 and 1 (the two tree sprites), the VDP selects the displayed pixel values at each point from the highest-priority plane that is not transparent at that point; therefore our view of the car is automatically blocked out as it passes behind the tree.

sprite's location on the screen in the horizontal and vertical axes. By changing this one set of coordinates, the sprite can be moved easily and quickly across the screen.

Sprites come in two sizes: 8 by 8 pixels (picture elements) and 16 by 16 pixels; they can be expanded to 32 by 32 pixels by using the magnification feature. Their resolution of movement is one pixel on the 192- by 256-pixel viewing area. Each sprite plane contains exactly one sprite; all the plane's area outside the sprite pattern is transparent. The sprite plane with the highest priority is identified as sprite 0, and the one with the lowest priority is sprite 31.

The ease of programming complex graphic displays through the use of sprites is the most significant feature of the TMS9918A.

Example of Sprite Use

Let's consider a possible application: displaying a graphics image of an automobile driving along a road through hilly country, past a field containing grass and a single tree, under a sky populated by clouds (see figure 2). Starting from the foreground, we see that there is a tree between our point of view and the roadway. Naturally we expect the car to be obscured by the tree when passing behind it. And the car should obscure the background hills wherever it goes.

This scene is set up on the 9918A as follows. The background, comprising the hills, grass, road, and sky, is "painted" on the pattern plane in a way similar to the use of any conventional display.

Since the size of the sprites is limited and each sprite can be only one color, it sometimes becomes necessary to use multiple sprites to define a single entity in the picture. (When the entity is to be moved across the screen, all the sprites that form it must be moved at the same time.) So, following this plan, sprites 0 and 1 are set up with patterns representing the tree's foliage and trunk. The sections of the car (front and rear of the body plus the two visible tires) are drawn using sprites 2 through 5. Finally, three clouds (of slightly different colors) are drawn using sprites 6 through 8. Sprite planes 9 through 31 are left transparent.

Animation Comes Easy

Once the static display has been established, we can see why sprites are so useful in animating the display,

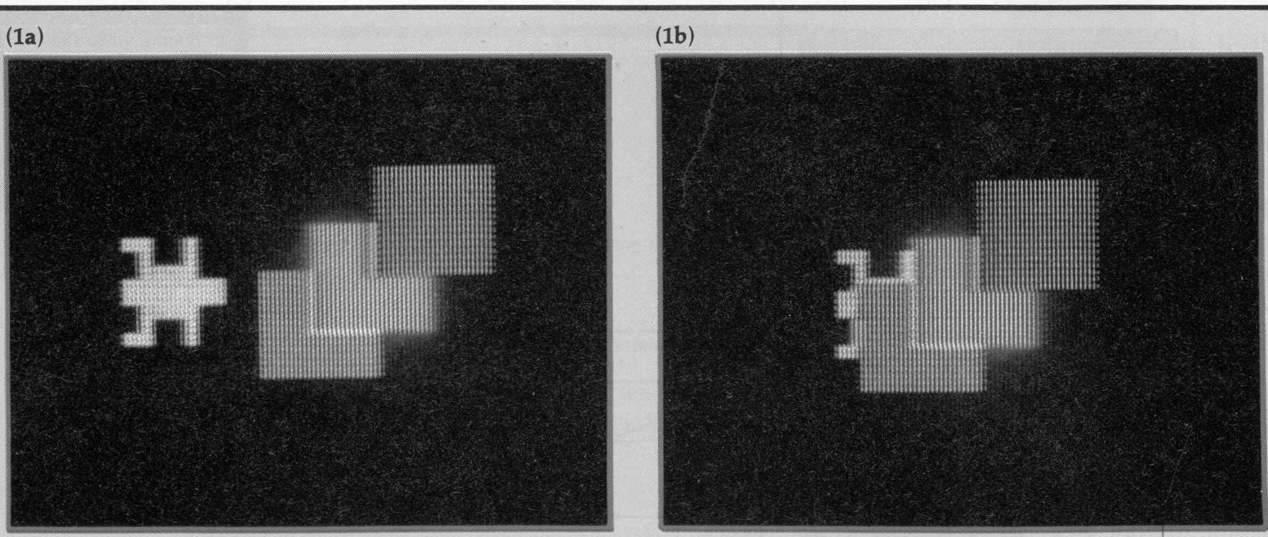


Photo 1: *A step-by-step illustration of the use of sprites and the concept of plane priority. The yellow turtle (sprite 3) is programmed to pass from left to right past the green box (sprite 0), the blue box (sprite 1), and the red box (sprite 2). The transparent pattern plane and backdrop cause the background to be black.*

that is, causing parts of it to move. What would ordinarily be an extensive programming task is handled almost entirely in hardware by the 9918A.

Unlike spriteless systems, moving the car does not require that the software repaint the entire display pattern. Simply by continuously changing a 2-byte address pointer in the sprite-attribute table in VRAM, each of the sprites can be made to move smoothly across the screen.

In addition, as sprites 2 through 5 (the car sprites) are moved past the position occupied by sprites 0 and 1 (the two tree sprites), the VDP selects the displayed pixel values at each point from the highest-priority plane that is not transparent at that point; therefore our view of the car is automatically blocked out as it passes behind the tree. Similarly, if the clouds are different colors (perhaps white and gray) and made to pass each other, they will also appear to pass in front or behind in a pseudo-three-dimensional view. This hidden-view capability is provided in hardware and requires no special software, unlike conventional graphics systems.

Additional Examples

Photo sequences 1 and 2 are step-by-step illustrations of the use of

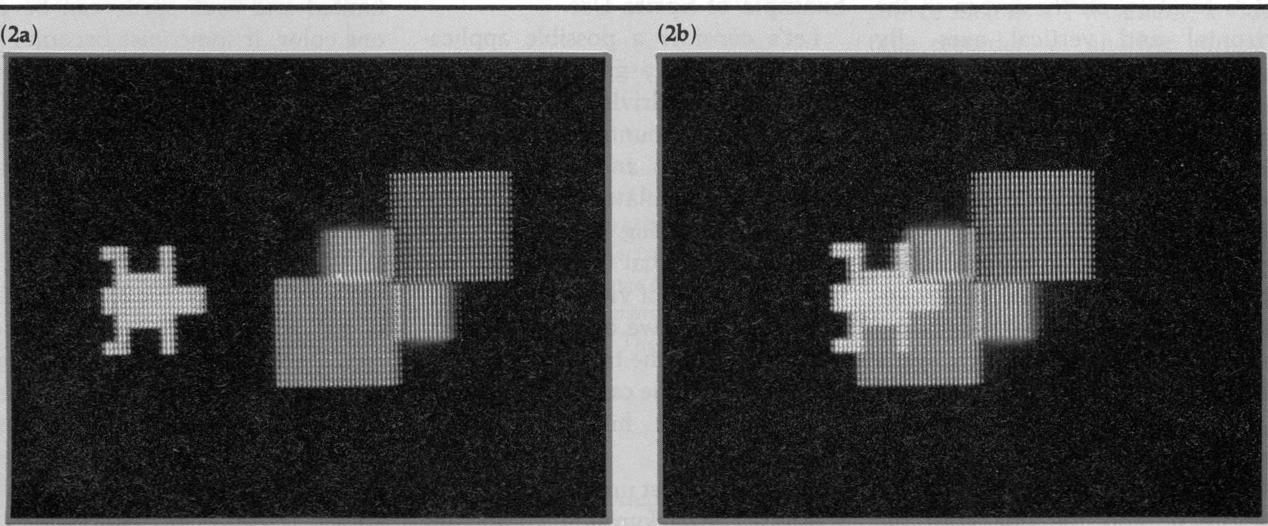
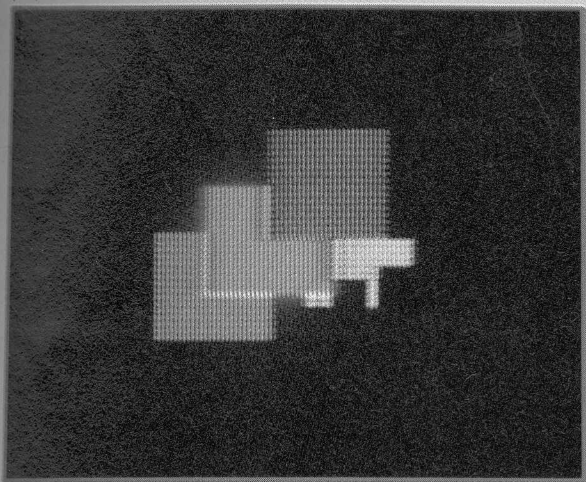
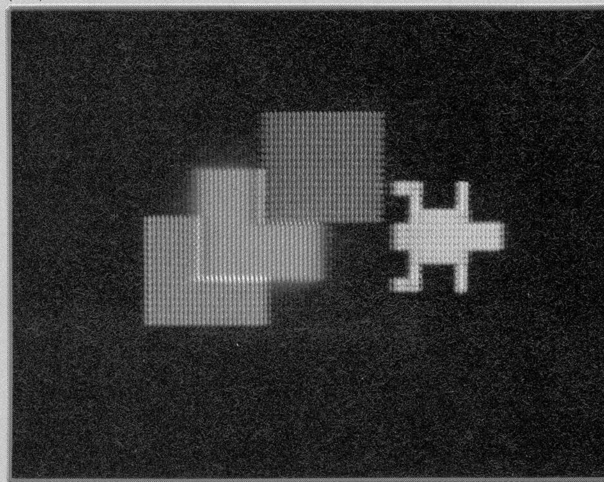


