Steve Ciarcia

Part 3: Software

Build the GT180 Color Graphics Board

A look at the software that drives the graphics system

During the last two months, we investigated the GT180's graphics hardware design, CRT basics, and the roles of key chips: the ACRTC, GMIC, GVAC, and palette D/A converter. This month, we'll look at the software that drives the graphics system. We'll start with an overview of the ACRTC registers and commands and then introduce a high-level software tool—Borland's Modula-2 with SB180/GT180 graphics extensions.

Programming the ACRTC

The ACRTC is an extremely complex device, containing three separate 16-bit processors, more than 200 bytes of registers, and 38 high-level commands. The on-chip CPUs perform separate tasks: timing control, display control, and drawing. Each CPU includes specialized registers optimized for its specific task.

In typical operation, the timing control registers establish the basic CRT timing. Once you initialize them, you rarely change them. The contents of the display control registers specify the frame-buffer scanning method, including hardware split screen and window. You will periodically reprogram these registers to move or resize splits and windows. Drawing commands and parameters issued to the ACRTC create an image on the screen.

A complete discussion of each of the more than 200 ACRTC registers is beyond the scope of this article. (This information is contained in the *Hitachi HD63484 User Manual*.) Instead, we'll highlight the main command and control registers.

Like other chips that contain a large number of registers, the ACRTC adopts an indirect addressing mechanism that reduces the number of address lines required to specify an individual register. The ACRTC uses only one address line, RS (register select), instead of eight address lines to access the more than 200 bytes of registers on-chip. Accessing a particular ACRTC register is a two-step process. First, write the register address of interest into the address register (RS=low). Then, read from or write to the selected register (RS=high).

Reading the status register returns the overall state of the ACRTC. Information returned includes whether a command has completed or a command error has occurred. Also, to support the clipping and hitting functions, an area-detection flag is provided. This is set when a drawing operation attempts to enter (hit) or leave (clip) a programmer-defined area on the screen. Another bit in the status register indicates when an optional light pen has been activated. (The GT180 uses this bit as a flag that indicates when vertical sync is occurring.) Finally, 4 bits reflect the state of the separate read and write first-in/first-out registers that communicate with the ACRTC drawing processor.

To speed drawing operations, separate 16-byte read and write FIFOs buffer communication to and from the ACRTC drawing processor. As mentioned above, the status register allows you to determine the FIFO's state. For the read FIFO, the status register shows whether the FIFO is full or not empty. For the write FIFO, the status register shows whether the FIFO is empty or not full. While the drawing processor is a 16-bit CPU (and the ACRTC has a 16-bit data bus), the SB180 interface is 8 bits wide. Consequently, commands, parameters, and data are transferred in high byte–low byte order.

Command Control Register

The lower 8 bits of the command control register correspond exactly to the 8 bits in the status register and are used to enable or disable each status bit from generating an interrupt to the CPU. For instance, as an alternative to polling, you could program the system so that the FIFO's state generates an interrupt, invoking the CPU to read or write the appropriate FIFO.

Besides polling and interrupt-driven transfer, the ACRTC can also request direct memory access transfer. This is ideal for high-speed reading and writing of the frame buffer. In response to a data-transfer command, the ACRTC will automatically invoke DMA to move the data between the frame buffer and main memory. You can program the type of DMA request as either burst or cycle steal (correspondingly, you must program the HD64180 DMA controller to be level- or edge-sensitive).

You specify the number of colors the ACRTC supports by programming the number of bits per dot as either 1 (monochrome), 2 (4 colors), 4 (16 colors), 8 (256 colors), or 16 (64K colors). In the GT180, 4 bits per dot is specified.

Finally, 2 bits allow you to abort or pause ACRTC command processing. An abort stops command processing, clears the FIFOs, and reinitializes the status register. A pause simply stops command processing without affecting the FIFOs or status register. Paused commands can be restarted later.

Operation Mode Register

The operation mode register determines the ACRTC's overall operation mode and

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.

The ACRTC alternates frame-buffer accesses between display and drawing operations. Thus, the GT180 can perform drawing operations at any time.

must be initialized before enabling the display.

Both display and drawing operations contend for access to the frame buffer. In some older designs, the display operation required full-time, top-priority access to the frame buffer to meet CRT timing constraints. The resulting approaches for drawing were either draw at any time, overriding display accesses, or draw only during retrace when the CRT is blanked. Neither of these is very productive. In the first one, the conflicting display/draw operation causes the well-known screen "flash" effect; the second one results in slow drawing since retrace time is only about 25 percent of total display time.

