BUILD THE GT180 COLOR GRAPHICS BOARD

PART 1: BASIC TECHNOLOGY

BY STEVE CIARCIA

Achieve PGA *resolution on the* SB180 *at a fraction of the cost*

Graphics has more of a direct influence on user satisfaction than any other aspect of a computer. For the most part, CP/M and ZCPR3 users have been excluded from a club whose ranks are swelled by owners of IBM PCs, Ataris, and Apples. With the GT180, I will endeavor to rectify this inequity and present a high-performance graphics subsystem for the SB180 that surpasses the graphics capabilities of most presently available computers at significantly less cost. Borland International has helped supply utilities that provide the basis for advanced graphics software development.

The GT180 (graphics technology for the SB180) is both an expansion board and an intelligent graphics system. As a plug-in peripheral to the SB180 XBUS, it adds Professional Graphics Adapter (PGA)–type 640- by 480-pixel color graphics capability to the SB180. When mated with the SB180, the two-board set defines a low-cost SCSI/RS-232C ported graphics terminal for any computer system (see photo 1).

Those who use the GT180 as a general-purpose workstation have the ability to add modern graphics while retaining compatibility with existing Z-System and CP/M application software. The GT180 can also serve as an embedded graphics engine for stand-alone applications like a graphics/videotext terminal, a presentation graphics system, or image processing. Finally, the graphics subsystem design core can easily be ported for application in non-SB180 systems like IBM PC, VME, Multibus, and S-100. In this case, the GT180 serves as a low-cost development tool for prototyping and bootstrapping the ported design.

GT180 OVERVIEW

The GT180 graphics specifications are compared to those of the IBM PGA and Enhanced Graphics Adapter (EGA), the Macintosh, Commodore's Amiga, and the Atari 520ST in table 1.

The key to the design is a recently announced CMOS VLSI graphics processor, the Hitachi HD63484 ACRTC (advanced CRT controller), and two companion chips: GMIC (graphic memory interface controller) and GVAC (graphic video attributes controller). These are supplemented by a highly integrated CMOS Brooktree BT450 palette D/A converter; a 512K-byte frame buffer that can hold three screens of data as well

(continued)

Steve Ciarcia (pronounced "see-ARE-see-ah") is an electronics engineer and computer consultant with experience in process control, digital design, nuclear instrumentation, and product development. The author of several books on electronics, he can be reached at P.O. Box 582, *Glastonbury,* CT 06033.

Table 1: *The* GT180's *graphics specifications are compared with those of other popular microcomputers.*

	GT180 (TTL)	GT180 (Analog)	Apple Macintosh	Atari 520ST	Commodore Amiga	IBM CGA	IBM EGA	IBM PGA
Resolution	640×400	640×480	512×350	640×200*	640×400	640×200†	640×350	640×480
Colors	16	16/4096	1	16/64	16/4096	2	16/64	256/4096
Digital/Analog	digital	analog	digital	digital	analog	digital	digital	analog
Scan Mode	noninterlaced	noninterlaced	noninterlaced	noninterlaced	interlaced	noninterlaced	noninterlaced	noninterlaced
Graphics Coprocessor	yes	yes	no	no	yes	no	no	yes

*Atari 520ST is 320×200 with 16 colors or 640×200 with 4 colors or 640×400 monochrome.
†IBM CGA is 640×200 with 2 colors or 320×200 with 4 colors or 160×100 with 16 colors.

as a library of graphics objects like windows, fonts, or icons; and an IBM PC-compatible keyboard connector (see figure 1).

The GT180 display resolution is 640 by 480 pixels with 16 of 4096 colors. It can also be set to 640 by 400, 640 by 350, 640 by 200, 320 by 200, and other resolutions by changing the initialization parameters (the lower resolutions also require changing the crystal frequency). The GT180 has both TTL RGBI (red-green-blue-intensity) and analog RGB outputs and can directly connect to a number of standard CRT monitors, for example, the Princeton Graphics SR-12 and SR-12P (see photo 2).

The GT180 is a big project that can't be easily explained in a few pages. As with any sophisticated computer peripheral these days, the end product is a combination of hardware and software. So that you can recognize the dividing line between these two camps and understand why I have built the GT180 as I have, I will begin with a description of basic graphics technology. This will also help those who may need a refresher course.

The GT180 project is spread over three months. Part 1 is devoted to the basic technology and understanding the hardware/software dividing line of a graphics peripheral. Part 2 presents the hardware and details the individual VLSI components. Part 3 gives an explanation of the GT180 Graphix Toolbox written by Borland International and also demonstrates its application using Borland's CP/M-compatible Modula-2.