Photo 2: *Some priorities have been exchanged from photo 1: the shapes have been set up on a new permutation of planes. The green and red boxes remain sprites 0 and 2, respectively, but the turtle is now sprite 1 and the blue box is sprite 3. The boxes now overlap in a different order; instead of the sequence green, blue, red, we now have green, red, blue.*

(1c)



(1d)



The turtle is obscured from view as it passes from left to right past the three boxes, beginning in photo 1b. It is not fully visible until it emerges again on the right in photo 1d. Since the three boxes reside on sprite planes of higher priority than the turtle's plane, the pixel values of the boxes take precedence in being displayed wherever the sprite shapes intersect. Also, the three boxes overlap according to their planes' priorities.

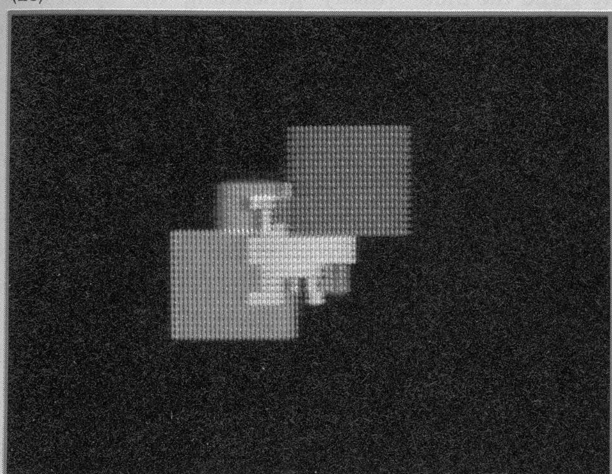
sprites and the concept of plane priority. Both examples use four sprites, but the priorities of the planes used for each sprite shape are changed to demonstrate different effects. Three of the sprites are solid-color boxes, and one is a shape described as a turtle. The turtle is programmed to pass from left to right past the boxes.

In photos 1a through 1d, the green box is sprite 0, the blue box is sprite 1, and the red box is sprite 2. The yellow turtle is sprite 3. No other sprites are involved, and the pattern plane and backdrop are transparent, resulting in a black background.

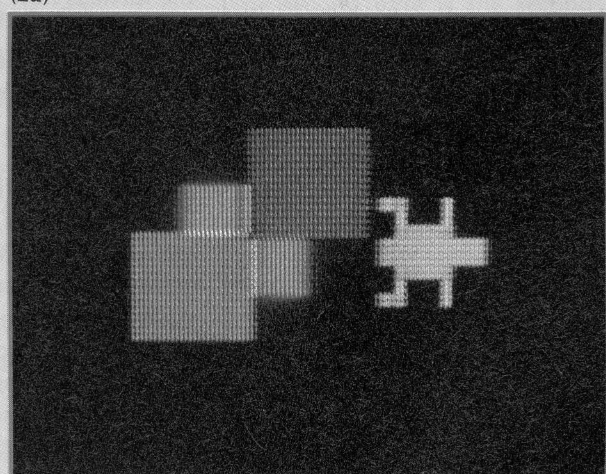
You'll notice that the turtle is obscured from view as it passes from left to right past the three boxes, beginning in photo 1b. Since the three boxes reside on sprite planes of higher priority than the turtle's plane, the pixel values of the boxes take precedence in being displayed wherever the sprite shapes intersect. Observe also that the three boxes overlap according to their planes' priorities. The green covers the blue, and the blue covers the red. As for the turtle, it has the lowest priority and is not fully visible until it emerges again on the right in photo 1d.

In photos 2a through 2d, some priorities are exchanged: the shapes have been set up on a new permutation of planes. The green and red boxes re-

(2c)



(2d)



As the turtle (now sprite 1) passes from left to right, it passes in front of the red box (sprite 2) and the blue box (sprite 3), as shown in photo 2b, but it goes behind the green box (sprite 0), in photo 2c.

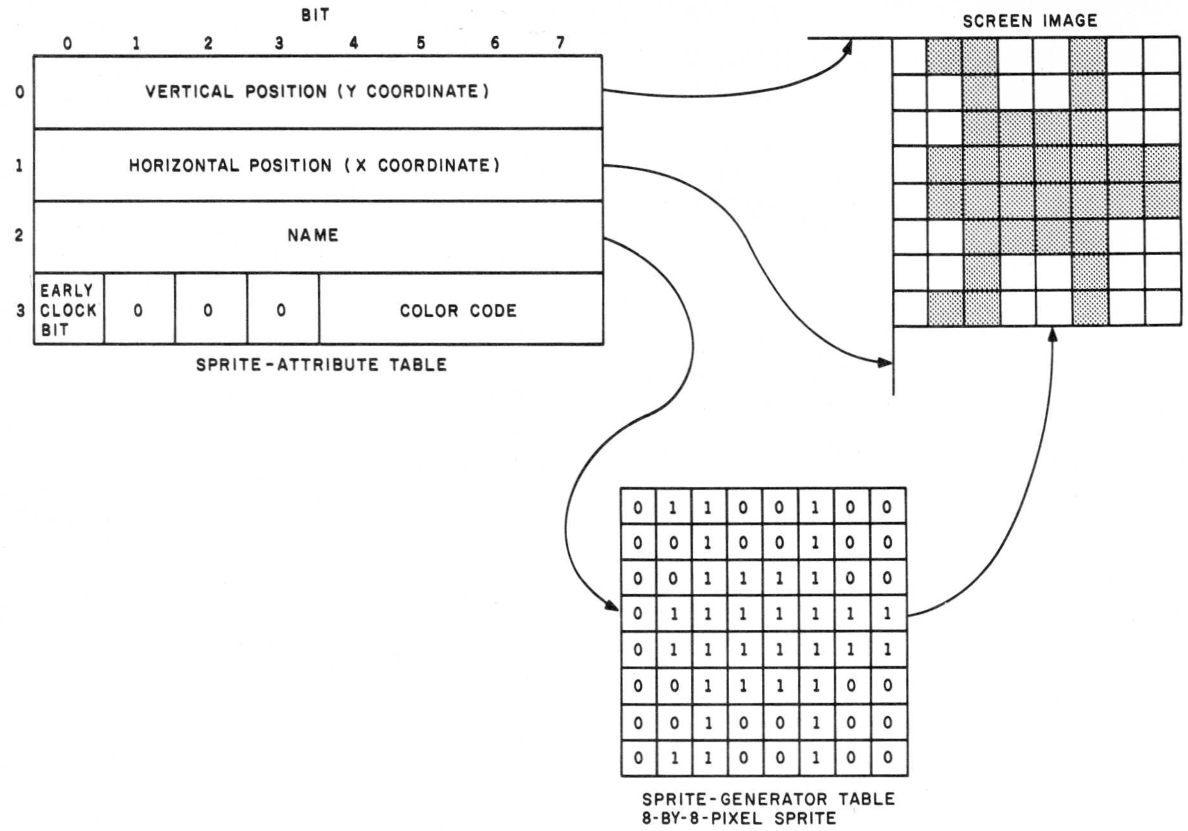


Figure 3: *The binary coding for an 8-by-8-pixel sprite pattern is stored in VRAM in the sprite-generator table in 8 bytes. Each bit in the pattern coding corresponds to one pixel in the displayed pattern. Wherever a 1 is stored in a pixel's pattern bit, the sprite will be colored; where the bit is a 0, the sprite will be transparent. Each sprite can be only a single color.*