The ACRTC has the ability to alternate frame-buffer accesses between display and drawing operations using a technique called interleaving (see the text box below). Thus, the GT180 can perform drawing operations at any time (during display and retrace) without screen flash occurring. When the ACRTC uses interleaving, however, twice as many bits must be pulled from the frame buffer each cycle to keep up with the display timing of the CRT. Calculation shows that to meet the constraints of the CRT and use interleaved mode requires pulling 64 bits from the frame buffer each display cycle. Thus, we program the ACRTC graphics address increment mode (within the operation mode register) as 4, meaning four 16-bit words, or 64 bits.

The dynamic RAMs used for the frame buffer need to be refreshed periodically. The ACRTC includes an on-chip DRAM refresh scheme that does the job. Once enabled, the DRAMs are automatically refreshed during horizontal retrace when the CRT is blanked. Some of you might suggest that the periodic scanning of the frame buffer for CRT display eliminates the need for specifically refreshing the DRAMs. This is fine if the frame buffer contains only one screen. In the case of the GT180, however, the frame buffer can hold multiple screens, fonts, icons, etc. Since only a portion of the frame buffer is being displayed at one time, we need to use the ACRTC refresh feature to preserve the contents of the undisplayed portion of the frame buffer.

Display Control Register

This register lets you enable, disable, or blank each of the ACRTC's four logical screen areas: the base, upper and lower split screens, and the window. Only the base screen must be defined (it can only be enabled or blanked, not disabled).

Timing Control Registers

Thirty bytes of timing control registers configure the on-chip timing control CPU to generate the appropriate CRT timing—particularly HSYNC and VSYNC frequency and pulse width. These depend on the specifications of the CRT being used and must be appropriately initialized before ACRTC display or drawing can occur. Also, the timing control registers hold configuration information for the split screens and window (see figure 1).

Display Control RAM

Forty-eight bytes of registers referred to as the display control RAM configure the on-chip display control CPU to modify the frame-buffer display address generation to account for the split screens and window (see figure 2). The split screens and window are specified in terms of physical frame-buffer addresses.

Drawing

Of the three on-chip CPUs (timing, display, and drawing), the drawing processor is most like a conventional CPU. Besides containing some registers, the drawing processor executes a sequence of user commands that correspond to a program on a conventional CPU. The drawing processor is programmed via FIFOs, providing the same high-performance benefits as a pipeline on a conventional CPU.

Register-Access Commands

Since communication with the drawing processor is via FIFO, the drawing processor provides a special set of commands to allow the programmer to access the drawing registers. Two distinct sets of drawing registers are used: the drawing parameter registers and the pattern RAM. These registers modify and control the way in which a drawing command is executed (see figure 3). Items programmed by the drawing parameter registers include colors, patterns, clipping area definition, modify mode, and other parameters.

Data-Transfer Commands

These commands allow high-speed reading, writing, clearing, and modifying of the frame buffer. This is especially useful for applications with digitizers or scanners, devices that construct an image as an actual bit map rather than as a sequence of drawing commands. Also, you can implement your own drawing commands using these data-transfer commands as basic building blocks.

Drawing Commands

These commands cause the ACRTC to automatically draw a number of common figures (like lines, circles, arcs, and rectangles) and to perform operations like filling and painting. The commands provide absolute and relative address versions. Absolute versions specify an address (like the endpoints of a line) as *x,y* displacements from an "origin" whose location in the frame buffer is set with the ORG command. Relative versions specify addresses as an *x,y* displacement from a "current pointer" location. You can change the current pointer location with

Interleaved Access Mode

The ACRTC's interleaved design for screen access provides considerable advantages over a noninterleaved access method. In the latter, drawing can occur only during retrace time minus the time for DRAM refresh. Since display time accounts for 68 percent of the total time available, and DRAM refresh occurs 7 percent of the time, drawing time is about 18 percent for a noninterleaved design.

An interleaved design permits display and drawing operations to alternate, increasing drawing time by 34 percent (half the display time of a noninterleaved system). This gives a total drawing time of 52 percent—nearly three times faster than a noninterleaved display.

In fact, for computer-bound (not bus-bound) instruction sequences, the relative improvement of interleaved mode will be higher than a factor of 3. This is due to the effect of idle drawing cycles—drawing cycles that the ACRTC can't use because it is performing an internal computation. In noninterleaved mode all idle drawing cycles are wasted, while in interleaved mode some idle cycles will overlap with timeshared display cycles, reducing the effective waste.

