Getting the Bugs Out

Ross M. Greenberg

Two debugging tools: Is AT Probe worth $1500 more than Periscope III?

Your project is due in a week. All beta test sites have been calling dutifully, indicating no major problems. Suddenly, one site after another calls with the same problem. It seems their hard disk dissolved, and your program was the only one running when it occurred. Time to start debugging.

Either of two hardware-assisted debugging systems, AT Probe ($2495) from Atron and Periscope III ($995) from The Periscope Company, may be able to help. Both provide a means of isolating their respective debugging programs from "normal" code space. They offer some unique features as well, including the ability to stop your code in the middle of *anything*, debug it, and then continue processing; real-time tracing, which only ICEs (in-circuit emulators) could do until now; the ability to breakpoint in real time, based on access to any component within the machine; and displaying and experimenting with the prefetch buffer. (For a discussion of debugging, see "Finding the Culprit" on page 154.)

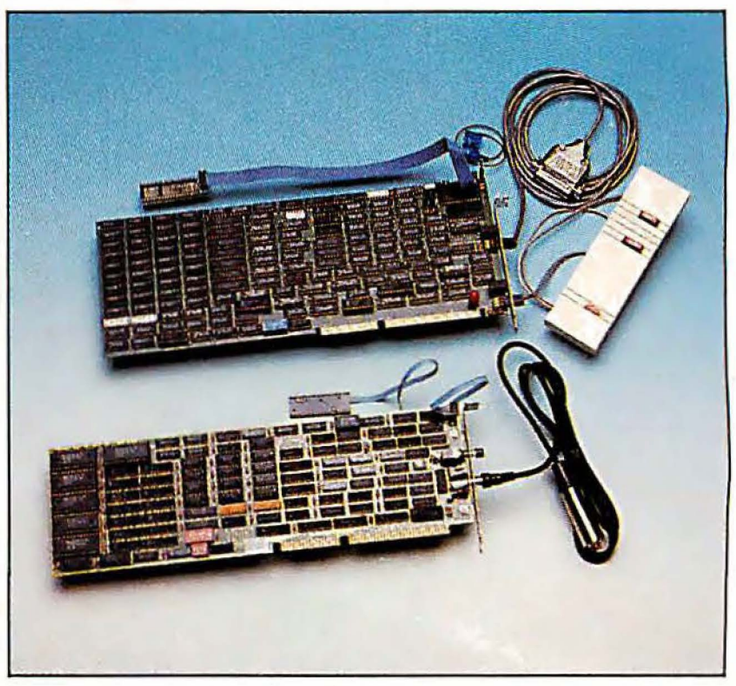
During the review process, I used the boards to solve a variety of bugs: a write access by a terminate-and-stay-resident (TSR) program outside its data space; problems with the initialization routines in two different device drivers; timing problems between incoming characters in an interrupt-driven TSR concurrent serial-communications program; spelunking through the innards of MS-DOS to determine real-time DOS usage of the PSP (Program Segment Prefix) and FCB (File Control Block) records; and using breakout switches to enter the debugger on an apparent total-system lockup. I might have been able to solve some of these problems without a hardware-assisted debugging board, but it would have taken significantly longer.

I tested the boards on a variety of high-speed AT clones with and without EGA displays. The EGA causes a problem with AT Probe because the debugger's 128K-byte RAM footprint conflicts with the memory space normally allocated for the EGA BIOS. (This problem is documented in the manual.) You can select another area, but the 128K-byte footprint is pretty large. At 64K bytes, Periscope III's DOS footprint is smaller than AT Probe's, so it ran on my EGA system without any modifications.

AT Probe

AT Probe 2.0 is a full-height, full-length, IBM PC AT card with 1 megabyte of write-protected on-board memory and the ability to be used in protected mode. It comes with a cable that plugs into the 80287 socket on the AT motherboard. Your 80287 plugs into the cable's piggyback socket. The debugger has both hardware and software components. Most of the software loads directly into AT Probe's own memory, but 128K bytes of system RAM is also required. Installation is easy; it takes only moments. (An XT version of the board is also available.)