Each sprite's attributes are stored in the 128-byte sprite-attribute table. Each set of attributes takes up 4 bytes. In each set of attributes, the first two bytes set the x,y coordinates of the sprite on the screen, referenced from the screen's upper left corner. The third attribute byte contains the sprite's "name" (actually the low-order bits of the address of its segment of the sprite-generator table), and the fourth byte defines the sprite's color, according to the 4-bit color values given in table 2.

Hexadecimal Value	Color
0	transparent
1	black
2	medium green
3	light green
4	dark blue
5	light blue
6	dark red
7	cyan
8	medium red
9	light red
A	dark yellow
B	light yellow
C	dark green
D	magenta
E	gray
F	white

Table 2: *Four-bit binary codes used by the 9918A to specify the color of a picture element or color pattern.*

main sprites 0 and 2, respectively, but the turtle is now sprite 1 and the blue box is sprite 3. The first feature of note is the reordering of the overlapping boxes. Instead of the sequence green, blue, red, we now have green, red, blue.

As the turtle (now sprite 1) passes from left to right, it passes in front of the red box (sprite 2) and the blue box (sprite 3), as shown in photo 2b, but it goes behind the green box (sprite 0), as we see in photo 2c. The appearance is that it is passing among rather than behind the boxes.

Boxes and turtles may not impress you very much in themselves, but remember that no complicated hidden-line algorithms are needed to determine pixel precedence. Everything I've demonstrated is done completely in hardware on the 9918A. The only software computation (other than initially generating the sprites) is to change a 2-byte x,y coordinate pair to move the turtle.

There is a restriction, however, on the number of sprites that may occupy a single horizontal scan line in the video display raster: only four may do so simultaneously. If a fifth sprite is moved into a position such that part of its pattern is on the same line with parts of four other sprites, the conflicting parts of the *lowest priority* sprite of the five will be made transparent on the display. Also, the number of the fifth sprite will appear in the 9918A's status register.

Structure of Sprites

There are two basic sizes of sprites: 8 by 8 pixels and 16 by 16 pixels. The 8- by 8-pixel sprite is more often used;

the binary coding for its pattern is stored in VRAM in the *sprite-generator table* (SGT) in 8 bytes, as shown in figure 3. The larger 16- by 16-pixel sprite requires 32 bytes for storage of its pattern coding.

Each bit in the SGT pattern coding corresponds to one pixel in the displayed pattern. Wherever a 1 is stored in a pixel's pattern bit, the sprite will be colored; where the bit is a 0, the sprite will be transparent. Each sprite can be only a single color.

Either size sprite may be enlarged (magnified) by a factor of 2 under software control; the magnification factor (1 or 2) is global, affecting all sprites. The display produced for the priority demonstration of photo sequences 1 and 2 consisted of 16- by 16-pixel sprite shapes made from 8- by 8-pixel sprites magnified to be twice as big as normal.

Each sprite's attributes (values that determine the characteristics of color, coordinate position, and SGT pattern location) are stored in the *sprite-attribute table*, or SAT, in VRAM. Each set of attributes takes up 4 bytes; to support 32 sprites, the table must be 128 bytes long. To find the storage location of a particular sprite's attributes, we merely take the sprite's number, multiply it by 4, and add the result to the base address of the sprite-attribute table, which is stored in the 9918A's register 5.

In each set of attributes, the first two bytes set the x,y coordinates of the sprite on the screen, referenced from the screen's upper left corner. The third attribute byte contains the sprite's "name" (actually the low-order bits of the address of the sprite's SGT segment), and the fourth byte defines the sprite's color, according to the 4-bit color values given in table 2.

Not Only Sprites

In addition to sprites, the TMS9918A VDP is capable of considerable graphic feats using only the pattern plane, which operates in any of four display modes. Not all modes use the full 16K-byte memory capacity that the 9918A is capable of supporting. The display mode and memory allocation are selected by setting bits in the VDP's registers. Let's look at some of these other methods of display.

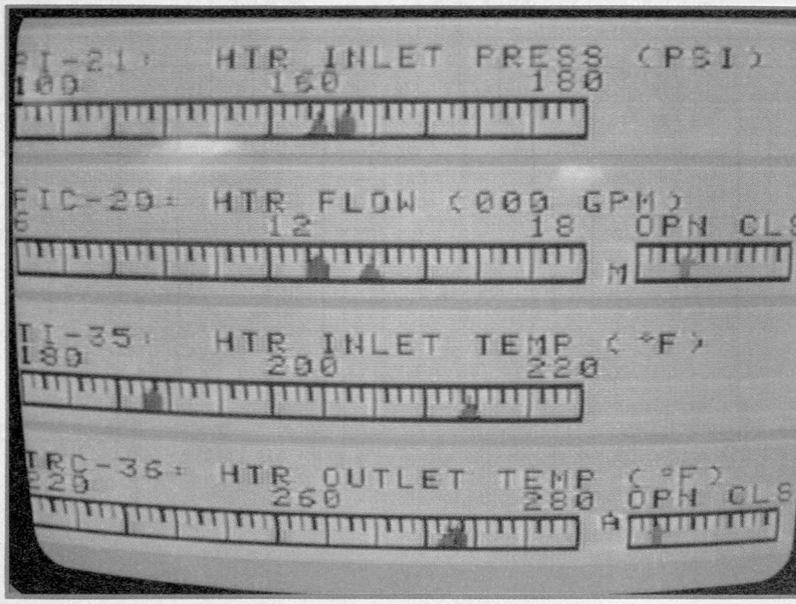


Photo 3: *A Graphics-II-mode display combined with sprites, showing a simulation of some analog sensor meters. The pattern plane contains the meter scales and alphanumeric labeling, while the pointers within the meter scales are sprites, which are easily moved to represent changes in the measured quantities.*

Graphics I Mode

In the Graphics I mode, the screen is divided up into a grid of pattern positions arranged in 24 rows of 32 columns: a total of 768 positions. Each pattern position contains 64 pixels arranged in 8 rows of 8 columns. The contents of the pattern-generator table (PGT) in VRAM determine what is displayed in these pattern positions, and the pattern-color table (PCT) defines the colors associated with them.