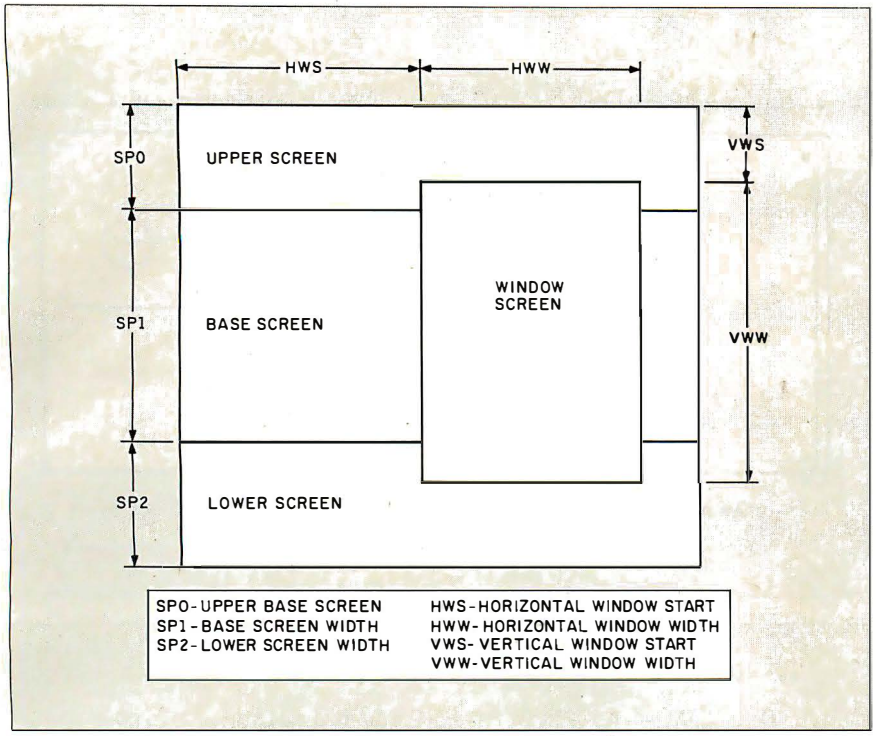

Figure 1: *Besides establishing basic CRT timing, the timing control registers partition the screen into the upper, base, lower, and window portions. The upper, lower, and base screens are all background screens that are overlapped by the foreground window. The vertical specifications (SP0, SP1, SP2, VWS, and VWW) are in units of rasters, while horizontal specifications (HWS, HWW) are in units of display cycles. Usually, only the base (covering the entire CRT screen) need be defined.*

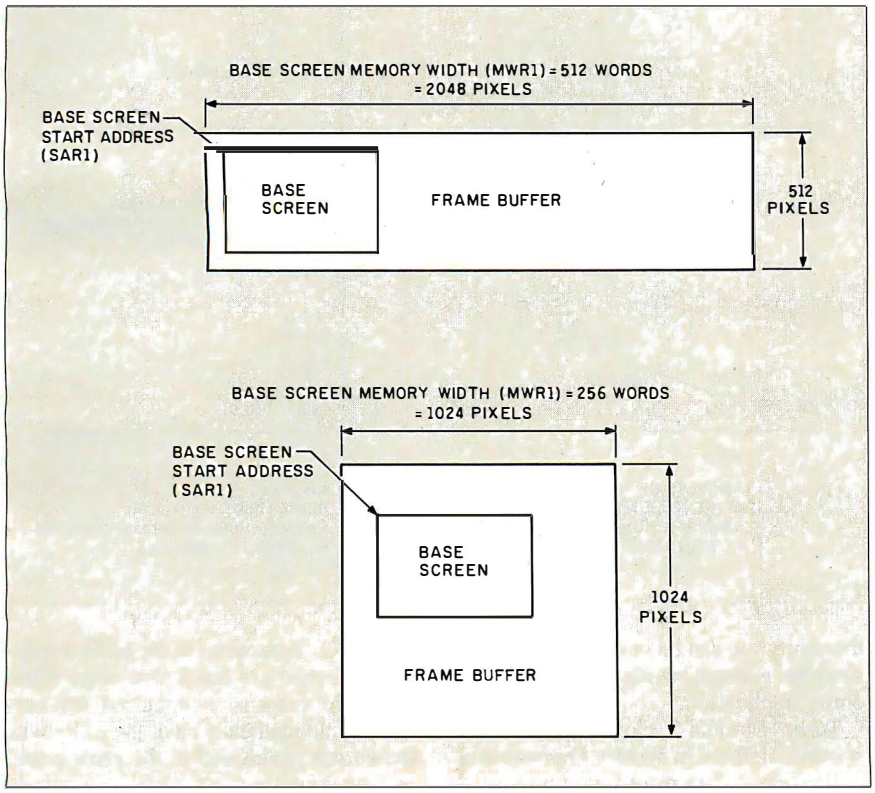

Figure 2: *The display control RAM's registers associate each CRT screen partition (upper, base, lower, and window) with a physical location in the frame buffer. In these examples, a 640 by 480 base screen is mapped into the frame buffer using two different memory-width values. (The memory-width parameter tells the ACRTC how many pixels in the x direction are associated with a single raster.) The number of pixels is equal to the memory width times 4, since the standard GT180 defines 4 bits per pixel and memory width is in units of words. The start address associates the top left corner of the screen with a particular pixel in the frame buffer. By changing the start address, the contents of the screen can appear to scroll smoothly in the horizontal and/or vertical direction.*

a MOVE command or as a result of a previous drawing command (see figure 4).