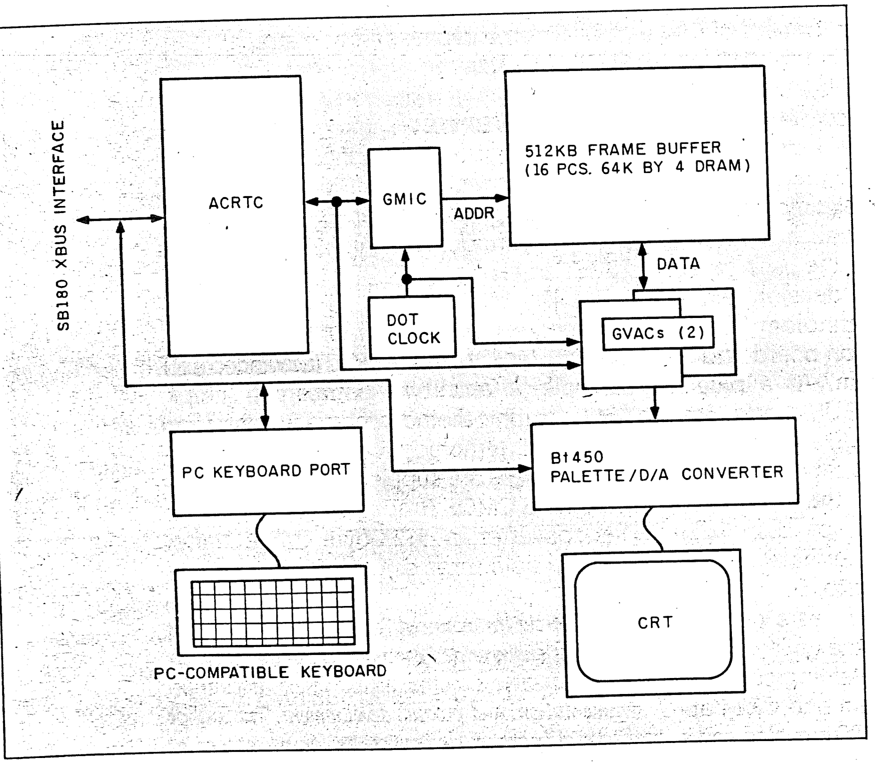


Figure 1: *A block diagram of the* GT180.

CRT Basics

While other display technology comes and goes, the dominant device remains the standard CRT. The underlying technology has been driven by one of the most popular products of all time: the TV set. Many have predicted the end of this glass dinosaur at the hands of other technologies (LCD, plasma, electroluminescence, etc.), but reports of the CRT's demise may be exaggerated. (I tend to use CRT to describe both the glass display tube and the entire terminal display unit with driver electronics.)

The basic principle of CRT operation is simple: An electron beam scans the CRT screen, which is coated with a phosphor. Where the beam hits the phosphor, light is generated. By varying the intensity of the beam, the amount of light generated changes accordingly. In the simplest case (monochrome), the beam is either on or off—each point on the screen is either illuminated or not.

The scanning pattern of the beam

is similar to the way you write with a pencil on a piece of paper. Starting at the top left corner, the beam scans to the right edge of the screen at which point the beam is brought back to the left edge of the screen, one line down from the top. This left-to-right scanning proceeds down the screen until the beam reaches the bottom right corner, then the beam is repositioned at the top left corner and the whole process repeats.

The process of repositioning the beam is called *retrace*. Repositioning from the right edge of the screen to the left edge on the next line down is called horizontal retrace. Repositioning from the bottom right corner to the top left corner is called vertical retrace. The beam is *blanked* (turned off) during retrace, just as your hand is raised off the paper when you reposition the pencil.

As you might guess, the signal that causes the CRT monitor to perform a horizontal retrace is called horizontal sync (HSYNC), and the signal that causes the CRT monitor to perform a vertical retrace is called vertical sync (VSYNC). An important point to realize is that the times at which horizontal and vertical retrace occur are the responsibility of the video-signal generator and not the CRT screen or monitor electronics (see figure 2).

It is important that the controller provide HSYNC and VSYNC timing within the limits specified by the CRT. Some CRTs will self-destruct if sync timing is incorrect! In the case of a computer, the video controller sync timing is generally software-programmable to allow for compatibility with different CRTs. This means the CRT has the distinction of being one of the few pieces of computer hardware that might be physically destroyed by software bugs. Read and heed!

CRT Timing

Scan timing involves a number of constraints. First, the CRT phosphor, like a dynamic RAM, needs to be refreshed. As soon as the beam passes a point on the screen, the image at that point will begin to fade. If the refresh period (time between consecutive beam passes) is too slow, the

(continued)

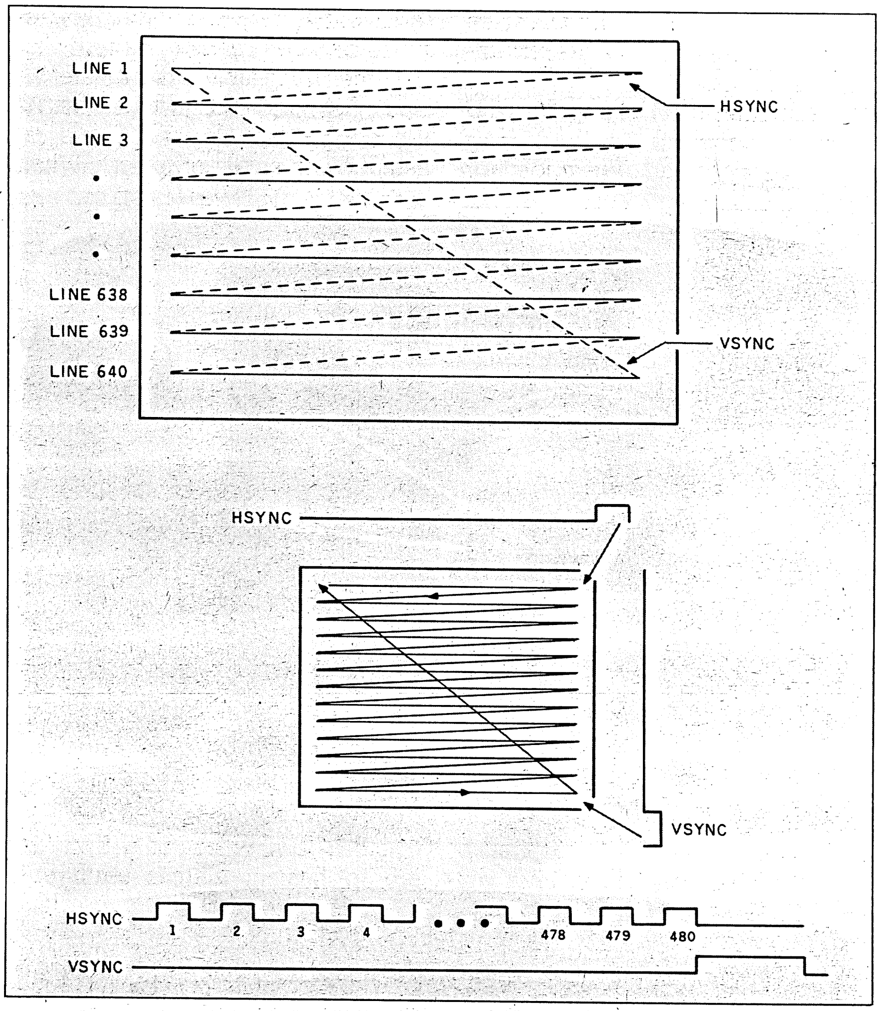


Figure 2: *The electron beam scans the* CRT *much as you would write on a piece of paper.* HSYNC *causes the beam to perform a horizontal retrace;* VSYNC *causes it to perform a vertical retrace.*