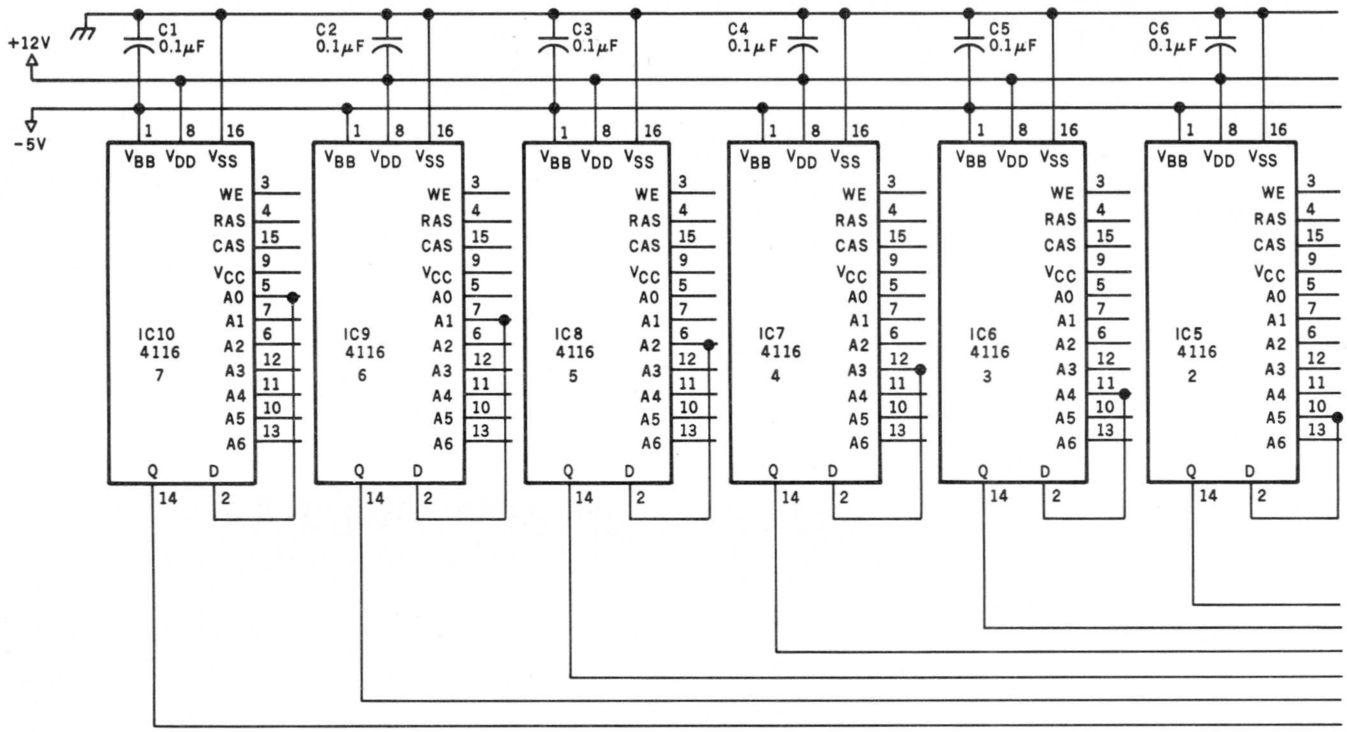
The ease of programming complex graphic displays through use of the sprites is the most significant feature of the TMS9918A.

In Graphics I mode, up to 256 different patterns can be stored; any one of these can be used in any of the 768 pattern positions, and each pattern can contain two of fifteen possible colors. The patterns can be alphanumeric characters or small sections of a large display picture, disassembled as if it were a jigsaw puzzle.

The pattern definition in the pattern-generator table consists of an 8-byte segment of memory; each bit in the segment corresponds to one pixel in the 8 by 8 matrix; the first byte is the top row of the matrix, and the second byte is the second row, etc. The colors to be used in a given pattern are determined by the two 4-bit values stored in the pattern's color byte in the pattern-color table; binary 1s and 0s are set in the pattern-generator table to turn on one color or the other for each pixel in the pattern.

Graphics II Mode

The Graphics II mode is similar to the Graphics I mode except that it allows 768 separate pattern definitions instead of only 256. In addition, instead of only two colors within each 8- by 8-pixel pattern block, Graphics II mode allows two colors to be defined separately for each byte in the pattern block, so potentially sixteen colors could appear in a single



Number	Type	+5V	GND	−5V	+12V
IC1	TMS9918A	33	12		
IC2	74LS00	14	7		
IC3	4116	9	16	1	8
IC4	4116	9	16	1	8
IC5	4116	9	16	1	8
IC6	4116	9	16	1	8
IC7	4116	9	16	1	8
IC8	4116	9	16	1	8
IC9	4116	9	16	1	8
IC10	4116	9	16	1	8

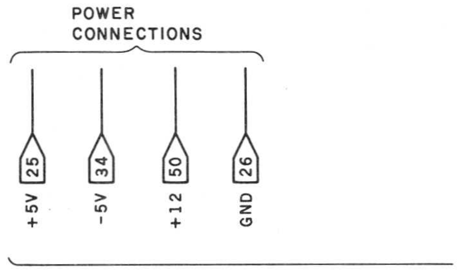


Figure 4: *A schematic diagram of the E-Z Color Graphics Interface. Very few components are needed to connect the TMS9918A to the computer's electrical bus; most of the integrated circuits are simply memory components used as the 9918A's VRAM.*

block. As you might expect, this mode uses more memory, potentially as much as 12K bytes of VRAM.

By allowing 768 distinct patterns for the 768 available pattern locations, the Graphics II mode equals the image capacity of the widely used conventional 256- by 192-pixel displays. Virtually any scene pictured in the Apple II high-resolution graphics mode, for example, can be recreated on the pattern plane of the 9918A. With a little additional application programming to set register pointers and load the pattern and color tables, the Graphics II mode can exactly synthesize the point- and line-plotting functions of conventional graphics interfaces. And you still can use the sprites.

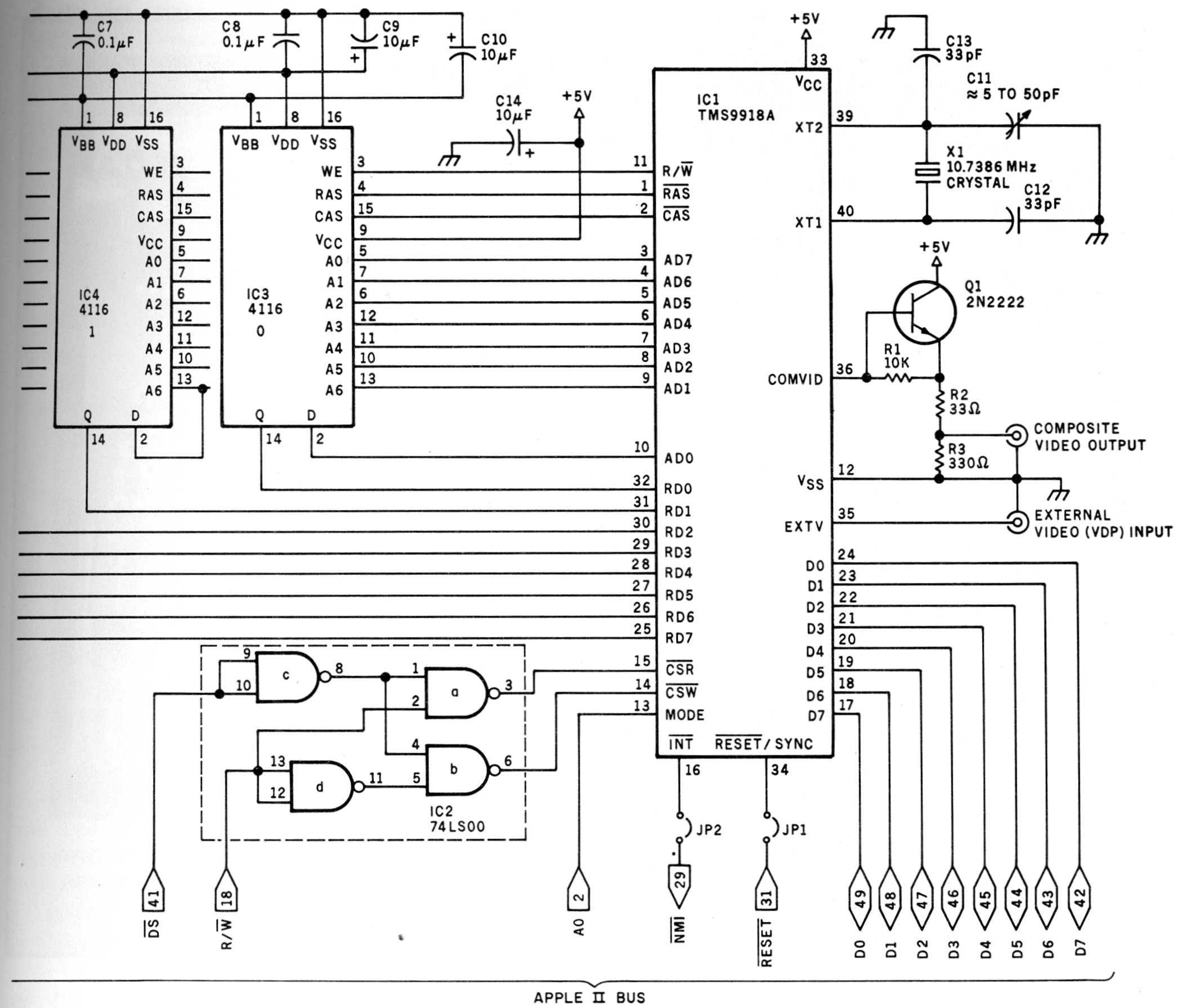
Photo 3 is an example of a Graphics-II-mode display combined with sprites, showing a simulation of some analog sensor meters. The pattern plane contains the meter scales and alphanumeric labeling, while the pointers within the meter scales are sprites, which are easily moved to represent changes in the measured quantities. Since there is no screen rewriting required to move the dial pointers, there is absolutely no flicker, and the pointer placement is an easily calculated x displacement.