High-Level-Language Graphics

By using detailed knowledge of ACRTC registers and commands, you can write an assembly language program to initialize the ACRTC and draw some figures. However, for more complex applications, many programmers prefer to use a high-level language, preferably with graphics extensions available.

When I considered which popular, high-performance, low-cost language to choose, Borland International's Turbo Pascal emerged as the best possibility. In contacting Borland, I made two fortuitous discoveries. First, an 8-bit version of a new language, Turbo Modula-2, was almost ready and looking for a beta test site. Second, key people at Borland, including R&D engineer Mike Weisert, the compiler writers, and even Philippe Kahn himself, had an interest in exploring the limits of this new hardware and software technology. Above all, Philippe wanted

continued

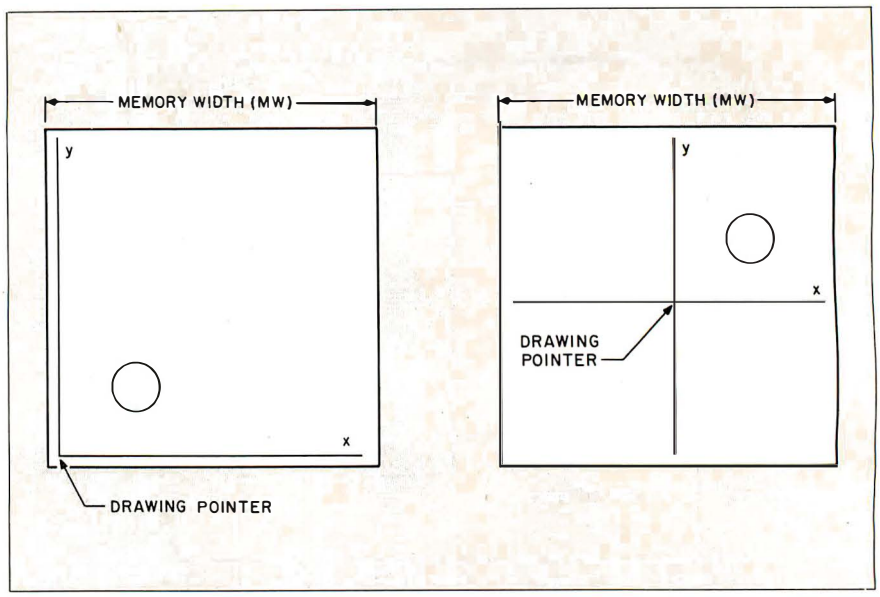

Figure 3: *The ACRTC's graphics-drawing commands (in this example,* MOVE *and* CIRCLE*) use a logical x,y coordinated pixel map independent of a pixel's physical frame-buffer address. The ACRTC uses the drawing pointer to make the translation from x,y coordinates to physical address. The drawing pointer specifies a screen (upper, base, lower, or window), a frame-buffer physical word address, and a dot offset within the word. Given the specified screen's MW and the physical address in the frame buffer associated with coordinates (0,0), the ACRTC can automatically translate an x,y address to a frame-buffer address. The two examples here show the origin in the bottom left corner and the origin in the center of the screen.*

