# In-System Silicon Validation and Debug

**Miron Abramovici**
DAFCA

*Editor's note:*
With the number of design flaws on the rise, the best way to be prepared is through a comprehensive debug infrastructure. This article presents such a facility, comprising on-chip debug hardware and off-chip software.
*—Erik Jan Marinissen, NXP Semiconductors*

**SILICON VALIDATION—PROVING A** chip works correctly at speed and in system under different operating conditions—is always necessary, even for a ''perfect'' design. Silicon debug—finding the root cause of a malfunction—is necessary whenever a design is not flawless. First-silicon validation and debug require a labor-intensive engineering effort of several months and have become the least predictable and most time-consuming part of a new 90-nm chip's development cycle. Lack of adequate tools and automatic procedures is a big factor in this bottleneck. The difficulty of silicon validation will increase at 65 nm and below because existing ad hoc methodologies don't scale with the unprecedented levels of SoC device complexity.

Even the most sophisticated SoC design methodology cannot fully account for all the parameters that affect silicon behavior, or for all logic corner cases occurring in the life of a chip working at speed and in system. For example, the simultaneous occurrence of two unlikely events might not be anticipated presilicon, so it is never simulated or analyzed; however, when it occurs in system, it can cause unexpected behavior. Presilicon verification methods—simulation, emulation, FPGA prototyping, timing analysis, and formal verification—don't address many deep-submicron problems that occur in the actual device.

In just one of many possible examples, consider a scenario in which an unanticipated circuit usage model is exercised in system, causing more extensive system activity in a confined area of the chip. This increases the local on-chip temperature and introduces additional delays in that region, which consequently affects the timing of a critical path, resulting in erroneous logic behavior. Such a problem, first revealed in system, is nearly impossible to detect presilicon. Many other signal integrity problems caused by crosstalk, noise, process variations, power drops, and design marginalities can be similarly difficult to anticipate presilicon.

Because complete system-level verification of a complex SoC at or below 90 nm is not feasible presilicon, in-system, at-speed silicon validation has become an essential step in the design implementation methodology.

## In-system operation

Operating in system, in its intended environment, a newly manufactured chip has its first opportunity to interact with other chips on a system board while running software and operating at its target frequency. Many aspects of SoC behavior can be fully validated only in system, including hardware-software integration; system configuration options; adaptive control of temperature, voltage, and power; and D/A interactions. At-speed, in-system usage under stress conditions introduces many new functional patterns and explores deep states and corner cases not previously encountered, thereby exposing errors that escaped presilicon verification. Moreover, because of severely limited observability and controllability of a chip's internal dynamic behavior, locating such problems is far more difficult and expensive than in presilicon or on a tester.

Unlike tester-based experiments that are fully repeatable, in-system operation is not completely deterministic, because of unpredictable interactions among independent events such as external interrupts,


0740-7475/08/$25.00 © 2008 IEEE Copublished by the IEEE CS and the IEEE CASS

irregular network traffic, bus arbitration, and transactions involving asynchronous clock domains. This nondeterminism makes many problems appear as intermittent and severely complicates silicon validation and debug.

Nondeterministic operation and the lack of in-system control over stimuli applied to the chip combine to create another difficulty: time-specific expected values are no longer known. Even if complete observability were possible, tester-style analysis such as ''at vector **k**, values of BUSX should be YYY'' would not work in system.

Yet another complicating factor is the difficulty of determining the nature of a problem. In addition to functional problems and timing errors, SoC operation might be affected by defects that eluded manufacturing tests and are detected only in system. Taking a chip that fails in system through manufacturing testing often results in a ''No Trouble Found'' outcome because the operational conditions that enable such failures cannot be reproduced on a tester. This is often the situation with chips that fail in the field. Performing in-system diagnosis seems to be the only way to locate such a defect.

We have developed a new silicon validation and debug solution at DAFCA that overcomes all these difficulties. This article assumes familiarity with the basic concepts and techniques of validation and debug. Vermeulen and Goel provide a good overview of prior work in this area.[1]

## A solution based on reconfigurable instruments

We have introduced a new approach to in-system silicon validation and debug.[2,3] In the presilicon stage of this methodology, instrumentation tools guide the insertion of reconfigurable instruments into the RTL model of the SoC and generate an instrumented RTL model that standard synthesis-based design flows can process. The instrumentation creates an infrastructure platform, which postsilicon tools dynamically configure and operate. Dynamic in-system configuration enables continual reuse of the instrumentation for a variety of applications. Configuration and control use the standard IEEE 1149.1 (JTAG) test access port (TAP), so no extra pins are required.

