Debug strategies needed for SoC, reliability

Bernard Cole - March 28, 2000

Debug strategies needed for SoC, reliability
No matter what the environment or what the application-desktop, embedded or Net-centric-software developers face many common challenges. But embedded designs in particular pose a number of unique problems. And in the emerging embedded Internet environment, developers will have even more problems to deal with as far as debugging code and ensuring reliability of the end system.

What all developers share in common, application environment notwithstanding, is that during debugging they must re-create the exact conditions, if possible, that reveal the fault seen during testing. Not only must the code execute the same instruction sequences, but the debugging tools must provide the developer with execution visibility and control, yet not affect the execution behavior of the program as it relates to latency and degree of deterministic response.

Compared with desktop computers and servers, however, embedded systems are some of the most sophisticated and complex systems being developed today. Moreover, they face constraints never seen in other realms, including concurrent designs, real-time constraints, environments where the code being developed resides not on the host but on another target system, device control dependencies and distributed hardware architectures.
Hardware probe techniques that collect data on applications running in the processor are obsolete, says Advanced Micro Devices' Daniel Mann. Software probes and code instrumentation can get the processor itself to supply information on debug interfaces.

To cap it all, perhaps one of the most difficult problems in embedded systems using the latest processors or customized system-on-chip (SoC) devices is figuring out how to gain visibility into the inner workings of the device and how to do non-intrusive testing.

"The embedded processors coming to market over the next few months will make the traditional development methodology obsolete in the coming years," said Ron Stence, a systems engineer in the Advanced Vehicles Systems Division at Motorola Inc. (Austin, Texas). In this week's Focus section on hardware/software debug strategies, Stence discusses the Nexus 5001, an emerging debug interface standard.

"The external address and data buses have virtually disappeared from the 8-bit processors of today," write Mark Heilpern and Norman Bartek in their article. They are both field engineers at Diab-SDS, a subsidiary of Wind River Systems Inc. (Alameda, Calif.). They believe that 16- and 32-bit embedded processors are reducing the use of external buses in favor of more specialized I/O functions. At the same time, integrating the memory on chip leaves little room for the external buses.

And with their disappearance, many traditional hardware probe techniques for collecting information on the application as it is running in the processor are now obsolete, according to contributor Daniel Mann, senior member of the technical staff at Advanced Micro Devices Inc.
"Now the focus is on software probes and code instrumentation to get the processor itself to provide the information needed and on new debug interfaces to enable that process," Mann said.

Motorola's Stence notes that due to the prevalence of new features, higher-pin-count packages, and increasing speed and complexity, few debug interfaces for embedded processors have kept up with improvements in technology. In combination with the emergence of newer debugging techniques appropriate to this new environment, the processors and SoC devices will have to incorporate ever more sophisticated debug capabilities on chip. This in turn will provide additional impetus to develop new techniques to find and eliminate bugs efficiently, effectively and non-intrusively.

In his contribution, Stence points out that the issue of non-intrusive debugging remains one of the toughest hurdles. "Any technique used to raise execution visibility or provide for program control must not interfere with the operation of the processor or the application it is running," he said.

That is because embedded applications have strict timing requirements and any intrusion that slows down the processor will nullify the test under way, Stence said. Typically, such approaches start by the instrumentation of the code-inserting probes into the program and rewriting certain constructs before submission to the compiler-to gather run-time information and control program execution.

After the code is instrumented, the object code is linked with the rest of the debugging tool chain and then executed under test. While that approach yields useful information, the additional code can have a serious impact on the execution behavior of the program, limiting its use in the testing of real-time embedded systems.

One aim of the many proprietary and architecture-specific on-chip debug interface protocols, AMD's Mann said, is to make such code instrumentation more effective and less apt to affect the execution of the program under test. Mann's article reports that his company's AMDebug interface has a number of features to simplify the task of profiling and code instrumentation. Its on-chip Trace Cache, for example, is primarily used to trace an executing program's instruction flow. But in another mode, said Mann, it can gather performance-profiling information instead.

As effective as these architecture-specific debug interfaces are, Stence said, they drive up the cost of development, forcing developers to make a choice: save money and focus development on one architecture (thereby running the risk of not having the processor necessary for the application); or invest in several architecture-specific tools to support development on multiple processors. Moreover, now tool vendors must devote even more resources to developing tool chains that are even more specific to particular processors.

The Nexus 5001 standard, he said, eliminates the need for such choices. "Developers receive the majority of the benefits from the Nexus 5001 standardization," Stence said. "Tools are available and checked out when the engineers need them at first silicon. These tools are based on standards, so there is less risk of quality issues for a new processor tool set. The tools have a standard set of features that the customer can count on, thus decoupling the tool from the processor selection process."
As developers grapple with the problems of higher integration and visibility into the internals of the system-on-chip, less attention has been given to the changing environment in which those embedded devices must operate.

In applications such as information appliances, Internet-enabled embedded devices, wireless PDAs and set-top boxes, more of the external environment will have to be factored into designs, said Michael O’Brian, chief executive officer at GoAhead Technology (Bellevue, Wash.). Alternatively, he said, significant investment will have to be made in a whole new category of tools to monitor and manage a network of cooperating devices to ensure that each one is operating correctly and that groups of embedded devices are working together.

The loosely (or barely) coupled multiprocessing nature of arrays of devices linked by a mechanism such as Universal Plug and Play or Sun's Jini has little in common with the tightly coupled nature of many embedded multiprocessor designs. Still, say some developers, many of the debugging and monitoring problems faced there will likely occur in this new distributed computing environment, and solutions will have to be found.

Since a distributed system involves more than just one point of control, traditional sequential monitoring and debugging techniques such as tracing and breakpoints will have to be extended. Also, communications delays between cooperating embedded devices make it difficult to determine a system's state at any given time.

Another fact that must also be faced, according to some, is that distributed computing systems as presently constituted are inherently nondeterministic. This means that two executions of the same system may produce different, but valid, orderings of events, making it difficult to reproduce errors and to test possible situations in which they might occur.

Finally, unlike traditional embedded systems, all such distributed systems bear an uncertainty principle that must be considered. In a classical embedded environment, the behavior of a program is not affected by the amount of elapsed time between the execution of two successive instructions. For example, according to one independent consultant, a debugger can interrupt a sequential process at a breakpoint without affecting the process' later execution. But in a distributed computing environment of cooperating embedded devices, stopping or slowing down one process may alter the behavior of the entire system.