A switchbox comes with AT Probe and is an intrinsic part of the package. It contains a Stop button, which you can use to break out to the debugger, and a Reset button, which you can optionally install to let you reset the computer. To install the Reset button, you crimp a connector (which is provided) to the power-supply reset wire (not available on all AT clones). This is a permanent change, and I'm surprised Atron didn't use a less intrusive method. However, if you need to remove AT Probe from your system, you simply pull its reset wire out of the connector, which then remains attached to the power-supply wire.



Hardware-assisted debuggers like AT Probe (top) and Periscope III (bottom) can significantly reduce debugging time—for a price.

Various add-ons to AT Probe are available: AT Source Probe ($395) lets you debug at the source code level in addition to the symbolic level; AT Software Performance and Timing Analyzer ($395) adds timing and performance measurement tools to the debugger; and Winprobe ($495) enables you to debug software designed to run under Microsoft Windows 2.0.

Three options are also available to optimize AT Probe for specific applications. The /ISO option ($395) lets you run the entire debugging process in the 80286's protected mode using no system memory. There are certain relatively obscure disadvantages to running the /ISO version (most having to do with the keyboard), but the beauty of running in protected space, where the debugger is almost invisible to your own code, is worth it.

The /PL option ($125) supports Plink86 Plus, a linker from Phoenix Technologies that resembles the standard Microsoft Linker but lets you use overlays in a more powerful way. The /PL option lets you set breakpoints in overlays

continued

AT Probe 2.0

Type
Hardware-assisted debugger

Company
Atron
12950 Saratoga Ave.
Saratoga, CA 95070
(408) 253-5933

Features
2K bus-cycle traceback buffer; 4 hardware breakpoints with 1 megabyte of on-board memory; 660K-byte symbol-table space; multiple software tools; breakout switch Reset button; 12-inch 80287 cable (longer lengths available); 2-foot cable on breakout switch

Size
5 by 13½ inches

Hardware Needed
IBM PC AT or compatible with 8-MHz/10-MHz AT hardware-compatible bus structure (one 16-bit slot required); one wait state required; 128K bytes of available memory

Software Needed
DOS 2.0 or higher; DOS 3.0 or higher recommended

Documentation
220-page AT Probe/AT Source Probe Manual

Options
AT Source Probe: $395
AT Software Performance and Timing Analyzer: $395
Winprobe: $495
/PL (Plink86 Plus) extension: $125
/ISO (protected-mode debugger): $395
/ISO and /PL extension: $495
/PRO (protected-mode operation): $975
/PRO and /PL extension: $1075

Price
$2495
386 Probe: $4195

Inquiry 887.

Periscope III 3.1

Type
Hardware-assisted debugger

Company
The Periscope Company
1197 Peachtree St., Plaza Level
Atlanta, GA 30361
(404) 875-8080
(800) 722-7006

Features
8K bus-event traceback buffer; 64K bytes of on-board memory with 32 hardware breakpoints; ability to use alternate display device; multiple software tools; 15-inch 80x87 cable (longer lengths available on request); 5-foot cable on breakout switch

Size
4⅕ by 13¼ inches

Hardware Needed
IBM PC, XT, AT, or compatible (16-bit bus slot required on AT), operating with one wait state at up to 10 MHz; 64K bytes of available RAM

Software Needed
DOS 2.0 or higher; DOS 3.0 or higher recommended

Documentation
218-page Periscope Manual; Quick Reference card

Price
8-MHz version: $995
10-MHz version: $1095

Inquiry 888.

that you haven't yet loaded from disk. Another option, /PRO ($975), lets you run in protected mode in memory above the first megabyte. Combinations of these options are also available: /ISO with /PL ($495), and /PRO with /PL ($1075). The particular software configuration I tested included the /ISO and /PL options and the AT Source Probe.

Probing for Bugs

Using AT Probe is straightforward. The command structure of the debugger requires a considerable learning curve because it is not similar to DOS DEBUG; however, the on-screen help facility in two command windows makes the process less painful than it would be if you had to turn to the manual all the time. The user interface, however, is simply a command-line interface.