The infrastructure platform is a distributed fabric consisting of reconfigurable instruments. Figure 1 illustrates an instrumented SoC and provides a high-level view of the infrastructure architecture (indicated by darker shading). The programmable trigger engine (PTE) and the reconfigurable logic engine (RLE) are analysis instruments that process system signals connected to their inputs. They can be configured to implement various functions, such as detecting and counting events, identifying transactions, and checking assertions. The signal probe network (SPN) is a pipelined multiplexer network configured to select a subset of the tapped system signals and connect them to other instruments. The tracer is a capture instrument that records its inputs in a circular trace buffer. Trigger signals generated by a PTE or an RLE can initiate and terminate capture. The CapStim is a capture-and-stimulate instrument that can apply the values preloaded in its memory as stimuli to specified block inputs; stimuli loading and stimuli injection both occur during system operation. Additionally, the CapStim contains all of the tracer's functionality. The primary controller (PCON) is the interface between the TAP and the instruments.

Except for SPNs, each instrument belongs to a single system clock domain and operates at the speed of that domain. An SPN includes FIFO buffers for clock-domain crossing, and the design ensures that the frequency used to process FIFO buffer outputs is higher than the input frequency. Configuration and control occur at JTAG clock speed.

The two analysis instruments target different trade-offs between flexibility and efficiency. A PTE can be created with built-in counters, timers, and comparators, and includes one finite-state machine (FSM) template, whereas an RLE contains FPGA-style simple logic cells that can implement more-general logic functions.

## Applications

Engineers can configure the platform for many different validation and debug applications, including signal capture and logic analysis, on-chip functional-block test, assertions, performance monitoring, fault or error injection, and in-system scan dumps. Most applications can run concurrently.

In simulation, designers have long enjoyed many advanced capabilities, such as transaction-level modeling. A transaction is a sequence of related events—for example, a (Request → Acknowledge → Grant) sequence, a direct memory access (DMA) transfer, or a packet transmission. Transaction-level modeling is a proven beneficial technique in presilicon verification because it raises the abstraction level used in

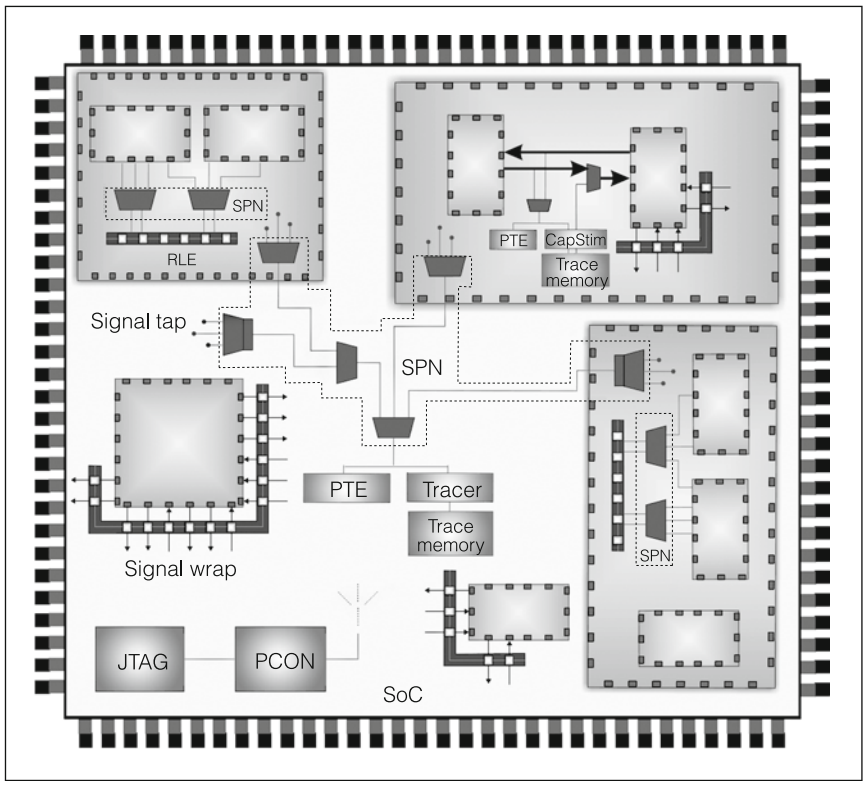


**Figure 1. An instrumented SoC. The darker-shaded areas give a high-level view of the infrastructure architecture. (CapStim: capture and stimulate; PCON: primary controller; PTE: programmable trigger engine; SPN: signal probe network.)**