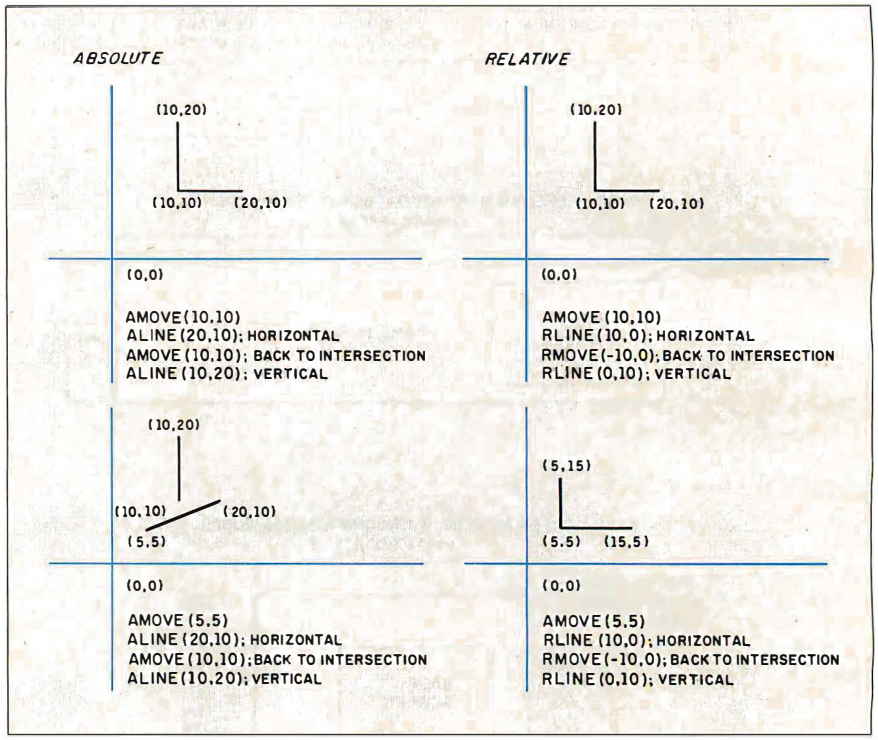

Figure 4: *Absolute-addressing drawing commands specify a displacement from the origin, while relative-addressing commands specify offsets from the current pointer (CP). The CP is set directly by the* MOVE *command and indirectly as the result of other drawing commands (for instance, it is set to the endpoint of a drawn line). These examples illustrate the virtue of using the relative mode. The intention is to draw the same figure at a different location by changing the first* AMOVE *command. Notice how the absolute version requires every instruction's coordinates to be changed, while the relative version works correctly.*

8-bit users to know that he had not abandoned them.

Modula-2 bears a very strong resemblance to Pascal. This is not a coincidence, since both were authored by Niklaus Wirth. Modula-2's primary difference (and improvement) is its inclusion of powerful facilities to allow modular program development. Modula-2 is closely aligned with the concept of structured programming, in which an application is dissected into functional modules. In fact, the details of the implementation of a particular module can be hidden or encapsulated—you need only know the interface definition in order to use the module. Furthermore, you can fix or change individual modules without having to recompile the entire application.

Turbo Modula-2 closely follows the standard defined in Wirth's *Programming in Modula-2*. Extensions are provided to handle I/O, string and exception handling, and other low-level system functions.

Turbo Modula-2 is a complete development environment, including integrated compiler, linker, editor, library manager, and more. It is quite similar in use to Turbo Pascal, including its menu-driven interface and WordStar-compatible editor.

For those of you unfamiliar with Turbo Pascal, you're in for a treat with Turbo Modula-2. Transitions in the edit-compile-run sequence are quick and easy. When a compile error is encountered, not only can you automatically enter the editor with the cursor positioned at the error point, but the compile automatically continues after you edit the flawed statement! Though a compiler, Turbo Modula-2 allows the free-flowing interactive style of programming normally associated with interpretive languages.

To boost performance and ease of use further, Borland has added special features to the SB180/SB180FX version of Modula-2 above and beyond those of the standard Z80 CP/M version. These in-

clude the use of new HD64180 op codes (like INO, OUTO to access on-chip I/O and MLT to speed up multiply routines). Also, the package uses the DU: (drive, user number) scheme for naming files (this worked so well, it was retrofitted to the CP/M version as well). However, the most important feature specific to the SB180 version is its ability to handle programs larger than 64K bytes. Whenever a module is called, Modula-2 reprograms the HD64180 memory management unit as required to access modules located in extended memory.

Turbo Graphix Tools

With Modula-2 in hand, Borland's next step was to create a series of tools (modules and procedures) that provide a simple, high-level interface to the raw power of the ACRTC. Modules are provided at different levels of abstraction. The various procedure modules are layered; higher-level modules use lower-level modules as primitive building blocks.

There are three layers of modules. The bottom layer provides simplified access to the most basic hardware resources contained in the ACRTC and the palette D/A converter. The next layer maps the ACRTC instruction set to Modula-2 procedures. In most cases, the ACRTC instruction format is directly mapped. In others, some preprocessing is done so that the instructions are more straightforward to use. The highest layer provides some enhanced graphics services like loading bit-map images and handling bit-mapped text.

Using these lower layers, you can write your graphics application as one or more higher layers. Examples might include routines to draw a specific image (like a bar or pie chart), a paint or draw program, or a multiwindow visual interface.

Toolbox Modules

Like the ACRTC registers, it is a bit much to try to explain all the Graphix Toolbox modules here. Instead, I'll briefly describe some of the more significant procedures.