When invoked, AT Probe looks for its configuration file, `PROBE.CFG`, which contains the base address at which to load the software, as well as the cursor- and screen-addressing information required to use a remote serial device as if it were a local console.

When you first load the debugger, it also looks for an `INIT.MAC` file, which can contain user-defined macros concerned with initialization. (An AT Probe macro contains a series of AT Probe commands.) As with any macro library, you eventually end up with your own set of specially designed tools, but AT Probe comes with a good set in `PROBE.MAC` to use for your initial debugging sessions. This set includes such macros as sector read, sector write, display program prefix segment, display registers, and display stack data.

Once within the debugger, you can specify the name of the file you want to debug and its associated symbol tables, as well as any additional macro files you wish to load. You can examine the code in its disassembled state, and symbol names are shown after their actual addresses to aid in your debugging. From past experience, I find that actually replacing the hexadecimal address with the symbol name is a bit more comfortable and closer to the original source code, but that's a matter of personal preference.

Debugging an application without a symbol table is more difficult, but since AT Probe lets you enter symbol names at specific addresses and thereafter displays those symbol names, exploring code fragments (even DOS itself) is easier than before. The board devotes 660K bytes to keeping your symbol table in its write-protected memory so that it won't be overwritten.

The heart of debugging, the setting of breakpoints, is an easy task with AT Probe. You can set them to occur when execution reaches a given location; when a particular range of memory is read, written, or fetched by the CPU; or when a specified I/O port (or a particular value for that port) is read from or written to.

The real power of AT Probe shows in its macros. The macro capability lets you execute an intelligent breakpoint request, such as: "Breakpoint when an area outside of this bounded area is written to, if and only if the code segment is within this range and the data value at this address is greater than a particular value and less than this value." You can make a macro just about any length you like, and it can call other macros (up to a nesting level of 5) with parameter passing.

The macro processing speed isn't as fast as I'd like. An intelligent breakpoint request runs at full speed on AT Probe, but once it reaches a breakpoint and starts executing in the CPU, processing slows down. When you know this, you will find yourself setting breakpoints a bit more sparsely.

You can have a total of only four hard-

continued

Finding the Culprit

Fixing a bug, once found, usually takes only minutes; finding it, however, can take minutes or weeks. A great number of software-only debuggers are available, all of which let you examine, step-by-step, each instruction the CPU executes, and compare the actual results with the results you expected. But, for a variety of reasons, hardware-assisted debuggers can make the process go much more quickly than software-only debuggers can.

One of the most common software debuggers is DEBUG, included with MS-DOS up to version 3.3. DEBUG is very useful for examining small assembly language programs or for resolving simple problems. However, it's woefully inadequate for any but the most rudimentary debugging. To debug program code easily, you need to be able to examine it with a symbolic listing; for example, `INC [TIMER_TICK]` is inherently clearer than `INC [ODCA]`. The first conveys at least part of the meaning the original programmer intended. To debug easily, you need to see the source code, or at least the symbol map. Many symbolic debuggers contain options to enter these symbols as required.

Other capabilities also affect the debugging process. When single-stepping through your code, for instance, a debugger needs to be able to set the "single-step" bit in the processor flag word. This bit generates an interrupt after each instruction, which the debugging program then processes, and it usually results in a display of the next instruction to execute (or the previous one) and the current setting of each register. If this information is stored in a traceback buffer, then you can see how your code arrived at the current point of execution. The larger the traceback buffer, the more instructions you can see, and the easier debugging becomes.

Determining when to examine that traceback buffer is more of an art than a science; being able to set breakpoints to stop execution of the current program and enter debug mode makes the process easier. The number of breakpoints a particular debugger allows is an important consideration, since the number and different types of breakpoints you use let you execute more of your code and less single-step tracing. The ability to issue breakpoints when your program reads or writes to a particular area of memory, when a register equals a certain value or range of values, or when your program accesses a particular I/O port is an important aid to debugging.