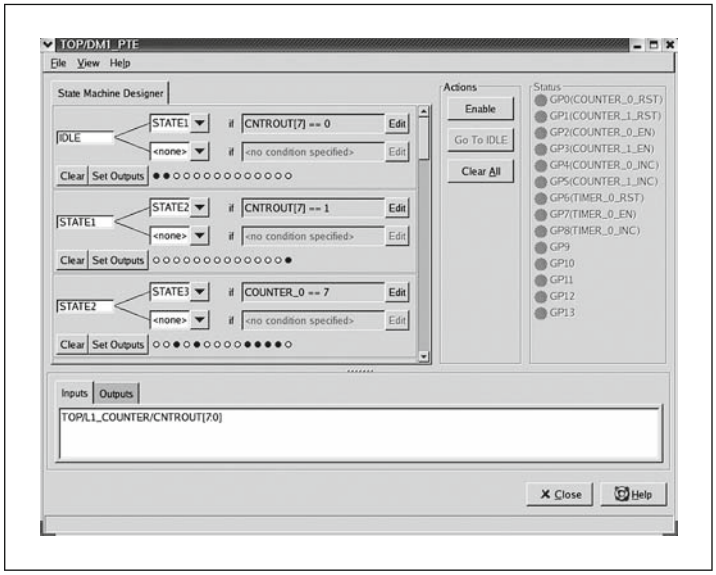


**Figure 2. Implementing a finite-state machine (FSM) with a PTE.**

analyzing SoC behavior. Transactions often follow a cause-effect chain, which can be easily traced forward or backward in time. It is often much more effective to follow a sequence of transactions than to analyze their corresponding bit-level waveforms. The reconfigurable platform extends transaction-level modeling from presilicon verification to silicon validation.

## Logic analysis

Engineers can configure the instruments to operate as an on-chip logic analyzer. On-chip logic analysis has been used before with dedicated instruments; what's new here is that it's implemented by reconfigurable instruments that can also perform many other functions. For example, assume we want to capture and analyze waveforms associated with a group of signals involved in a certain bus transaction. First, we need to configure an FSM in the PTE for this analysis, as Figure 2 shows, to recognize the completion of the transaction. The FSM model describes the states, the conditions causing state transitions, and the outputs activated in each state. We configure the SPN to connect the target signals to a tracer and the PTE, and we start a continuous trace. We program the PTE to stop the trace when the transaction is detected; thus, the trace buffer contains signal values captured immediately prior to the bus transaction. After the recording has stopped, we extract the time-stamped values from the buffer via the TAP and display them with a waveform viewer.

## Performance monitoring

Identifying and counting transactions in hardware is the foundation of performance monitoring. For example, it's possible to configure a PTE to count cache hits, cache misses, or packets received during a certain interval, or to measure latency in servicing requests for a shared hardware resource. Unlike software implementations of performance monitoring, which usually modify the data they attempt to measure, methods implemented with infrastructure hardware are nonintrusive. Of course, performance

monitoring is not a new concept; here, however, it's implemented with the same reconfigurable instruments used for other applications.

## Assertions in silicon

An assertion represents a relation between signals that must always be satisfied in a correctly operating system. For example, an assertion may state that data should never be written into a FIFO buffer that is full. An assertion fires when it detects a malfunction; therefore, we know where and when to start debugging, which significantly accelerates error location. Assertions enhance the effective observability of internal signals without requiring new pins.

Assertions have been used extensively in presilicon verification. Although a few assertions can be easily implemented as part of the functional circuit, it is not feasible to implement hundreds of assertions in silicon. Nevertheless, our platform enables, for the first time, the implementation of numerous assertions in silicon. Reconfigurable instruments permit assertions to be configured dynamically in an SoC running at speed. A group of assertions can run concurrently throughout the SoC indefinitely or for a specified interval. The ability to repeatedly configure and run different groups of assertions in a loop enables continuous reuse of the instrumentation and provides an automated, at-speed, in-system validation procedure. Just a few minutes of real-time system operation subjects each assertion to more stimuli than it would get in weeks of simulation or days of emulation, thus extending the coverage and usage of assertions for in-system validation. Moreover, validation engineers can create new assertions on the basis of a target problem's immediate needs and run them immediately. This is an important new feature, not previously available for SoCs.