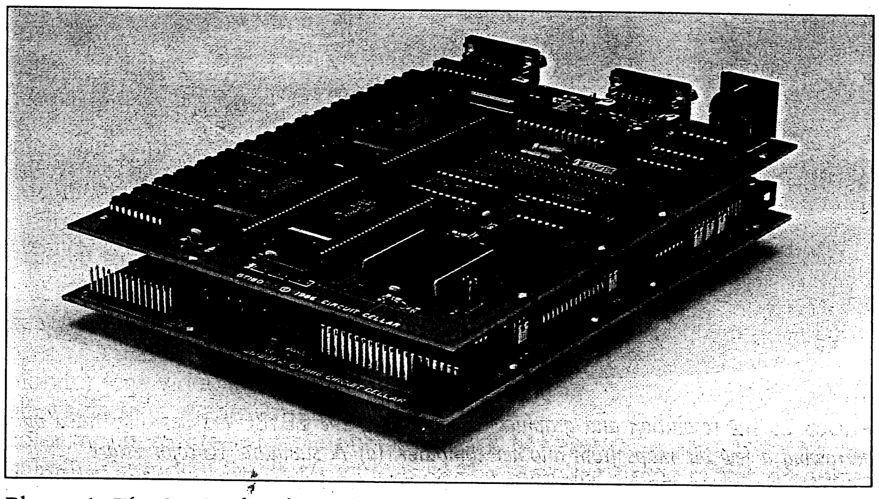
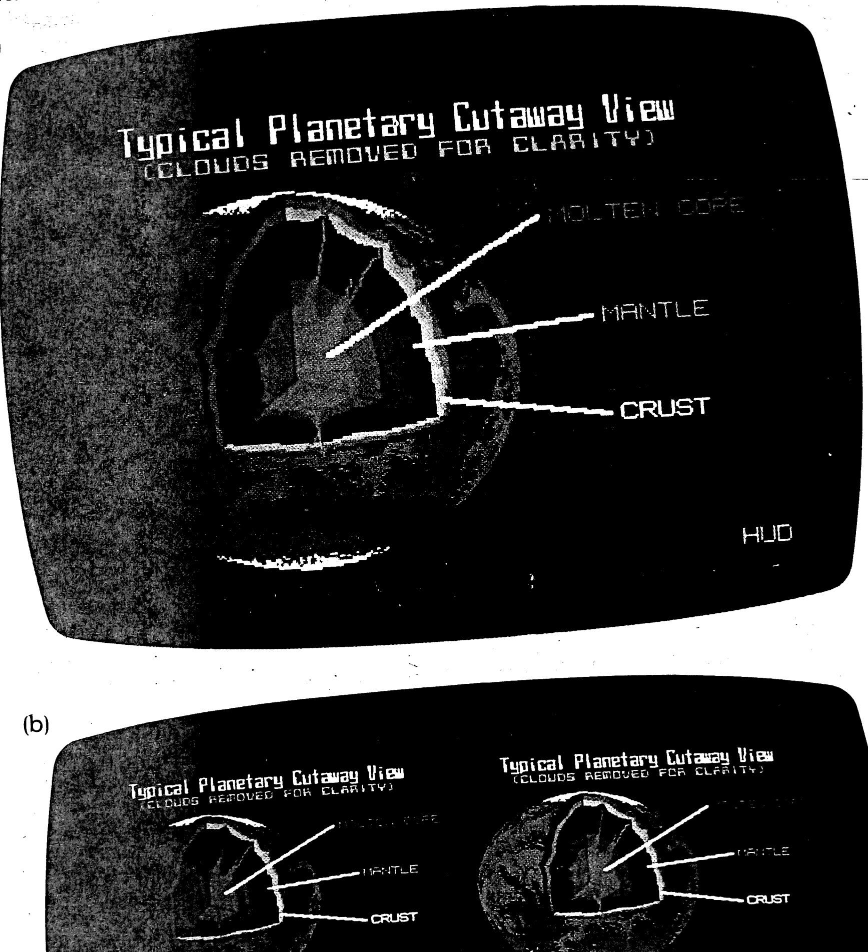


Photo 1: *The* GT180 *board attached atop the new* SB180FX *single-board microcomputer. (The* SB180FX *is an enhanced version of the* SB180. *See the September* 1985 *Circuit Cellar for a description of the* SB180.)

result will be an annoying flicker as each point cycles through the refresh-fade sequence.

One solution is to use a special CRT phosphor characterized as *long persistence*—this means the phosphor fades more slowly, allowing longer times between refresh before flicker becomes noticeable. Unfortunately, a phosphor with a persistence that is too long produces an effect that is just as annoying as flicker: smear. When the picture on the CRT changes, a ghost of the previous image persists. If the CRT is updated quickly, detail is lost as each new image is superimposed on the vestiges of the past images.