Multicolor Mode

The Multicolor mode is essentially a low-resolution graphics mode. In it, the screen is divided into 3072 blocks, each measuring 4 by 4 pixels, in a 48-line by 64-column format. The color of each block can be any of the fifteen colors or transparent.

Text Mode

In the Text mode, the screen is divided into a grid measuring 24 lines by 40 columns of pattern positions,



The circuit shown is intended for use with an Apple II computer, with the circuit board plugged into a slot on the motherboard (usually slot 4), but other versions of the circuit for S-100-bus computers and the IBM Personal Computer are under development. The E-Z Color Graphics Interface may also be adapted for use with other computers.

each of which measures 6 by 8 pixels. The Text mode is intended for display of alphanumeric characters rather than graphics patterns. There can be up to 256 unique character patterns defined at a single time to fill the 960 pattern positions. The sprite planes are not available in Text mode. (If you need both sprites and text simultaneously, you can generate character patterns in the Graphics I mode if you don't mind a slightly shorter line length than in the Text mode.)

The character set is stored in the pattern table in VRAM. Since the cells measure 6 by 8 pixels, the characters should occupy a 5- by 7-pixel format to allow some space between characters. By properly setting the register pointers, it is possible to have the table addresses for the character patterns equal the characters' ASCII (American Standard Code for Information Interchange) values, which makes character generation easy.

Use of Memory

While the 9918A project I built has 16K bytes of VRAM, not all modes use that much. A typical application that uses only two colors with 256 unique 8- by 8-pixel patterns and 32 sprites would take less than 4K bytes of VRAM. By providing 16K bytes of VRAM with the 9918A, I found that I often had room to store four complete displays; the VDP can switch between them by simply changing pointers in the registers.

E-Z Color Graphics Interface

Figure 4 is the schematic diagram of my project for this month, which I call the Circuit Cellar E-Z Color Graphics Interface. The design is a typical 9918A color graphics interface in that it is interfaced to a microcomputer bus with a minimum of compo-

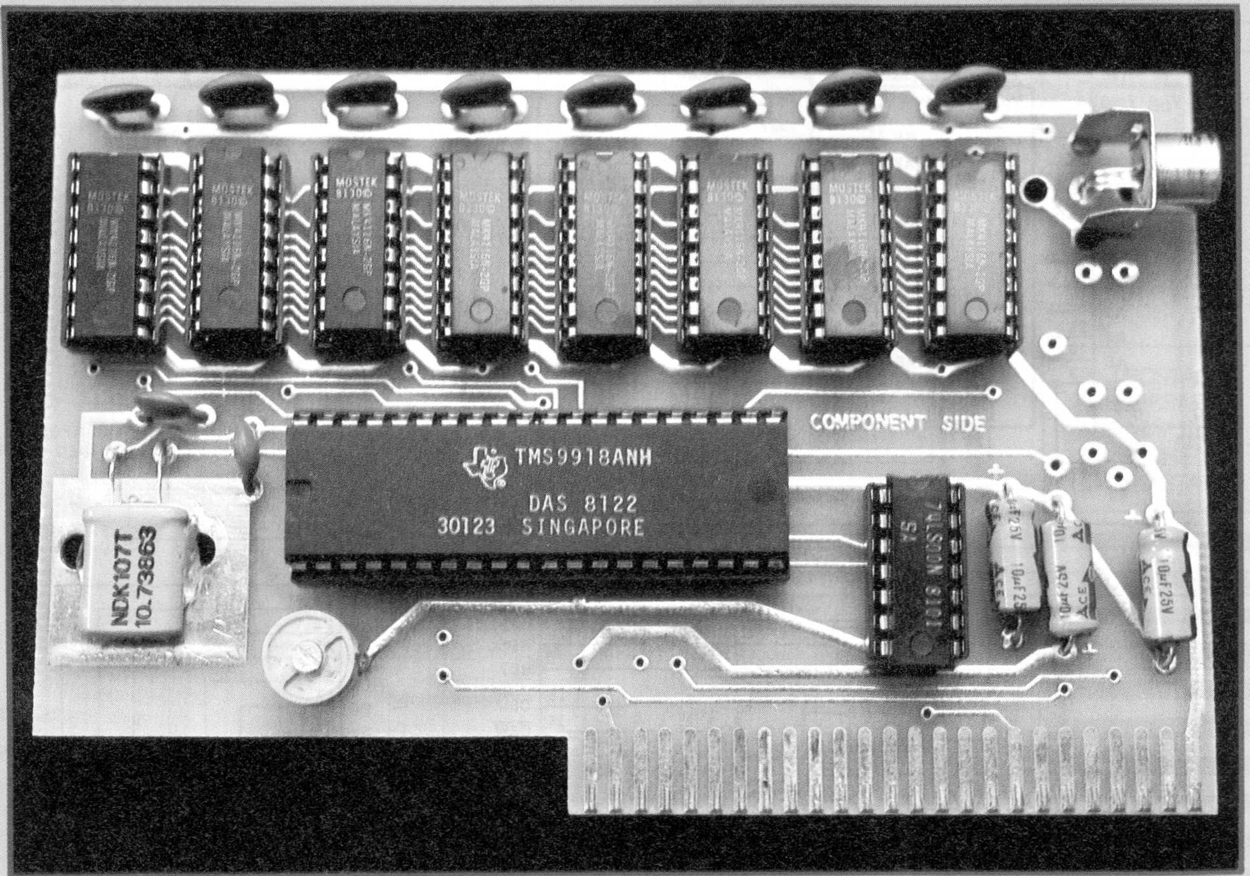


Photo 4: *The Circuit Cellar E-Z Color Graphics Interface; a prototype printed-circuit board is shown. This typical TMS9918A color graphics interface is interfaced to the Apple II microcomputer bus with a minimum of components.*

nents. A prototype printed-circuit board is shown in photo 4.

This particular design has been configured for use with an Apple II, yet its signals are compatible with those used in many other computer systems. If you are willing to add a 40-pin connector and do some hand-wiring, you can use this board with some other kind of microcomputer.

The circuit requires an 8-bit bidirectional data bus, one address line (typically A0), and the two control signals Read Enable ($\overline{\text{CSR}}$) and Write Enable ($\overline{\text{CSW}}$). For operation with the Apple II, these signals are formed by logically combining the Apple's $\overline{\text{DS}}$ (Device Select) and R/$\overline{\text{W}}$ (Read/Write) lines. The two control signals are known by different names in other computer systems, but their functions are generally compatible. Two additional lines, INT (Interrupt) and $\overline{\text{Reset}}$/Sync, are shown as jumper connections. They are available for various optional enhancements, such as interrupt-driven animation or synchronization with external video sources.

