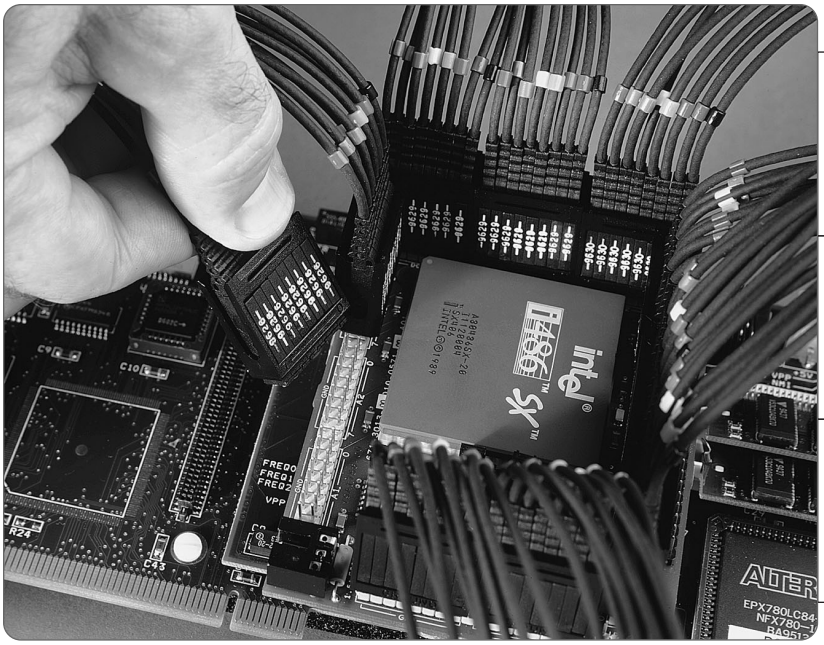
Access to the Target System



When considering how to best provide access to the target system, embedded system developers often have two choices: on-chip emulation or ROM monitors. There are advantages and disadvantages with either approach which must be carefully considered when building an integrated debug environment.

On-Chip Emulator vs. ROM Monitors

For processors that don't have on-chip emulation support, a method that is often used is to couple the TLA700 logic analyzer with a ROM monitor that resides in the target system ROM and communicate with the software debugger running directly on the TLA700's integrated Windows® 95-based PC.

A ROM monitor provides run control of the target system. Typical features include setting breakpoints, single-stepping, displaying program code, display and modification of program data, stack backtrace, as well as loading code into the target system. It must be coupled with a logic analyzer for real-time trace.

A ROM monitor consists of two pieces. The software debugger runs on the developer's computer. It's fairly tightly coupled with the program source code and symbol database to provide a high-level view of the program execution. The other piece is the monitor program that actually controls the target system and provides access to its resources.

The monitor program typically runs in ROM on the target system, which is why it's called a ROM monitor. Compared with the newer on-chip emulators, the ROM monitor offers essentially the same feature set. However, their different implementations lead to some subtle but important differences in behavior that can have a substantial impact on the behavior of a real-time embedded system.

ROM-based monitors are implemented as a set of interrupt and exception handlers that cause the developer's program to be suspended while the monitor program runs. Once the monitor program is in control, it can access target system resources and provide debugging features such as viewing memory or processor registers.

This implementation produces intrusion into the target system on several levels. First and most obviously, the ROM monitor takes up address space. This is often not a problem because ROM monitors are typically fairly small (usually less than 16 KB), and can usually be relocated to accommodate a particular system's memory configuration.

Secondly, the ROM monitor uses interrupt and exception vectors which would normally be available to the target system. This may be useful in the early phases of development, before the program has its own interrupt and exception handlers. A more mature system may run into conflicts supporting its own interrupt structure along with the monitor program.

On the other hand, ROM monitors do have certain advantages compared with on-chip emulators. Because the ROM monitor is integrated with the application software and perhaps an operating system, it may be able to provide additional insight into the target system. In particular, it may be better able to support multi-tasking debugging features such as task-specific breakpoints as well as display and modification of operating system data structures.