ACRTC Module

Procedures within the ACRTC module initialize a myriad of ACRTC registers and set up a default palette. Typically, you should compile this module and include it in your system START alias to initialize the graphics system automatically when the Z-System is booted. The module defines key graphics parameters, including CRT timing and resolution. Thus, by changing the contents of ACRTC (and in some cases the timing crystal), you can accommodate different monitors. Assorted initialization files for a 25-megahertz crystal are included with the Graphix Toolbox. (The GT180 board can use up to a 32-MHz crystal for greater than 780 by 520 resolution.)

The GT180 can use up to a 32-MHz crystal for greater than 780 by 520 resolution.

REGISTERS Module

These routines access the ACRTC FIFO, control registers, drawing parameter registers, and the pattern RAM. The FIFO is accessed constantly to issue commands and transfer bit maps. The ACRTC control registers, like those contained in the display and timing processors, can be directly accessed for special-purpose routines. The drawing parameter registers

continued

Listing 1: *A simple bar-chart program.*

```
MODULE bar;
FROM ACRTC IMPORT Xres, Yres;
FROM Graphics IMPORT aMove,rMove,rLine,rFilledRec,Pattern;
FROM Registers IMPORT ReadParamReg,WriteParamReg,ParamReg,WritePatRAM;
FROM Fonts IMPORT FONT, LoadFont;
FROM BitTexts IMPORT graphic,GotoRC;
FROM Patterns IMPORT SelectPattern, PatternName;

PROCEDURE labelaxis;
TYPE
   month = ARRAY [0..8] OF CHAR;
VAR
  months: ARRAY [0..11] OF month;
  curfont: FONT;
  i: CARDINAL;
BEGIN
  months [0]:='January ';
  months [1]:='February';
  months [2]:=' March  ';
  months [3]:=' April  ';
  months [4]:='  May   ';
  months [5]:=' June   ';
  (* load a font *)
  IF LoadFont (curfont,'M:14X8.FNT',Xres+16*8,0,0FFFFH,0) THEN END;
  GotoRC (3,20);
  WRITE (graphic,'XYZ Company Sales - 1st Half 1986');
  GotoRC (7,0);
  WRITELN(graphic,'Sales'); WRITE(graphic,'$000s');
  GotoRC (33,10);
  FOR i:=0 TO 5 DO
  WRITE (graphic,months[i],'  ');
  END;
  GotoRC (0,0);
END labelaxis;
```

continued

```
PROCEDURE drawaxis;
BEGIN
  WriteParamReg (ColReg0,0H); WriteParamReg (ColReg1,0FFFFH);
  SelectPattern(Empty);  (* black & white - solid pattern *)
  aMove(60,30);
  rFilledRec (2,360);  (* Y axis *)
  aMove(60,30);
  rFilledRec (490,2);  (* X axis *)
  SelectPattern(Arrow);  (* arrowhead *)
  aMove (53,390);
  Pattern(16,11,0);    (* arrowhead y axis);
  aMove (550,39);
  Pattern(16,11,6);    (* arrowhead x axis);
  aMove (85,33);
  WriteParamReg(ColReg1,0FFFFH);  (* setup color for drawbar *)
END drawaxis;

PROCEDURE drawbar (color:CARDINAL; Pat: PatternName; datavalue: INTEGER);
VAR
  cpx,cpy: CARDINAL;
BEGIN
  SelectPattern(Pat); (* dollar sign pattern *)
  color := color*4096 + color*256 + color*16 + color; (* bar color *)
  WriteParamReg(ColReg0,color);
  ReadParamReg (CurPtr1,cpx); ReadParamReg(CurPtr2,cpy); (* save CP *)
  rFilledRec (45,datavalue); (* draw the bar *)
  rMove (-12,4);
  WRITE (graphic,datavalue); (* label bar value *)
  aMove(cpx,cpy);  (* restore CP *)
  rMove (80,0); (* position for next bar *)
END drawbar;

BEGIN
  SelectPattern(Solid);  (* dollar sign pattern *)
  labelaxis;
  drawaxis;
  drawbar(10,CrossHatch,208);  drawbar(12,Arrow,110);  drawbar(8,Hand,220);
  drawbar(9,Triangle,240); drawbar(3,Hatch,296); drawbar(2,HalfTone,318);
END bar.
```