By the time you read this article, I shall have completed the designs for S-100-bus and IBM Personal Computer versions of the E-Z Color interface. Check with the parts source given at the end of the article for availability.

Assembly-Language Sprite Use

As I alluded before, the 9918A is initialized by loading values into control bits and address pointers in eight write-only registers. Drawing and moving sprites across the screen is simply a matter of choosing the proper register parameters and changing the pointers.

Listing 1 on page 68 is a program that demonstrates the routines needed to display and move sprites. The program is written in 6502 assembly language to run on an Apple II computer equipped with the E-Z Color Graphics Interface, installed in motherboard slot 4 at hexadecimal address C0C0.

The first requirement is to initialize the eight registers and clear the VRAM. In this example the 9918A is set to the following operating specifications: Graphics II mode, external video input disabled, and 16- by 16-pixel sprites, with selectable magnification to twice the normal size (32 by 32 pixels) under keyboard control.

When the program starts, four different sprites are displayed, as shown in photo 5. You can change the display as follows. When you press the M key, the sprites' position coordinates are incremented and the sprites move. Pressing the O key and then a hexadecimal digit 1 through F will set one of the fifteen background colors or transparency (shown). Pressing the left- or right-arrow keys will vary the sprites' size between 16 by 16 and 32 by 32 pixels.

If you are ambitious, one possible exercise is to add more sprites to this program. Photo 6 shows how complicated things get when we have 24 sprites.

Logo Sprite Use

If you don't care to concern yourself with the intricacies of assembly language, you may choose to use routines written in Terrapin's version of MIT Logo to control the E-Z Color graphics.

Terrapin Logo normally uses a single video monitor for all its display functions: text listings and line drawing. The colors available are limited to the six supported by the Apple's high-resolution graphics mode. When the E-Z Color Graphics Interface is installed, the regular display screen is still used for text display and the regular turtle graphics; the E-Z Color board must be connected to a second color video monitor for its display to be simultaneously visible. Photo 7 on page 68 shows the two-monitor setup. (If you don't need to see both displays at once, you could set up a switch to select the video output of one source or the other for display on a single monitor.)

The Logo procedures developed by Leigh and Pat implement user commands to specify the characteristics of each sprite; these commands include SETSHAPE, SETCOLOR, and SXY (for "set x,y position"). If you like, you can map out your own sprite shapes and incorporate them into the routines, but some predefined patterns, shown in photo 8, are provided. (People from Terrapin seem to like turtle shapes.)

The photo sequences 1 and 2 used earlier to demonstrate sprite planes were done using a Logo program. For example, the three boxes (shown in photo 9) are drawn in Logo using the following groups of simple statements:

Text continued on page 80

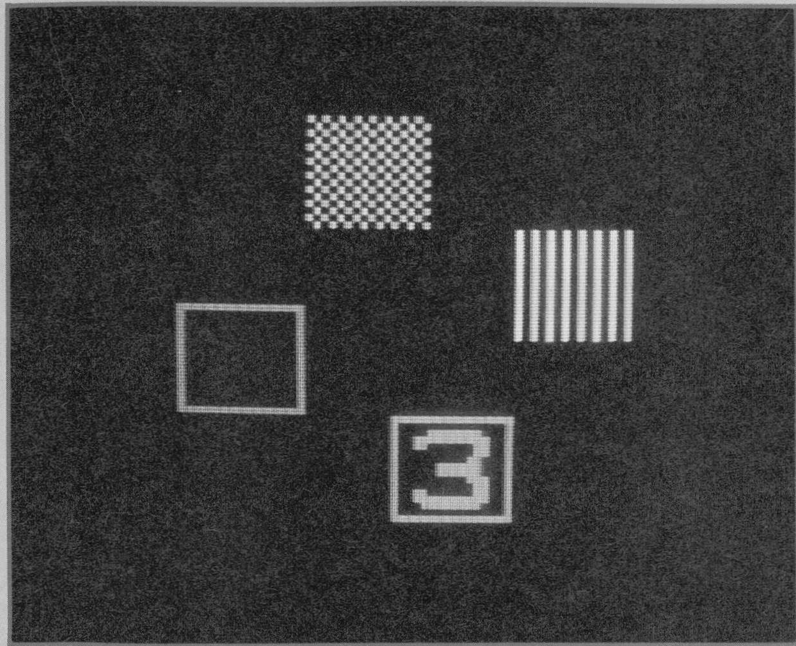


Photo 5: *Display of four sprites produced by the 6502 assembly-language program of listing 1. The user can change the display in the following ways. Pressing the M key causes the sprites to move. Pressing the O key and then a hexadecimal digit 1 through F sets one of the fifteen background colors or transparency (shown). Pressing the left- or right-arrow keys varies the sprites' size between 16 by 16 and 32 by 32 pixels.*

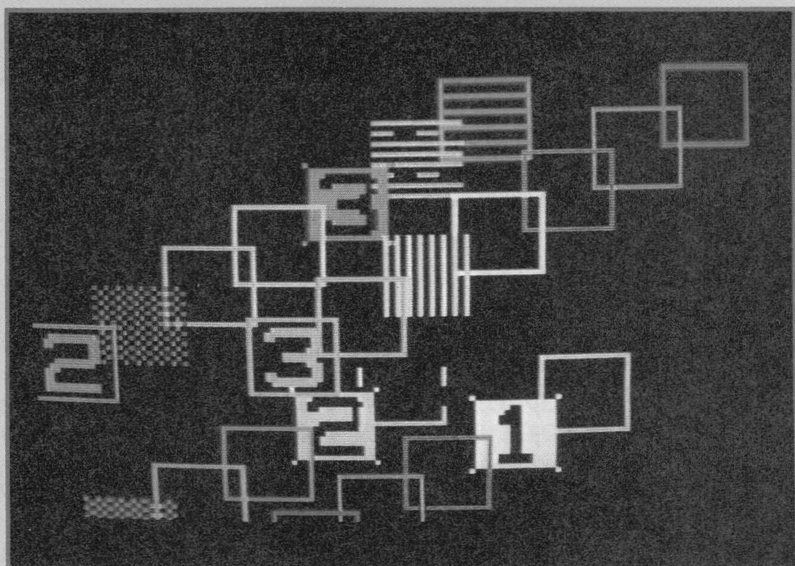


Photo 6: *The display can get complicated when 24 sprites are visible.*

Photo 7: *When the E-Z Color Graphics Interface is installed in the Apple II, the regular display screen is still used for Terrapin MIT Logo's text display and turtle graphics; the E-Z Color board must be connected to a second color video monitor for its display to be simultaneously visible.*

Listing 1: *Program written in 6502 assembly language to run on an Apple II computer equipped with an E-Z Color Graphics Interface installed in motherboard slot 4.*

```
LOC  CODE    LINE
0000         ;
0000         ;*************************************************************
0000         ;
0000         ;                    ***  VIDEO DEMO  ***
0000         ;
0000         ;
0000         SLOT    =       $40                 ;SLOT = NO. X 10 HEX
0000         KBD     =       $C000               ;APPLE KEYBOARD DATA
0000         KSTRB   =       $C010               ;KEYBOARD DATA CLEAR
0000         VREG    =       $C081+SLOT          ;VDP REGISTER
0000         VDATA   =       $C080+SLOT          ;VDP RAM
0000         ;
0000                 *=       $1000               ;PROGRAM STARTING ADDRESS
1000         ;
1000         ;***************  INITIALIZE VDG  *************************
1000         ;
1000 A087            LDY #$87                    ;REGISTER SELECT
1002 A207            LDX #$07                    ;INITIALIZE COUNTER
1004 BDC610  INIT1   LDA ITAB,X                  ;LOAD INIT TABLE
```