Also, it's important to note that the ROM monitor runs at some priority within the priority scheme of the target system's processor. This provides the developer with a tremendous amount of flexibility for debugging real-time systems while handling external interrupts. The monitor priority can be lowered to allow critical interrupt routines to be serviced while the monitor program is running. This may be absolutely essential for debugging certain real-time applications, which cannot hold off external interrupts without crashing the target system. Conversely, the monitor priority can be raised so that breakpoints can be set in lower priority interrupt handlers.

In contrast, many on-chip emulators do not allow any external interrupts to be serviced while debugging the program. While in background mode, all normal processor activity is suspended. This may not be acceptable for a target system that must service interrupts or that is relying on bus activity to refresh dynamic RAM. On the other hand, on-chip emulators don't require any address space or exception vectors. This autonomy from the target system program can be quite useful. An on-chip emulator never has to be designed into the target system. It will never be taken out because of board space requirements or a resource conflict. It's always there if you need it, even after a design moves into production.

The Logic Analyzer and On-Chip Emulators vs. ROM Monitors

Neither on-chip emulators nor ROM monitors are capable of making real-time measurements or providing a real-time trace. A logic analyzer is required to observe execution in the target system without interrupting the executing program. In order to correlate the code that the embedded software developer wrote with the executed code that was acquired by the logic analyzer, the TLA700 Series incorporates High-Level Language (HLL) source code support. This combination allows complex trigger setups to be specified in terms of high-level program symbols. The resulting data displayed by the TLA700 Series shows the HLL source code that were executed by the target system.

The connection between the logic analyzer and either the on-chip emulator or ROM monitor can be made more powerful by configuring the TLA700 logic analyzer to interact in real-time with the target system. The debugger/on-chip emulator can be programmed to assert a signal that the TLA700 will recognize through either its Trigger In or Signal In port. Conversely, the TLA700 can assert a signal to interrupt or stop the software debugger/on-chip emulator. This can be useful for entering the on-chip emulator's background debug mode or the ROM monitor after a sequence of function calls with a specific set of parameters are passed on the stack.

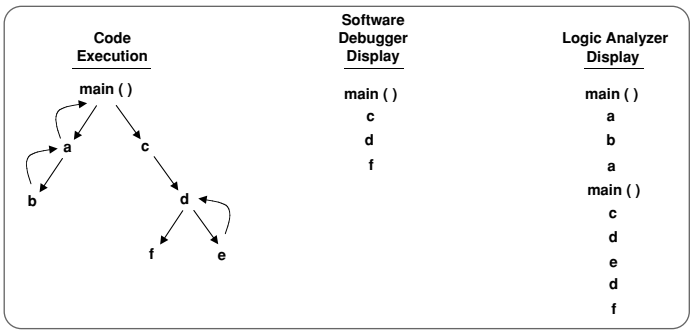
Although the hardware breakpoints offered by a logic analyzer are the least intrusive form of breakpoint, it's important to note that they too have a set of limitations. The most obvious problem with hardware breakpoints is called "skid." It takes time for the trigger event to propagate through the logic analyzer and back to the processor. The result is that the break occurs a few clock cycles after the trigger event. Software breakpoints are required for precise breaks in execution.

Hardware breakpoints provided by the logic analyzer can also be troublesome if the processor implements some sort of pre-fetching scheme, such as pipelining or caching. Since hardware breakpoints are based on external bus-level activity, the breakpoint occurs in response to the instruction or data being fetched, not when it's actually executed. If an instruction is fetched, but never executed, a hardware breakpoint may occur at the wrong place.

Breakpoints: Software vs. Hardware

The key to embedded system debug is having control of the target system's processor. The breakpoint is the key mechanism to gaining control of a running target system that implements all monitor functions.

There are two types of breakpoints: hardware breakpoints and software breakpoints. Hardware breakpoints, which require either dedicated circuitry or external instrumentation, are used to stop processor execution when the processor makes an external access; e.g., "break on memory read." Hardware breakpoints, however, cannot tell the developer if a particular portion of code that was fetched was necessarily executed. Software breakpoints, which use the exception processing instructions of the target processor (e.g., TRAP instruction), are used to stop processor execution only when executed by the processor. Software breakpoints, however, cannot tell the developer if an external access occurred.