Our system provides parameterized configurations for the well-known Open Verification Library assertions. Using the interface illustrated in Figure 2, we can implement an assertion in a PTE state machine, or we can define it in Verilog and compile it into an RLE configuration. The reconfigurable instruments can be reprogrammed at any time with new assertions, enabling a systematic application of assertions across the entire SoC. Focusing the validation effort on one functional block at a time (including operation under stress conditions) provides a confidence-building strategy that is beneficial when unexpected behavior is detected. This strategy avoids wasting effort analyzing already validated areas, and we can narrow the set of suspected blocks to the more likely culprits.

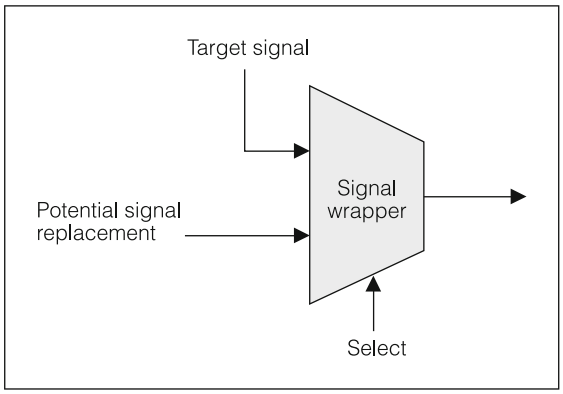


**Figure 3. Signal wrapper.**

Assertions can work in conjunction with the embedded logic analyzer to aid in debugging. For example, a firing assertion can serve as a trigger to stop recording in the trace buffer, so that the recorded signals provide a window into activity preceding the malfunction detected by the assertion. The most useful signals to record are inputs to the block where the assertion fired, because their values can help explain what caused the assertion to fire.

## Wrapping signals for controllability

The applications described so far focus on enhancing observability. Our platform also supports applications that enhance the controllability of specific signals. Basic controllability mechanisms rely on using a wrapper to replace a system signal. A signal wrapper, shown in Figure 3, is a multiplexer fed by the target signal and a possible replacement signal. A *static* wrapper can replace the target signal with a constant 0 or 1. For a *dynamic* wrapper, the replacement signal is generated by another instrument. A *reconfigurable* wrapper combines an RLE with a dynamic wrapper, so that any wrapped input can be replaced by a signal generated in the RLE.

Although a wrapper adds a multiplexer delay on a functional path, the circuit's performance is usually not affected, since the insertion is done in the presynthesis RTL model and synthesis takes this delay into account.

## Fault and error injection

Systems with high-reliability requirements are designed with fault tolerance or error resilience mechanisms. These mechanisms involve both hard-

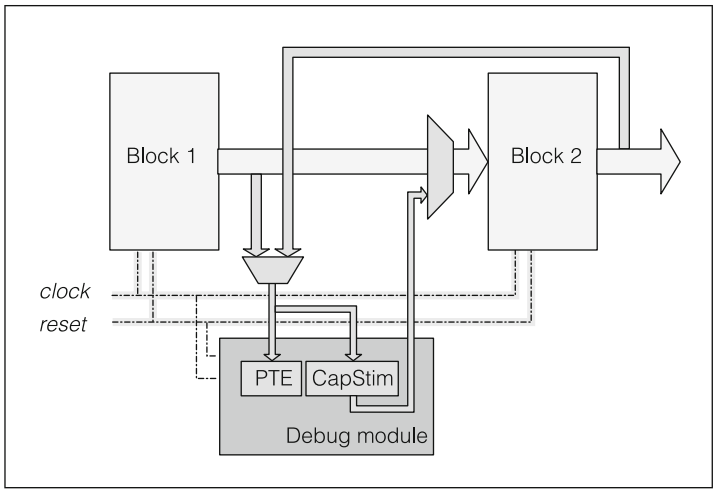


**Figure 4. On-chip test with CapStim.**

ware and embedded software and are computationally very expensive to verify presilicon, because they require full system-level simulation or emulation and usually deal with only one error at a time. Wrappers enable simple and efficient dynamically programmable fault or error injection. For example, we can emulate

- a stuck-at fault by replacing a signal with a constant,
- a soft error by replacing a signal with its complement for one cycle,
- a protocol violation by delaying a signal used in a handshake by several cycles, and
- a complex protocol error by replacing control fields used in a transaction with incorrect values.

Controlling the replacement multiplexer's Select signal lets us emulate permanent, transient, or intermittent problems. Because the system is working at speed, it's easy to validate the real-time system response against a large number of injected faults or errors.