(a)



(b)

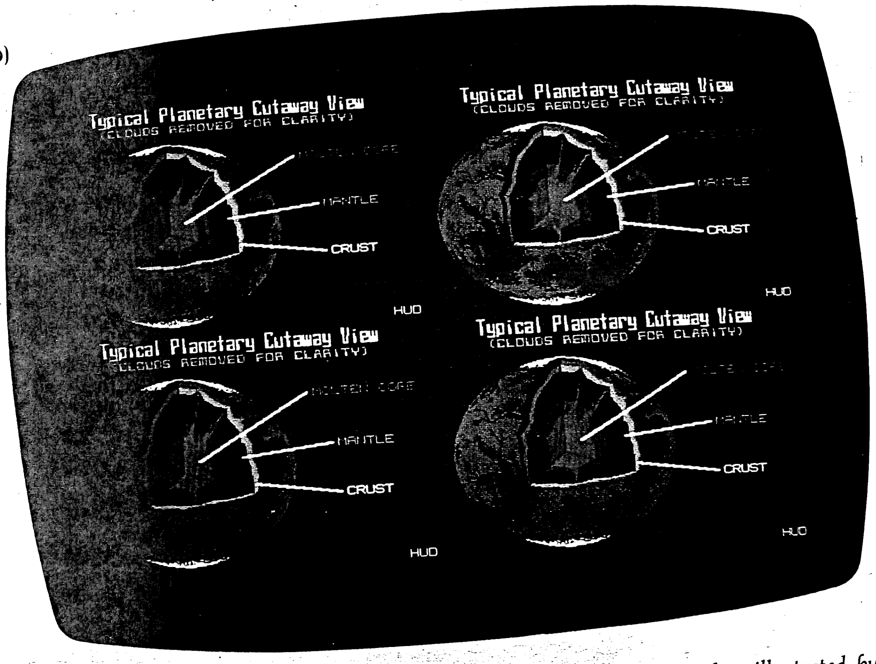


Photo 2: *The resolution and graphics capabilities of the SB180 are best illustrated by borrowing a few bit maps from another computer. (a) A standard 16-color Atari 520ST bit-mapped picture with 320- by 200-pixel resolution. By setting the GT180 to a resolution of 640 by 400, four Atari pictures can be put on a single screen. The result is shown in (b). The GT180 can be set to a display resolution as high as 720 by 500.*

Therefore, the first major constraint can be characterized as the number of times the beam sweeps the entire screen per second (i.e., the number of VSYNCs per second). It's clear that faster is better, since a short-persistence phosphor (fast updating with no smear) can be used while avoiding the threshold of perceivable flicker. Most commonly, designers choose 60 hertz (60 VSYNCs per second) because this provides a good performance, matches standard phosphor characteristics, and is easily derived from the 60-Hz component of the AC power line (within the U.S.). High-performance (and high-price) systems may offer faster vertical scan rates (i.e., 65–80 Hz).

Since we want to sweep the entire screen, line by line, 60 times each second, a rough calculation shows that very high speed is required. After all, an enhanced screen (640 by 400) contains 256,000 dots. If we insist that the dots be refreshed 60 times a second, that means the video controller must transmit more than 15 million dots per second!

The scanning technique discussed so far, in which each line on the screen is scanned sequentially, is called noninterlaced. To ease the high-speed timing constraints, some systems often use another scanning approach: interlaced. In this scheme, the beam still scans the entire screen at 60 Hz. However, instead of scanning every line during a sweep of the screen, half the lines are scanned, reducing the amount of information that must be transmitted. In fact, conventional TVs use interlaced scanning; although the picture contains 525 lines, only half of them are refreshed each 1/60 of a second. (Note: Actually, not all lines are displayed on the TV screen; some contain supplementary information like closed captions for the hearing impaired and stereo audio.)

If you've followed the previous discussion, you may have deduced that

TV sets must use a long-persistence phosphor to ensure that flicker doesn't occur. The price you pay for the long-persistence phosphor is some smear, but objects on TV are moving slowly enough so that the effect is unnoticeable. Unfortunately, the same cannot be said for computer images, in which operations like scrolling quickly highlight the smear effect. Generally speaking, high-performance computer displays use only noninterlaced scanning.

Accepting the 60-Hz noninterlaced vertical scan, the other important CRT timing considerations include the horizontal scan rate and video bandwidth. Together, these three parameters dictate the actual resolution (i.e., the number of dots that can be displayed on the screen).

As the vertical scan rate (60 Hz) corresponds to the rate of VSYNC, the horizontal scan rate corresponds to the rate of HSYNC. Given the constraint of scanning all the lines of a screen in 1/60 of a second, you can see that for a given line width, the number of lines that can be displayed on the screen is directly related to the speed with which each line is horizontally scanned. For example, older displays like the IBM CGA, which have a resolution of 640 by 200, use a horizontal scan rate of 15.75 kilohertz. Similarly, displays with a resolution of 640 by 400 have a horizontal scan rate of 31.5 kHz.

The final CRT timing parameter is known as bandwidth or dot rate. This refers to the timing of the actual video signal that modulates the electron beam in the CRT. This is influenced by a number of factors, including the CRT video-signal input circuits and the response time and accuracy of the electron gun. The higher the required resolution, the higher the bandwidth needed. A 640 by 200 noninterlaced 60-Hz CRT typically requires a bandwidth of about 15 megahertz, while a similar 640 by 400 CRT needs 30-MHz bandwidth.

Although CRTs have a number of other timing parameters (like sync pulse widths), the three primary parameters—vertical scan rate, horizontal scan rate, and bandwidth—along with scan mode (interlaced or noninterlaced) define the overall performance envelope.

Monochrome vs. Color

Previously, we assumed that a single scanning electron beam could take only two states: on or off. In fact, this is the case for a monochrome display like the Macintosh. Achieving a color display is an elaboration of the same basic scheme. Two changes must be made: The CRT must be able to display multiple colors, and the video controller must provide color information for each dot.

The most prevalent technique for making color CRTs (and color TVs) is to replace the single electron beam with three beams and to coat the CRT face with three phosphors. Each beam/phosphor is responsible for generating a different color: red, green, and blue. You can generate eight different colors by combining red, green, and blue. For instance, you produce a white dot by turning the red, green, and blue beams on simultaneously.