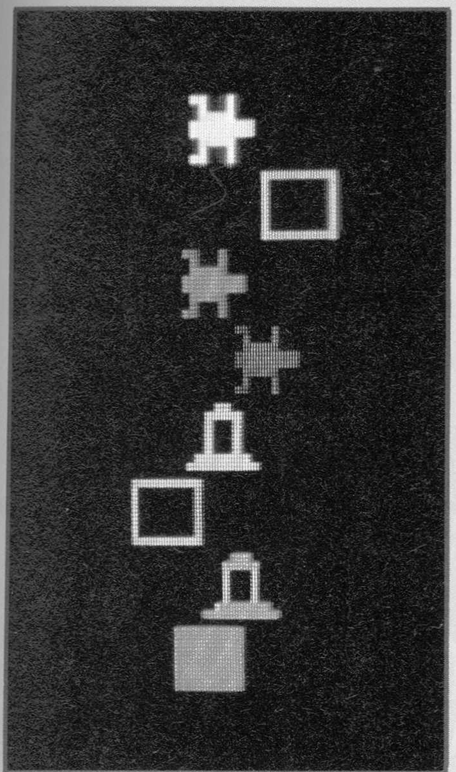


Photo 8: *The Logo procedures developed at Terrapin Inc. provide you with commands such as SETSHAPE, SETCOLOR, and SXY. You can map out your own sprite shapes and incorporate them into the routines, but some predefined patterns are provided, including a box, a rocket, a turtle, and a block.*

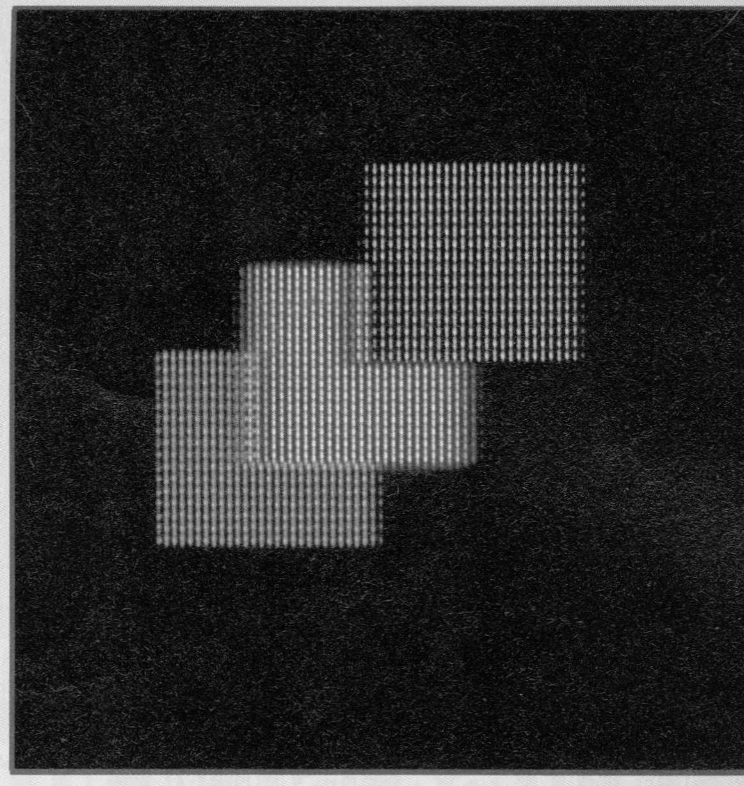


Photo 9: *Each of the three boxes is drawn and placed in position with only four Logo statements.*

Listing 1 continued:

```
1007 209F10         JSR SREG                    ;WRITE TO VDP
100A 88             DEY                         ;DECREMENT REGISTER
100B CA             DEX                         ;DECREMENT COUNTER
100C D0F6           BNE INIT1                   ;DONE?
100E            ;
100E            ;**************  CLEAR ALL MEMORY  *************************
100E            ;
100E A040           LDY #$40                    ;BYTE2 ADDRESS SET UP
1010 A900           LDA #$00                    ;BYTE1 ADDRESS SET UP
1012 209F10         JSR SREG                    ;WRITE TO VDP
1015 A2C0           LDX #$C0                    ;COUNTER HIGH BYTE
1017 A000   NEXF    LDY #$00                    ;COUNTER LOW BYTE
1019 8DC0C0 FILL    STA VDATA                   ;WRITE TO VDP RAM
101C C8             INY                         ;INCREMENT LOW COUNTER
101D D0FA           BNE FILL                    ;LOW COUNTER FULL?
101F E8             INX                         ;INCREMENT HIGH COUNTER
1020 D0F5           BNE NEXF                    ;HIGH COUNTER FULL?
1022            ;
1022            ;********  LOAD SPRITE ATTRIBUTES  *************************
1022            ;
1022 A047   LOOP    LDY #$47                    ;BYTE2 AT 0700 HEX
1024 A900           LDA #$00                    ;BYTE1 ADDRESS SET UP
1026 209F10         JSR SREG                    ;WRITE TO VDP
1029 A200           LDX #$00                    ;INITIALIZE COUNTER
```

Listing 1 continued on page 70

Listing 1 continued:

```
LOC  CODE    LINE

102B BDCE10 NEXA   LDA ATAB,X          ;LOAD ATTRIBUTE
102E 8DC0C0        STA VDATA           ;STORE TO VDP RAM
1031 E8            INX                 ;INCREMENT COUNTER
1032 8A            TXA
1033 C910          CMP #$10            ;TEST COUNTER
1035 D0F4          BNE NEXA            ;DONE?
1037               ;
1037               ;**********  LOAD SPRITE PATTERNS  ************************
1037               ;
1037 A040          LDY #$40            ;BYTE2 AT 0000 HEX
1039 A900          LDA #$00            ;BYTE1 ADDRESS SET UP
103B 209F10        JSR SREG            ;WRITE TO VDP
103E A200          LDX #$00            ;INITIALIZE COUNTER
1040 BDDE10 NEXTS  LDA PTAB,X          ;LOAD PATTERN BYTE
1043 8DC0C0        STA VDATA           ;STORE TO VDP RAM
1046 E8            INX                 ;INCREMENT COUNTER
1047 8A            TXA
1048 C980          CMP #$80            ;TEST COUNTER
104A D0F4          BNE NEXTS           ;DONE?
104C               ;
104C               ;*************  CHANGE BACKGROUND  ************************
104C               ;
104C AD00C0 CBACK  LDA KBD             ;TEST FOR
104F C9CF          CMP #$CF            ;"O" KEY INPUT
1051 D008          BNE CSIZE           ;TO SET BACKGROUND COLOR
1053 20A610        JSR LOADN           ;READ KEYBOARD
1056 A087          LDY #$87            ;BYTE1 REGISTER 7
1058 209F10        JSR SREG            ;STORE TO VDP
105B               ;
105B               ;*******************  CHANGE SIZE  ************************
105B               ;
105B AD00C0 CSIZE  LDA KBD             ;TEST FOR LEFT ARROW
105E C988          CMP #$88            ;MAGNIFICATION X 1
1060 D00A          BNE ONE
1062 ADC710        LDA ITAB+1          ;LOAD REGISTER 1
1065 29FE          AND #$FE            ;MASK 0 ON LSB
1067 A081          LDY #$81            ;BYTE1 REGISTER 1
1069 209F10        JSR SREG            ;STORE TO VDP
106C C995   ONE    CMP #$95            ;TEST FOR RIGHT ARROW
106E D00A          BNE MOVE            ;MAGNIFICATION X 2
1070 ADC710        LDA ITAB+1          ;LOAD REGISTER 1
1073 0901          ORA #$01            ;MASK 1 ON LSB
1075 A081          LDY #$81            ;BYTE1 REGISTER 1
1077 209F10        JSR SREG            ;STORE TO VDP
107A               ;
107A               ;******************  MOVE SPRITES  ************************
107A               ;
107A AD00C0 MOVE   LDA KBD             ;MOVE?
107D C9CD          CMP #$CD            ;TEST FOR "M" KEY
107F D018          BNE JUMP
1081 EECE10        INC ATAB            ;SPRITE0 UP
1084 CECF10        DEC ATAB+1          ;SPRITE0 LEFT
1087 EED210        INC ATAB+4          ;SPRITE1 UP
108A EED310        INC ATAB+5          ;SPRITE1 RIGHT
108D CED610        DEC ATAB+8          ;SPRITE2 DOWN
1090 CED710        DEC ATAB+9          ;SPRITE2 LEFT
1093 CEDA10        DEC ATAB+$C         ;SPRITE3 DOWN
1096 EEDB10        INC ATAB+$D         ;SPRITE3 RIGHT
1099 2C10C0 JUMP   BIT KSTRB           ;CLEAR KEYBOARD
```