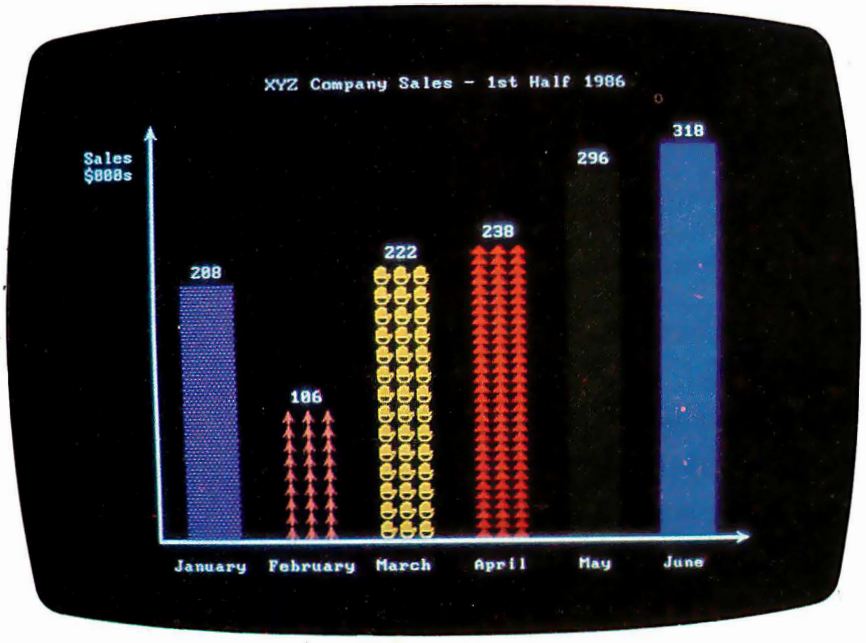

Photo 1: *This display is generated by the program shown in listing 1.*

and pattern RAM affect the basic operation of figure-drawing commands and should be set appropriately before a drawing command is issued.

PALETTE Module

These routines are used to access the BT450 palette D/A converter. Single colors or the entire palette can be read or written, either immediately or at the next vertical retrace. Since changing the color of an object is simply a matter of changing the corresponding palette entry, you can produce interesting effects like "flowing" water by dynamically reloading the palette.

GRAPHICS Module

This module contains all the ACRTC figure-drawing commands. Each command has a separate version for absolute and relative addressing, and they all use logical pixel *x,y* addressing; you don't have to translate to a physical address in the frame buffer.

Besides simply mapping directly to the

associated ACRTC command, some procedures perform useful error checking and pre/postprocessing. For example, the ACRTC on-chip PAINT command cannot handle overly complex figures, while the Turbo Graphix Toolbox PAINT command can.

Two parameters apply to specific commands. When drawing circles, ellipses, and arcs, you set the circular motion parameter to indicate the drawing direction as clockwise or counterclockwise. For the pattern and graphic copy commands, which move rectangular blocks of pixels, the CPScan parameter defines the scan direction during the block transfer. This allows you to slant or rotate an object during the transfer.

GRAPHMODES Module

Figure drawing is subject to various modes, which include operation, color, area, and edge modes. Like the drawing parameter registers and pattern RAM, you need to set up the drawing modes prior to issuing most commands. In simple applications, once you initialize the modes, you rarely need to modify them.

DATATRANSFER Module

Besides drawing figures, the other primary way to create a display is by moving bit-map images between host main memory and the frame buffer. (Since the frame buffer holds more memory than can be displayed on one screen, you can also "draw" pictures by moving them around within the frame buffer.) This module implements the ACRTC data-transfer commands designed for this purpose. Unlike the figure-drawing commands, the data-transfer commands use physical, instead of logical *x,y*, frame-buffer addresses.

The basic functions (read, write, clear, copy, and modify) are available, with or without DMA and "on the fly" masking and logical operations. The DMA option is used for large bit-map transfers (for example, loading an entire screen image), while the non-DMA versions are best for handling the transfer of a single word. A complete screen (640 by 480) DMA transfer between SB180 RAM and the frame buffer takes only a fraction of a second.

BITTEXT Module

One important requirement is to handle bit-mapped alphanumerics. Sometimes a word is worth a thousand pictures. The BITTEXT module makes writing text on the graphics screen as easy as writing it to a terminal.

FONTS Module

In conjunction with BITTEXT, the FONTS module lets you select multiple disk-based fonts. The fonts are loaded into an undisplayed area of the frame buffer. Font size and color are programmable, and you can add your own fonts as well.

PATTERNS Module

The ACRTC pattern RAM stores patterns (up to 16 by 16 dots), which are useful in two ways. First, all the figure-drawing commands refer to the pattern RAM when drawing. As each dot is drawn, pattern-RAM pointers are updated to point to the next dot in the pattern. This allows effects like dashed lines and tiling. Essentially, the "pen" can become a multidot pattern instead of just a single dot. Second, the pattern command simply moves the contents of the pattern RAM into the frame buffer, with optional rotation and slanting. This is useful for commonly used patterns like characters, cursors, and arrowheads.