However, if the problem you're trying to find stems from the interaction of various hardware components over which you have no control (such as a timer tick occurring every 1/18 second or a conflict of serial-port interrupts), then even the niceties of Microsoft's CodeView (a full-screen symbolic debugger) may not suffice. It's not that software debuggers don't have the ability—most do; it's simply that tracing or single-stepping through your code can be terribly slow, because the CPU is executing perhaps hundreds of debugger instructions each time it executes one of your program's instructions. Trying to use the same CPU chip to process all these "intelligent" breakpoints, to manage a traceback buffer, and to protect your code and the debugger code from each other is a difficult, if not impossible, task.

Finally, sometimes the software debugger itself has an adverse effect on what you're debugging (since the position of your program in memory may depend on where the debugger is placed). Debugging may even be impossible, since the errant pointer causing your problem may actually modify a portion of the memory space occupied by the debugger—thus causing the debugger to crash mysteriously. In these situations, hardware-assisted debuggers like the ones reviewed here are your only choice.

ware breakpoints active at any given time. I found that this wasn't sufficient in many cases, especially when I was trying to trace carefully through the many different cases on a C `case/switch` statement. I don't like having to outsmart limitations a debugger places on me.

Fetch and Carry

For the CPU to be as fast as possible, it automatically reads the next few instructions into a small on-board buffer while it executes its current instruction. In most programs, after a given instruction has run, the instruction most likely to run next is the one that follows it; since that instruction is already in the prefetch queue, little access time is needed except to read or write the operands.

Since prefetch is used only for instruction execution, it might be useful to set a breakpoint on prefetch into any area of memory that should never be executed but can be read or written to, such as your data-segment area or your low-memory interrupt table.

AT Probe's traceback buffer is 2K bus cycles, or events, in length. Atron calls this a *dequeued* buffer. You won't see instructions that have been fetched but not executed unless you specifically ask to see them. This is an important consideration when looking at the traceback buffer; it can be confusing to see instructions that were never executed.

A few features of note are AT Probe's ability to debug using Microsoft C's local stack referencing, its complete 80287 support, an on-line editor and notepad (in AT Source Probe), the ability to follow a nested calling convention backward on the stack, and the ability to re-enter data items by data type instead of hexadecimally, byte by byte.

One caveat, however, regarding AT Probe: One of the manual's appendixes contains a note about executing breakpoints while in a non-reentrant BIOS call, such as the keyboard- or monitor-service routines. Atron says that since AT Probe uses these routines, you could crash the system if you breakpoint within them.

Writing a Wrong

I used AT Probe to find a `write` access by a TSR outside of its data and code space. First, I defined a write-to-memory breakpoint for each of two memory-address ranges: one before the initial TSR code segment, and one after the last TSR segment. Both DOS and foreground programs, such as `COMMAND.COM`, normally access these areas of memory. Therefore, the debugger had to filter out all such valid `write` accesses.

Using a macro to set the breakpoints, I checked the code segment after each breakpoint to see if it fell within the TSR range. If it didn't, I resumed program execution; if it did, I knew I had found the culprit and could use the real-time traceback buffer to determine how I had gotten to the instruction causing the problem. Again, the macro didn't execute as fast as I'd like, but with some fine-tuning of its logic, I managed to speed it up.

Periscope III

Periscope III 3.1 is a full-length hardware-assisted debugging card with 64K bytes of write-protected RAM, an 8K-bus-event real-time traceback buffer, 32 hardware breakpoints, and a remote breakout switch. It requires an IBM PC XT, AT, or compatible with 64K bytes of available RAM. (Since the board is not a full-height card, it runs in the XT as well as the AT.) Periscope III comes in two

continued

models: an 8-MHz version ($995) and a 10-MHz version ($1095). It requires DOS 2.0 or higher, but DOS 3.0 or higher is recommended because DOS 2.0 and 2.1 contain some bugs involving improper changes to the stack; The Periscope Company warns that if your breakpoint happens to occur at just the right—or wrong—moment, you may have trouble. Note that this is true for AT Probe, too.