▶ ***Figure 1.*** *Real-time trace capability of a logic analyzer.*

To successfully set a software breakpoint, the software debugger must be able to write to the memory location specified. Therefore, software breakpoints cannot be set in ROM. Also, there will be unexpected results using software breakpoints in the presence of an instruction cache. If the memory is changed, but not the cached copy, the breakpoint will not be executed.

To alleviate some of the pitfalls of software breakpoints, more and more processors include breakpoint capabilities as part of their on-chip emulation capabilities. These typically take the form of special purpose debug registers that can be loaded with an instruction address. When an instruction at that address gets executed, the software debugger is entered. This allows setting breakpoints in ROM-based systems.

On-chip emulator breakpoints are fairly limited in number. On some processors, the debug breakpoint exception occurs after the instruction is executed. This is in contrast to a software breakpoint. When a software breakpoint is hit, the processor's program counter is backed up and the original instruction is restored in memory. The effect is that execution is interrupted before the instruction at the specified address. On-chip emulator breakpoints may also support setting breakpoints based on data values rather than instruction addresses. They may also include some basic status information such as read/write and data access size. Data breakpoints offer some of the capability of hardware breakpoints which is what a logic analyzer is often used for; however, the data breakpoints offered by a logic analyzer can be very sophisticated compared to the very simple capability offered by on-chip emulation data breakpoints.

Additional Logic Analyzer and On-chip Emulation Support/ROM Monitor Measurements

In addition to full-speed or real-time trace and hardware breakpoints, logic analyzers provide two other key measurements that complement on-chip emulators/ROM monitors.

Stack Trace Back. Software debuggers can provide the calling sequence of any functions that have not finished executing; i.e., their calling address still resides on the stack. Once the function is off the stack, however, any information is lost to the software debugger. Given the real-time trace capability of a logic analyzer, it can display the exact calling sequence of all routines, whether still on the stack or not (see Figure 1).

T**iming of Code.** Newer logic analyzers have built-in counter/timers that run at full-speed, thereby providing timing resolution down to several nanoseconds (e.g., 4 ns on TLA700). If your code has specific execution requirements such as interrupt service handlers or algorithms that must execute within a specified time window, you can use the logic analyzer's high-speed counter/timers to verify if your code meets specifications. Coupled with a software debugger, you can change the test conditions, thereby "stress-testing" your software to see how it performs under a variety of conditions.

Conclusion

The digital system developer should give careful consideration to the impact of how debug tools will access their target system. Today's modern logic analyzers, such as the TLA700, support a variety of hardware access methods such as on-chip emulation or ROM monitors. The integrated debug approach offered by the TLA700 provides both hardware and software breakpoints plus critical measurements that are important to the successful debug and validation of modern digital systems.

Contact Tektronix:

ASEAN Countries & Pakistan (65) 6356 3900

Australia & New Zealand (65) 6356 3900

Austria +43 2236 8092 262

Belgium +32 (2) 715 89 70

Brazil & South America 55 (11) 3741-8360

Canada 1 (800) 661-5625

Central Europe & Greece +43 2236 8092 301

Denmark +45 44 850 700

Finland +358 (9) 4783 400

France & North Africa +33 (0) 1 69 86 80 34

Germany +49 (221) 94 77 400

Hong Kong (852) 2585-6688

India (91) 80-2275577

Italy +39 (02) 25086 1

Japan (Sony/Tektronix Corporation) 81 (3) 3448-3111

Mexico, Central America & Caribbean 52 (55) 56666-333

The Netherlands +31 (0) 23 569 5555

Norway +47 22 07 07 00

People's Republic of China 86 (10) 6235 1230

Poland +48 (0) 22 521 53 40

Republic of Korea 82 (2) 528-5299

Russia, CIS & The Baltics +358 (9) 4783 400

South Africa +27 11 254 8360

Spain +34 (91) 372 6055

Sweden +46 8 477 6503/4

Taiwan 886 (2) 2722-9622

United Kingdom & Eire +44 (0) 1344 392400

USA 1 (800) 426-2200

For other areas contact Tektronix, Inc. at: 1 (503) 627-7111

For Further Information

Tektronix maintains a comprehensive, constantly expanding collection of application notes, technical briefs and other resources to help engineers working on the cutting edge of technology. Please visit **www.tektronix.com**

03/02 OA/XBS 52W-12479-1

Tektronix
Enabling Innovation