BITMAPS Module

BITMAPS contains routines that let you transfer large bit-map images between the frame buffer and disk (floppy, hard, or RAM). Of course, it is quite possible to convert other machines' bit maps (like the Macintosh, Amiga, and Atari 520ST) for use on the GT180.

SCREENS Module

SCREENS eases the interface to the ACRTC display controller that manages the ACRTC split screens and window. It is easy to specify the screen's size and position as well as the display address of the contents. These routines can be used as the basis for a window manager, pull-down menus, status lines, and other visual interface techniques.

Using the Turbo Graphix Toolbox

The best way to get up to speed is to run through an application example. Let's use Modula-2 and the Turbo Graphix Toolbox to build a simple bar-chart program (see listing 1). The program accepts data values, legends, and bar color information and constructs a bar chart on the graphics screen. In this simplified example, the data values and legends are hard-wired into the program to keep the focus on the graphics routines. Obviously, your own chart program could adopt much more sophisticated data capture and scaling routines.

Since we are writing a program rather than a group of procedures, we don't need a definition module. After telling the compiler the name of the main module (bar), we use a series of FROM statements to specify which modules we are planning to use. IMPORT is used in conjunction with FROM to load specific functions and procedures from each module. We'll use a variety of Turbo Graphix Tools to complete the chart: text, patterns, filled rectangles, and others.

High-performance graphics hardware now available will let the SB180 and 8-bit software evolve to include graphics applications.

First, the labelaxis routine uses the bit-mapped text modules to label the graph, axis, and bar representing each month. Note the use of a disk-based font and the similarity of the bit-mapped text routines to the conventional terminal text routines. For instance, GotoRC locates the cursor at the correct line on the screen (depending on font size). Also, I extended the conventional WRITE ('text') statement—which prints text on the terminal—with the WRITE (graphic,'text') function that prints text on the graphics screen.

Next, the drawaxis routine draws the *x* and *y* axes. I used filled rectangles to make thick (three pixels wide) lines. This is easier than drawing three lines next to each other, which would achieve the same effect. However, unlike multiple lines, the filled rectangle approach works only for thick lines parallel to the *x* or *y* axis. The arrows at the end of each axis are a nice touch obtained by selecting the arrowhead pattern (with selectpattern) and then drawing it with the pattern command. Note how the same arrow pattern is used for both axes by changing the scan direction parameter of the pattern command.

Finally, each bar is drawn by calling drawbar with a data value and a color. Besides solid colors, you could use selectpattern to spruce up each bar with an illustrative pattern (see photo 1).

In Conclusion

As a stand-alone computer, the SB180/SB180FX, like most 8-bit systems, has traditionally been limited to alphanumerics. When 8-bit systems were introduced, a good graphics subsystem cost thousands of dollars, often more than the computer itself. Now that high-performance, low-cost graphics hardware is available, the SB180 and 8-bit software can evolve to include graphics applications. Using Modula-2 and the Graphix Tools, you can write software to tailor the SB180/GT180 for a variety of different graphics applications.

continued

Finally, a project as big as the GT180 could have been accomplished only with the help of many people. Foremost among them, I would like to personally thank Philippe Kahn of Borland International. His unwavering support for this project and 8-bit users in general demonstrates that he is a man of his word.

Experimenters

As with the the majority of Circuit Cellar projects, I encourage you to build them. To aid you in that endeavor, the Circuit Cellar BBS, (203) 871-1988, has been set up as an interchange for communication among builders and as a source for the various free software routines that complement these projects. With regard to the GT180, assorted graphics utilities are available for downloading.

Also, if you have been a supporter of the SB180 and are now interested in knowing more about the SB180FX, contact me and I'll send you a schematic and spec sheet. Finally, even though the SB180FX is not a BYTE project, I will offer support to BYTE readers who wish to build it. The object code of the monitor boot ROMs for the SB180FX and the original SB180 are posted on my BBS, and the BIOS will be sent in exchange for a picture of your handiwork. As with all the software supplied in this manner, it is completely free but limited to noncommercial personal use.

Circuit Cellar Feedback

This month's feedback begins on page 58.

Next Month

Next month's project features an infrared remote controller. ■

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

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.

The following items are available from

The Micromint Inc.
4 Park St.
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's 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 552-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
   SB180FX board fully populated with 512K bytes, SCSI chip, and software $599
   9.216-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's Turbo Modula-2 is also available for most CP/M Z80 machines. Contact Echelon Inc., 885 North San Antonio Rd., Los Altos, CA 94022, (415) 948-3820. 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.

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.