Like AT Probe, Periscope III has a cable that plugs into the 80287 socket on the motherboard, and an existing 80287 plugs into the cable's piggyback socket. A plug socket at the end of the card attaches to a breakout button that immediately enables the debugger.

Once you install the hardware (which takes about 5 minutes), you must install the software—also a rapid and straightforward process. You can run the software portion of the debugger without the hardware. After you install the software, you're ready to run a TSR called PS.COM, which takes command-line or indirect file switches. One switch lets you specify which area of memory the board is to use. Other switches let you specify the size of the symbol table, various window sizes, and other setup parameters.

No software extensions are available at this time, including those that allow for protected-mode operation; thus, most of the symbol table is visible in system memory. Although the debugger program itself is write-protected and is therefore safe from typical programming mistakes, the symbol table and other important tables are not protected. And since these tables are stored in system memory, you may not have enough memory to debug a large program with a large symbol table. This was not a problem in my testing, however.

Up Periscope

Once you load PS.COM, it sits quietly in the background until you call it. You can access it in either of two ways: through the breakout switch, or through RUN.COM, which automatically loads your program and its symbol table. Once your program is loaded, you are in the debugger, ready to debug your code.

Part of Periscope III's power is, surprisingly enough, in its user interface. You can choose from a variety of screen windows. I found that having two data windows, one stack window, a register window, and a disassembly window usually sufficed for my needs. Changing window sizes is an easy operation; for example, if you want two data windows with lengths of 4 lines and 6 lines, respectively, you enter /W D:4 D:6.

There is a not-so-subtle difference between Periscope III and AT Probe: AT Probe feels as if it were designed for programmers, while Periscope III feels as if it were designed by programmers. This shows through not just in the display of the various windows, but also in the command set and options, designed for programmers who want ultimate control over their debugging environment. For example, to issue a breakpoint when the byte, COUNT, equals 08, you would enter BB COUNT EQ 08. For AT Probe, the command would be BP 1=.COUNT W DATA 08.

Periscope III provides breakpoints on byte, code, interrupt, line, memory, port, register, user test, word, and exit. AT Probe has many of these abilities, but its interface uses a less intuitive format.

The myriad breakpoints that Periscope III allows also reflect this philosophy: You can set up to 16 hardware breakpoints, allowing read, write, execute, and fetch breaks, and including breakpoints above the first megabyte of memory for AT-class machines. You can also set breakpoints when the data on the bus meets a particular value, or when input or output to a port occurs (you can specify up to 16 different ports). These breakpoints can be set directly on Periscope III's board. Finally, you can set a *pass count*, which breaks out only after the breakpoint has been executed a certain number of times; for example, "break on the fifth write to this screen location."

You can also set software breakpoints. Aside from those breakpoints that are put in Periscope III hardware for speed, you can set additional interrupts when a particular byte, word, or register meets certain tests (less than, equal to, or greater than a specified value), as well as when a particular interrupt executes, when a particular line of source code runs, when returning from the currently executing routine, or when a user-supplied (and previously loaded) assembly language routine returns a certain value.

With Periscope III's "go using monitor" mode (GM command), some of these software breakpoints can work in combination with the hardware breakpoints. Thus, the program can run at full speed until it reaches a breakpoint on the board. Control then transfers to the board to determine if the specific condition exists, such as "break on register value" (BR breakpoint). If not, the code resumes execution immediately. If the conditions are met, the breakpoint executes, and you can start debugging.

One real disappointment is that "break on register value" allows only one breakpoint test for each register. Thus, testing for a write access to a range of memory with a double range check on the CS register isn't as easy to accomplish on Periscope III as on AT Probe. (AT Probe's macro capability can handle this.)

Periscope III lacks macro capability, but, with its USEREXIT capability, you can make tests as arbitrarily complex as you wish. A USEREXIT is user-written assembly language code (USEREXIT.ASM) with which Periscope III can interface. It is loaded as a memory-resident program, called USEREXIT.COM, that uses an available interrupt. Of course, you'll have to debug the assembly language routines that make up the USEREXIT routine itself.