The video controller must also provide three signals, instead of the single signal required for monochrome. In fact, a fourth signal, intensity, is often provided, giving the effect of 16 possible colors instead of 8. For example, white is red+green+blue with intensity on; gray is red+green+blue with intensity off. This type of CRT is referred to as an RGBI monitor.

Digital vs. Analog

In the previous discussion, for both monochrome and color, the video signal (one signal for monochrome, four signals for RGBI) was assumed to take one of two states: on or off. This is called a digital TTL monitor.

An analog monitor adds the capability to modulate the electron beam(s) to intermediate levels between on and off. For a monochrome monitor (one video-input signal), this allows various shades of gray. For a color monitor (three video-input signals: R, G, and B), it allows various tints and hues. The visual results are much more pleasing than those of digital monitors. TV sets use this approach. Moving upscale from monochrome to either digital color or analog color is just a matter of more memory and more speed—no problem in this era of VLSI.

The vertical scan rate, the horizontal scan rate, and the video bandwidth dictate the actual resolution.

The Frame Buffer

I have explained how a CRT turns video signals into pictures on the screen. Where does the video signal come from?

In the case of a conventional TV receiver, the video signal comes from a station transmitter and is captured by the receiver in the TV for display. The TV station transmits a continuous stream of display frames, eliminating any requirement for video-frame retention or buffering in the TV receiver.

The computer holds a digital representation of the screen in a special memory called the frame buffer. Display circuits in the computer extract and condition this video information to generate the video signal(s) required to recreate the image on the CRT screen. A key part of this function is parallel-to-serial conversion. The display controller pulls a number of bits (8, 16, 32, or more) from the frame buffer at once and shifts them out as a serial video signal. The shift clock is also known as the dot clock and corresponds directly to the CRT bandwidth discussed earlier. To change the image on the CRT—a process known as drawing—the drawing processor need only change the contents of the frame-buffer memory (see figure 3).

Bit Mapping

The simplest frame-buffer organization is called bit mapping. In this organization, a bit (monochrome) or bits (color) in the frame buffer are pro-

(continued)

vided for each dot on the CRT screen. The bits in the frame buffer map directly to points on the CRT screen. The minimum amount of frame-buffer memory required is equivalent to the number of dots on the screen times the number of bits required to specify the color information for each dot.

For a monochrome bit map, only 1 bit is required for each dot on the screen. Thus, a 640 by 400 monochrome display requires about 32K bytes. For a 16-color RGBI display, where each dot requires 4 bits of information (R, G, B, and I), a 640 by 400 image requires about 128K bytes.

Palette D/A Converter

For a TTL RGBI color monitor, it is easy to see that 4 bits (R, G, B, and I) are required for each dot on the display. Furthermore, the system is totally digital. The frame buffer contains 1s and 0s, and the CRT itself accepts 1s and 0s (i.e., the CRT R, G, B, and I inputs are either on or off). What is the difference between it and an analog RGB display?

First, there is a difference in the CRT monitors themselves. In a TTL RGBI monitor, the three beams are either at a low or high intensity as defined by the state of the intensity signal. To achieve a greater variety of colors, the intensity of each beam is varied. Instead of on/off TTL levels that would be applied to a TTL RGBI monitor, an analog RGB monitor is sent an analog voltage for each beam.

Within the frame buffer, the color information is still stored digitally, but each dot must now include the intensity settings for each beam as a 6-bit (64 colors), 9-bit (512 colors), or 12-bit (4096 colors) data word. This digital information is converted to analog video-signal levels through a D/A converter.

Since each of the three CRT guns (R, G, B) must be driven independently, we need three D/A converters. Unlike the industrial-control-oriented D/A converters discussed in previous articles, these "video D/A converters" need to be very fast: Bandwidth (the speed with which the D/A converter converts the digital frame-buffer information to an analog video signal) must match that of the CRT. Furthermore, other D/A converter performance characteristics are also strict in order to produce a glitch-free display.

As in the case of the digital monitor, the number of colors that can be displayed is a function of the number of bits in the frame buffer for each dot. Thus, 4 bits/dot means each dot can take one of 16 colors, 8 bits/dot=256 colors, 12 bits/dot=4096 colors, etc. Since the system drives three guns (R, G, B), we normally select a number of bits per dot that is evenly divisible by 3. A typical application might have 12 bits per dot, using three 4-bit D/A converters. Thus, each signal (R, G, and B) can take on 16 levels. The total number of combinations yields 4096 colors.

With that in mind, it would appear that to move up from a 16-color display to a 4096-color display simply requires tripling the frame-buffer size, since each dot now requires 12 bits. While the increased memory is supportable (384K bytes for a 640 by 400 display), unfortunately a substantially higher bandwidth is also required. To