Listing 1 continued on page 72

Listing 1 continued:

```
LINE# LOC CODE   LINE

0104 109C 4C2210         JMP LOOP                     ;JUMP TO START
0105 109F               ;
0106 109F               ;
0107 109F               ;*********  STORE VIDEO REGISTERS  ************************
0108 109F               ;
0109 109F 8DC1C0 SREG    STA VREG                     ;STORE BYTE1
0110 10A2 8CC1C0         STY VREG                     ;STORE BYTE2
0111 10A5 60             RTS                          ;RETURN
0112 10A6               ;
0113 10A6               ;*************  LOAD KEYBOARD INPUT  **********************
0114 10A6               ;
0115 10A6 2C10C0 LOADN   BIT KSTRB                    ;CLEAR KEYBOARD
0116 10A9 2C00C0 WAIT    BIT KBD                      ;TEST KEYBOARD
0117 10AC 10FB           BPL WAIT                     ;IS KEY PRESSED ?
0118 10AE AD00C0         LDA KBD
0119 10B1 29F0           AND #$F0                     ;TEST IF NUMERICAL INPUT
0120 10B3 C9C0           CMP #$C0
0121 10B5 F006           BEQ LETER
0122 10B7 AD00C0         LDA KBD
0123 10BA 290F           AND #$0F                     ;MASK OFF HIGH NIBBLE
0124 10BC 60             RTS                          ;RETURN
0125 10BD AD00C0 LETER   LDA KBD
0126 10C0 18             CLC
0127 10C1 6909           ADC #$09                     ;CQNVERT INPUT TO HEX VALUE
0128 10C3 290F           AND #$0F                     ;MASK OFF HIGH NIBBLE
0129 10C5 60             RTS                          ;RETURN
0130 10C6               ;
0131 10C6               ;***********************  TABLES  ***
0132 10C6               ;
0133 10C6 02     ITAB    .BYT $02,$C2,$01,$80         ;INITIALIZE TABLE
0133 10C7 C2
0133 10C8 01
0133 10C9 80
0134 10CA 01             .BYT $01,$0E,$00,$01
0134 10CB 0E
0134 10CC 00
0134 10CD 01
0135 10CE               ;
0136 10CE 40     ATAB    .BYT $40,$60,$00,$03         ;SPRITE 0 ATTRIBUTE
0136 10CF 60
0136 10D0 00
0136 10D1 03
0137 10D2 60             .BYT $60,$60,$04,$07         ;SPRITE 1 ATTRIBUTE
0137 10D3 60
0137 10D4 04
0137 10D5 07
0138 10D6 40             .BYT $40,$80,$08,$0B         ;SPRITE 2 ATTRIBUTE
0138 10D7 80
0138 10D8 08
0138 10D9 0B
0139 10DA 60             .BYT $60,$80,$0C,$0F         ;SPRITE 3 ATTRIBUTE
0139 10DB 80
0139 10DC 0C
0139 10DD 0F
0140 10DE               ;
0141 10DE FF80   PTAB    .DBY $FF80,$8080,$8080,$8080      ;SPRITE 0 PATTERN
0141 10E0 8080
0141 10E2 8080
0141 10E4 8080
```

Listing 1 continued on page 76

Listing 1 continued:

```
LINE# LOC  CODE    LINE

0159 1156 AAAA            .DBY $AAAA,$AAAA,$AAAA,$AAAA
0159 1158 AAAA
0159 115A AAAA
0159 115C AAAA
0160 115E          ;
0161 115E                 .END

ERRORS = 0000 <0000>


SYMBOL TABLE

SYMBOL   VALUE

ATAB   10CE  CBACK  104C  CSIZE  105B
FILL   1019  INIT1  1004  ITAB   10C6
JUMP   1099  KBD    C000  KSTRB  C010
LETER  10BD  LOADN  10A6  LOOP   1022
MOVE   107A  NEXA   102B  NEXF   1017
NEXTS  1040  ONE    106C  PTAB   10DE
SLOT   0040  SREG   109F  VDATA  C0C0
VREG   C0C1  WAIT   10A9
END OF ASSEMBLY
```

Text continued from page 67:

```
TELL 0
SETSHAPE :BOX
SETCOLOR :GREEN
SXY 20 20
```

The first command specifies that sprite 0 is being addressed. The second tells Logo to use the predefined box pattern, while the third says that the sprite is to be colored green (remember, the rest of the sprite plane will be transparent). Then the fourth command states that the sprite is to be drawn at coordinate 20,20.

Now, to add the blue box as sprite 1 at *x,y* coordinates 12,12.

```
TELL 1
SETSHAPE :BOX
SETCOLOR :BLUE
SXY 12 12
```

Finally, to draw the red box as sprite 2 at position 5,5.

```
TELL 2
SETSHAPE :BOX
SETCOLOR :RED
SXY 5 5
```

A turtle can be drawn simply by using a similar procedure substituting the command SETSHAPE :TURTLE.

At this writing, Terrapin MIT Logo does not support turtle velocity (automatic constant movement actuated by the commands SETSPEED and SETHEADING) as does the Logo package available for the Texas Instruments TI 99/4A microcomputer, but a future version of Terrapin's product may do so.

In Conclusion

The TMS9918A Video Display Processor has many more capabilities than I have room to write about here, and my examples of a few boxes and turtles are an inadequate demonstration of the powerful combination of the E-Z Color Graphics Interface and Terrapin MIT Logo. I am certain that you can fully appreciate them only by observing a dynamic display and seeing how few commands are needed to create it.

I don't usually get excited over mega-bit-width processors or super-high-level languages. What does excite me, however, is taking one of my projects hot off the soldering iron and seeing it operate so easily in synergism with someone else's work. After seeing the graceful mating of the E-Z Color Graphics Interface with Terrapin MIT Logo, I can't help but be excited about other sprite-graphics applications.

Next Month:

Build the MicroVox text-to-speech voice synthesizer. ■

References

1. Guttag, Karl and John Hayn. "Video Display Processor Simulates Three Dimensions," *Electronics*, November 20, 1980, page 123.
2. Nelson, Harold. "Logo for Personal Computers," BYTE, June 1981, page 36.
3. *TMS9918A Video Display Processor.* Houston, TX: Texas Instruments Semiconductor Group, 1981.

Editor's Note: *Steve often refers to previous Circuit Cellar articles as reference material for each month's current article. Most of these past articles are available in reprint books from BYTE Books, 70 Main St., Peterborough, NH 03458.* Ciarcia's Circuit Cellar, Volume I, *covers articles that appeared in BYTE from September 1977 through November 1978.* Ciarcia's Circuit Cellar, Volume II, *contains articles from December 1978 through June 1980.* Ciarcia's Circuit Cellar, Volume III, *contains the articles that were published from July 1980 through December 1981.*

To receive a complete list of Ciarcia's Circuit Cellar project kits available from the Micromint, circle 100 on the reader service inquiry card at the back of the magazine.

Parts Source

The following products are available from:

The Micromint Inc.
917 Midway
Woodmere, NY 11598
telephone: (516) 374-6793
(for technical data)
(800) 645-3479
(orders only)

Apple II plug-compatible E-Z Color Graphics Interface, provided with user manual, sample programs, and TMS9918A reference manual.

Assembled and tested $175
Kit $150

Terrapin MIT Logo for the Apple II; requires 48K-byte user memory and one floppy-disk drive.

On DOS version 3.3 disk. . Call for price

S-100-bus and IBM Personal Computer versions of the E-Z Color Graphics Interface are planned. Call for price and availability.

Prices include shipping in the U.S. Foreign orders add $8 for shipping. Residents of the state of New York please add 7% sales tax.