Like AT Probe, Periscope III has traceback-buffer capabilities. However, Periscope III's traceback buffer is 8K bus events long (a bus-event record is 48 bits wide), so it can hold substantially more information than AT Probe's 2K-bus-cycle buffer. You can set up the buffer to capture 8K events before the breakpoint, 4K events before and 4K events after the breakpoint, or 8K events after the breakpoint. However, much of Periscope III's traceback information consists of prefetch instructions. As such, except for the obvious ones, like jmp, call, ret, and int, the trace buffer contains instructions that may not have been executed but were only in the prefetch buffer. This makes debugging more difficult.

Periscope III comes with a number of auxiliary programs, including one to let you use the map-file output from Plink86 Plus, and another to run through assembly language source code and make each symbol public. Since the more symbols you have, the easier the debugging becomes, the PUBLIC program should be a welcome addition to any library.

Driving Devices Crazy

Debugging a device driver can be difficult, especially since the device driver is loaded before you can load your debugger. Periscope III's distribution disk contains a program, SYSLOAD, that lets you load the important parts of Periscope III as device drivers so you can debug the device-driver initialization routine. When I combined this with Periscope III's ability to perform a "short boot" (which usually

continued

leaves the interrupt vectors loaded), debugging a device driver became as easy as debugging any other program.

The problem I had was that the serial-communications device driver was crashing the system somewhere in the initialization routine—sometimes. At other times, and at high data transfer rates, characters were being dropped. The worst kind of bug is the kind that isn't always repeatable.

By setting breakpoints on input or output instructions to ranges of ports, I found that the device-driver initialization routine was turning interrupts on too soon. By using Periscope III's user-exit capability, I found that the time used to return after a polling loop and before generating the End-of-Interrupt instruction was sufficient for a character to arrive when receiving at 115K bytes per second. Without a hardware-assisted debugging board, either condition would have been difficult, if not impossible, to discover.

The Competitive Edge

Determining which debugger is better is a difficult task. There is no clear and simple benchmark that you can base a judgment on. Once you find a bug with one debugger, it's gone, and it's impossible to replicate the problem. (But, sadly, another bug always lurks around the corner; there's never a shortage of them.)

If you're developing Microsoft Windows programs; if you're using the overlay capability of Plink86 Plus substantially; if your code is very large (greater than 400K bytes); if you have a huge number of symbols; or if you're writing a protected-mode application (and with OS/2 on the horizon, protected-mode operations are going to become increasingly important), then you need the support offered as options for AT Probe 2.0.

However, the interface on Periscope III is easier to work with than AT Probe 2.0's; the capabilities, such as the number and type of breakpoints, are substantially better; and the command summary is, to my way of thinking, closer to the debugging needs of a programmer. And if you're a small developer with limited funds, Periscope III's lower price becomes paramount.

But all this is changing. New releases on both products are forthcoming. The features in the new releases clearly indicate how acutely each company is aware of the competitive edge the other has to offer.

The Periscope Company says that Periscope III 4.0 (due out this month) will support Microsoft Windows, Plink86 Plus's overlays, and Microsoft C's local symbols. Periscope IV, due out this spring, is planned to support the 80386 as well as the 80286. Atron says that the next release of AT Probe, also due out this month, will contain a user interface similar to that of Microsoft CodeView. Atron's 386 Probe for 80386 systems has recently been released. It's an interesting race to watch.

Both AT Probe and Periscope III serve admirably as debuggers. Each does quite a bit more than its advertising represents, and each also has major enhancements in the works. But does AT Probe make up the $1500 difference in price (almost $2000 if you include AT Source Probe) with added programmer productivity?

The answer isn't as clear-cut as I'd wish. If you don't need the special capabilities AT Probe has that Periscope III doesn't have, then, no, AT Probe isn't worth the difference in price. If, however, you do need them, and you need them badly, then the price doesn't matter as much. ■

Ross M. Greenberg, owner and chief executive officer of Software Concepts Design in New York, is a computer consultant and software designer.