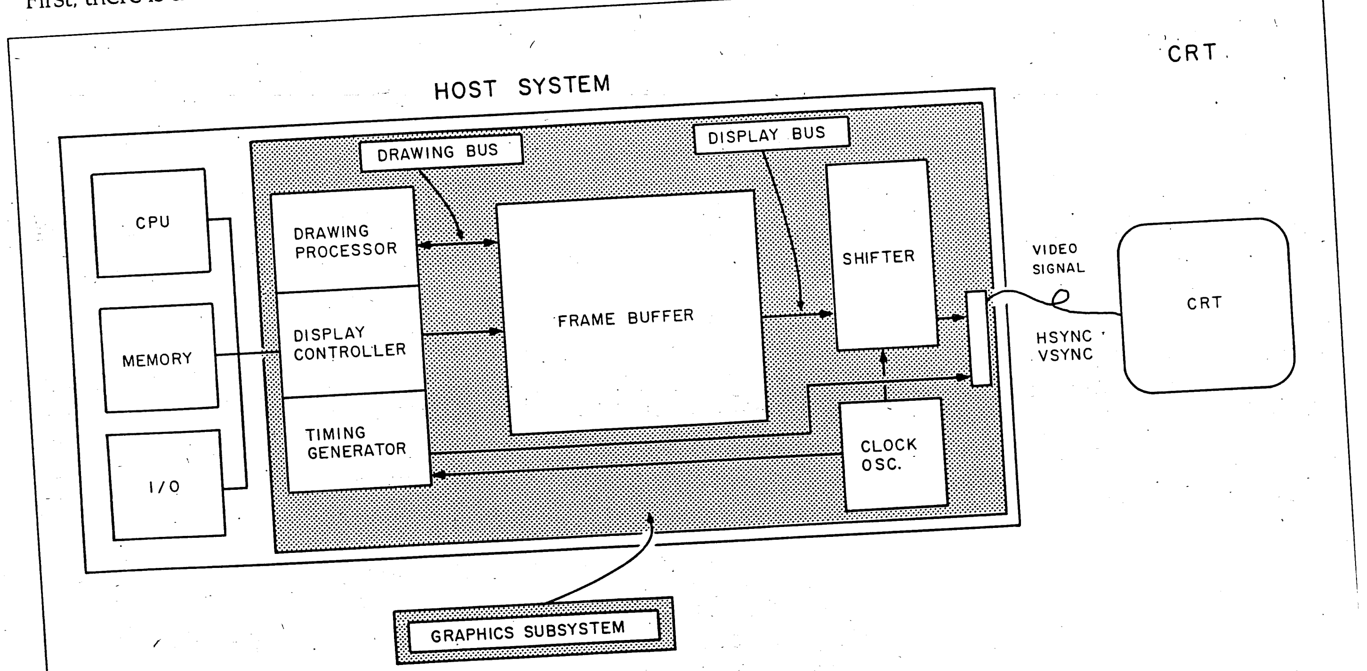


Figure 3: *A diagram of a typical graphics system, including host computer, graphics subsystem, and CRT. Key components of the subsystem are the frame buffer, drawing processor, display controller, timing generator, and shifter. Data is read in parallel fashion from the frame buffer by the display controller, which passes it to the shifter to be serialized into the video signal. The timing generator controls basic system timing and generates the HSYNC and VSYNC control signals.*

display a dot on the screen, 12 bits (4096 colors), instead of 4 bits (16 colors) or 1 bit (monochrome), must be pulled from the frame buffer to refresh each dot on the screen 60 times a second.

As we calculated earlier, for a 640 by 400 monochrome display with 60-Hz vertical scan, we need to extract 15,360,000 bits from the frame buffer each second. While this is fast, most memory ICs and display controllers can handle the required transfer rate of 2 megabytes per second. With 12 bits instead of 1 bit per dot, however, the speed requirement increases twelvefold! Instead of 15.36 megabits per second, the system must extract 184 megabits per second (23 megabytes per second)! This is too fast for most microcomputers.

The solution is to use a device called a palette D/A converter. Like an artist's palette, this device provides the total range of color possibilities, even if we can use only a few of them at one time. The technique is straightforward: The typical palette RAM, like the Brooktree BT450, is organized as 16 cells, each containing 12 bits of data. Within the 16 cells are stored 12-bit values representing 16 of the 4096 possible colors (these 16 cells are loaded by the CPU prior to a video scan). Each dot on the display is defined by a 4-bit value in the frame buffer. These 4 bits serve as an address selecting one of the 16 palette entries. The 12 bits stored in the selected cell are in turn presented to three 4-bit D/A converters connected to the RGB lines (see figure 4).

While the 16 colors available through a palette D/A converter may seem to offer no improvements beyond a 16-color RGBI system, remember that we can choose 16 of 4096 colors. It is possible for the CPU to change the contents of the palette D/A converter at the end of each horizontal scan line. Rather than showing only 16 colors then, an entire screen can display all 4096 colors. Reloading the palette can create some interesting effects. For example, the color of a certain area on the screen can be changed by simply changing the corresponding palette entry—this is considerably faster than actually changing the color of each dot in the frame buffer.

Split Screens and Windows

A powerful visual interface like the Macintosh depends on graphics techniques like split screens and windows to implement popular features like pull-down menus, dialog boxes, and application windows. In a simple bit-mapped system, the display controller cycles sequentially through the frame buffer from beginning to end and back to the beginning. In other words, the relative position of an image on the screen corresponds exactly to its relative position in the frame buffer. To move a window on the screen requires physically moving the associated image in the frame buffer. If you shrink or move a window on the display, you must also restore the newly uncovered background.

Having the host CPU responsible for all this is possible, but problems emerge as the amount of information to be moved increases. The screen response for tasks like dragging a window can become annoyingly sluggish. In any case, CPU cycles devoted to screen housekeeping can be used for more useful work.

Adding intelligence to the display controller provides hardware split screens and windows. This feature requires that the display controller, in-

Moving a screen image becomes as simple as programming a new start address instead of actually moving each dot.

stead of scanning in a fixed sequential order, be able to scan different areas of the frame buffer at will. The programmer defines the physical position on the screen of a split or window. Then the display controller monitors the physical location being displayed and accesses the appropriate location in the frame buffer. This allows the various pieces of the screen image (menus, dialog boxes, windows, etc.) to be stored in separate areas of the frame buffer—the hardware will automatically put each piece in the right place on the screen (see figure 5). Moving a screen image becomes as simple as programming a new start address instead of actually moving each dot. (Note that I am implying that the frame buffer may be

(continued)

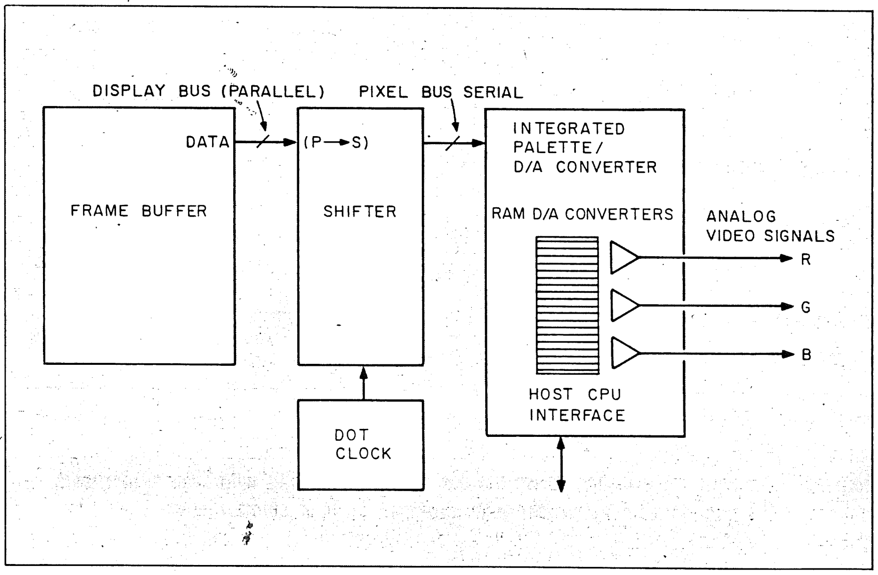


Figure 4: *New integrated palette D/A converters, like the Brooktree BT450 used on the GT180, combine a number of functions formerly requiring separate chips.*

larger than can fit on the screen at one time.)

DRAWING

The process of writing into the frame buffer is called drawing, while the process of reading from the buffer to the CRT is called display. The drawing and display processes contend for access to the frame buffer. One way to resolve such contention is to use dual-port RAMs for the frame buffer. Another approach is timeshared access to a single-ported frame buffer.

In any case, the application programmer needs to be able to move pictures in and out of the frame buffer and issue commands like LINE, CIRCLE, PAINT, WINDOW, and ZOOM. Ideally, the programmer can use logical *x,y* coordinate addressing, rather than being burdened with computing a frame-buffer physical address (e.g., "move the pointer to screen *x,y* position (100,200)" instead of "move the pointer to the dot in the high-order 4 bits of address 1234 hexadecimal"). A myriad of other graphics commands can be imagined for use in a variety of applications.

Opinion has been divided as to whether the host CPU should perform these drawing algorithms, or whether a separate drawing processor should be used. My opinion has always been to do it in hardware, and I think the trend to higher resolution, more colors, and faster response will ultimately lead others to conclude that an intelligent graphics coprocessor is the only way to go.

TRANSPORTABLE GRAPHICS

Any graphics system has several distinct command levels. At the lowest level is the hardware: the registers associated with the graphics-display chips used on the display board. At the next level is usually a group of drawing primitives: point plotting and line drawing. Next, a graphics environment like that found in GEM and Microsoft Windows may be defined that provides standard function calls like "draw a box containing this text" or "open this window." At the highest level is the application program that provides the user with commands necessary to complete the task the program was designed to perform.

An ongoing debate is occurring over which of the above levels should be performed by hardware and which should be performed by software. (Most of the systems in use today perform the higher two levels in software on the host system.) Further, there must be, at some level, a standard interface between the host processor and the graphics device. For example, on IBM's CGA card, the hardware level consists of the 6845 registers (the 6845 is the PC's graphics-controller chip). At the primitive level, the BIOS provides simple character output. To remain compatible with the various graphics boards and clones on the market, programs should make calls to the BIOS for character display. Programs that access the 6845 directly may run into problems.

The Hitachi HD63484 ACRTC takes control of graphics operations one level higher than most graphics boards do. Most graphics boards provide the host processor with a set of registers and perhaps a simple set of drawing primitives in EPROM. The host processor is still responsible for all drawing overhead, including translation of *x,y* coordinates to absolute memory addresses, setting of color information, and calculating each dot for a given line or arc. Consequently, the graphics device/host processor interface takes place at the lowest level: the hardware level.

The ACRTC, however, performs many primitive functions right on the chip. For example, it translates *x,y* coordinates into absolute addresses; draws lines, given the endpoints, boxes, arcs, and ellipses; and performs screen clears and area fills using an optional pattern. All these functions are performed in hardware. Using this as the graphics controller, the graphics device/host processor interface shifts up a notch to the primitive level. The host processor does all its communications with the graphics device via calls to primitives rather than register accesses.

Another issue related to the graphics board/host processor interface is that of how to maintain compatibility when using a program with different kinds of hardware. There are any number of ways to build a graphics controller in hardware that provides a given resolution and number of colors. Each design is bound to use a different graphics-controller chip and, as a result, a different method of communicating with the host processor

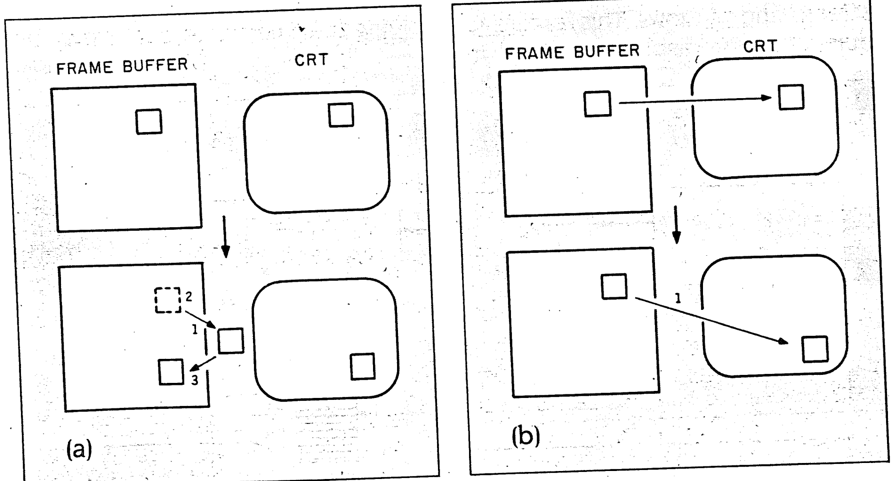


Figure 5: *(a) If the display controller isn't "smart," moving a window requires actually moving the data. The algorithm might require a three-step process: Copy the window contents to a temporary buffer; erase the old window, restoring what was underneath; and copy the window contents to the new location. (b) In a smart display controller, changing a window position is as simple as reprogramming a few display-controller address registers. This method is fast, and the time required to move the window is independent of window size. Also, the hardware window has priority, so what was underneath the moved window reappears.*

(the registers will probably be different between boards).

The primitive level is usually used as a buffer between a higher-level program that uses a standard set of function calls and the hardware that is different between various boards. If the higher-level program stays on its side of the fence and doesn't cross over into the hardware level, it will function properly with every graphics board, regardless of hardware configuration. When the program does start accessing hardware registers specific to a particular graphics board, it will likely have trouble when used with a different board.

The implementation modules for the set of tools used with the GT180 were written with the HD63484 in mind. However, as long as the target graphics board can display the same resolution and number of colors as the GT180, a program written using Borland's Graphix Toolbox (designed to interface via Borland's Modula-2) can be ported with little effort to the new graphics board. The definition modules of all the tools will stay the same, regardless of what machine it is running on. The implementation modules contain all the machine-specific code.

There still hasn't been a graphics standard defined that is widely used in industry. The problem is exacerbated by a graphics technology that grows in leaps and bounds, often opening new frontiers not considered in older standards proposals. The issue of where to place the graphics board/host processor boundary is a constant issue. For example, all the IBM graphics adapter cards require hardware-level programming (even though there are BIOS calls, the processor still must do all the work). With the SB180/GT180, the ACRTC does most of the low-level primitives.

The physical interface between devices must also be considered. Most display adapters plug directly into the backplane, allowing the processor direct access to all hardware registers. The GT180 also plugs directly into the SB180's XBUS, but the processor must talk to the ACRTC through a pair of I/O ports, even though the ACRTC does have internal registers. The host processor doesn't need to access the hardware registers often (if at all). Since this graphics device/host boundary is at a higher level requiring less information exchange, it is easier to merely send the ACRTC a stream of drawing commands.

The real problem of defining a graphics standard comes down to defining a set of commands that is rigid enough to allow simple program transportation and that takes hardware differences into account while allowing the programmer to take advantage of any special features of the graphics processor. There is no easy solution, as evidenced by the lack of a widely accepted graphics standard. I hope that the SB180/GT180 with Modula-2 and Graphix Toolbox will make the waiting more constructive.

CIRCUIT CELLAR FEEDBACK

This month's feedback begins on page 58.

NEXT MONTH

Part 2 looks at the GT180 hardware. ■

Special thanks to Tom Cantrell, Ken Davidson, and Mike Weisert for their contributions to this project.

All screen pictures presented in this article were produced using the GT180 with a Princeton SR-12P monitor. The bit-mapped pictures were originally composed on an Atari 520ST using DEGAS by Tom Hudson. They are reproduced and used here by permission.

The following items are available from

The Micromint Inc.
25 Terrace Dr.
Vernon, CT 06066
(800) 635-3355
(203) 871-6170
Telex: 643331

1. GT180 graphics board: RGBI version less palette D/A converter. Comes with demo disk and user's manual.
board alone $395
board with Modula-2 and GT180 Graphix Toolbox $449

2. GT180 graphics board: RGBI and analog version with palette D/A converter. Comes with demo disk and user's manual.
board alone $449
board with Modula-2 and GT180 Graphix Toolbox $499

3. Borland International Turbo Modula-2 and GT180 Graphix Toolbox software for the SB180 and SB180FX computers, optimized for the 64180 processor. Supplied on 5¼-inch DS/DD SB180 format disks with 300-page manual.
SB180 Modula-2 alone $69
SB180 Modula-2 with Graphix Toolbox alone $89

4. SB180FX 5.75- by 8-inch single-board computer, accommodates 512K bytes of memory, two serial ports, three parallel ports, parallel printer port, floppy disk controller, SCSI controller, ROM monitor, 6-MHz 64180. Comes with ZRDOS, ZCPR3, hard disk BIOS, and user's manuals. Populated with 256K-byte memory, less 53C80 SCSI controller chip.
SB180FX board alone $409
SB180FX board with software $499
9.288-MHz 64180 processor upgrade (SB180FX only) $50

GMIC, GVAC, ACRTC, and palette D/A converter chip sets are available for experimenters who wish to hand-assemble the GT180. Call for price and availability information. Borland Turbo Modula-2 is also available for most CP/M Z80 machines. Consult regular CP/M and 8-bit software distributors for various disk formats and prices. The SB180FX is hardware- and software-compatible with the SB180.

Surface delivery (U.S. and Canada only): add $5 for U.S., $10 for Canada. For delivery to Europe via U.S. airmail, add $20. Three-day air freight devlivery: add $8 for U.S. (UPS Blue), $25 for Canada (Purolator overnight), $45 for Europe (Federal Express), or $60 (Federal Express) for Asia and elsewhere in the world. Connecticut residents please add 7.5 percent sales tax.

There is an on-line Circuit Cellar bulletin board system that supports past and present projects. You are invited to call and exchange ideas and comments with other Circuit Cellar supporters. The 300/1200/2400-bps BBS is on-line 24 hours a day at (203) 871-1988.

Editor's Note: Steve often refers to previous Circuit Cellar articles. Most of these past articles are available in book form from BYTE Books, McGraw-Hill Book Company, P.O. Box 400, Hightstown, NJ 08250.

Ciarcia's Circuit Cellar, Volume I covers articles in BYTE from September 1977 through November 1978. *Volume* II covers December 1978 through June 1980. *Volume* III covers July 1980 through December 1981. *Volume* IV covers January 1982 through June 1983. *Volume* V covers July 1983 through December 1984.

To be included on the Circuit Cellar mailing list and receive periodic project updates and support materials, please circle 100 on the Reader Service inquiry card at the back of the magazine.