### On-chip functional-block test

Often validation engineers would like to validate a certain aspect of the on-chip operation of a third-party core, a user block, or an interface by applying input stimuli and observing the response. Such an on-chip functional test is possible using a PTE and CapStim, as Figure 4 shows. Each input to the target Block 2 feeds a dynamic wrapper to enable the replacement of functional inputs with patterns from CapStim. The stimuli are preloaded in the CapStim memory. We configure the instrumentation so that when the PTE detects a trigger event, it selects the CapStim outputs to drive Block 2 inputs. While applying its stored vectors as at-speed stimuli to the block, the CapStim also captures its output signals, which are then extracted and analyzed off chip. Several blocks can share a PTE-CapStim pair.

### Integrating at-speed and scan-based validation

Observing register values can be useful in validation, but at-speed monitoring of many flip-flops is not practical. It is possible, however, to reuse the scan mode that all registers have for testability so that we can connect all registers in one scan chain that is easily accessed by scan operations from the JTAG port after functional clocks have been stopped. In-system scan has already been used in this manner.[4] The innovation here is the user's unprecedented flexibility in controlling scan dumps by specifying complex at-speed events to stop the clocks.

Scan dumps are most effective when combined with at-speed assertions and logic analysis. We can make the firing of an assertion stop the clocks when a malfunction is detected and use the register values obtained by scan dumps to complement information provided by signals recorded in the trace buffer.

## Overcoming difficulties

As mentioned before, in-system validation and debug must solve the problems resulting from lack of observability and controllability, nondeterminism and lack of time-specific expected values, and faults detected in system.

### Lack of observability and controllability

The embedded instrumentation provides direct observability of the signals connected to a trace buffer. For validation, we instrument key signals from every area of the chip, so that recording these signals can provide enough information about activity inside the chip. Key signals include those that identify important transactions; indicate operation modes; and control interfaces, communication protocols, dynamic voltage adjustments, frequency scaling, clock gating, and power gating. In-system scan dumps provide direct observability of all registers in the SoC for one clock cycle.

Analysis instruments that implement triggers and assertions provide indirect observability of the signals connected to their inputs. For example, a trigger can identify a transaction that could be recorded in the

**Table 1. Design characteristics of implemented devices.**

| Characteristic | Device A | Device B | Device C | Device D |
|---|---|---|---|---|
| Function | Serial ATA controller | ARM SoC platform | Processor complex | Digital TV controller |
| Technology | 90 nm | 90 nm | 65 nm | 90 nm |
| Size (gates) | 2 million | 2 million | 1 million | 4 million |
| Clock domains | 8 | 2 | 2 | 21 |
| Maximum frequency | 300 MHz | 266 MHz | 300 MHz | 400 MHz |
| Microprocessor used | NA | ARM9 | ARM9 | MIPS |

trace buffer or counted for performance measurements. Because an assertion fires when it detects a malfunction, assertions provide immediate observability of internal errors.

Wrappers provide controllability for the wrapped signals. Wrapping important control signals (such as resets, enables, and interrupts) lets the software easily create corner cases and error scenarios. Additional applications based on wrapping signals include on-chip functional-block test, error and fault injection, and soft fixes.

### Nondeterminism and lack of time-specific expected values

Nondeterminism is a problem only when we want to enforce a tester type of behavior-checking over the system operation. For example, assume that a simulation performs a scan dump after 145,668 clock cycles, and we record the result to use as expected values for the same scan dump done in system. But a correct SoC working in system with the same stimuli might legitimately produce a different result. Similar problems can occur if we compare the real values with expected values extracted from a ''golden'' chip or from emulation. Making sense of such comparisons is difficult.[5]

We avoid these problems by not using time-specific expected-value checks. Assertions satisfy this requirement because they check relations between signals that are supposed to be valid at any time. For example, a check that Acknowledge follows Request in at most three cycles does not depend on the time Request was issued. Similarly, if we record both Request and Acknowledge in a trace buffer, the relation between them can be checked by analyzing the recorded data regardless of the absolute times when the events occurred. Starting and stopping the recording can also be based on detecting specific events or transactions and not on their time of occurrence. Similarly, values of specific registers obtained from event-triggered scan dumps can be safely checked against values expected after the trigger event.

### Dealing with faults detected in system

Because assertions check correctness properties of the specified SoC behavior, assertion checkers can detect any problem that results in violating currently checked properties. Recent work has shown that assertions can provide good fault coverage,[6] so they can be successfully used for in-system fault detection. Using assertions for diagnosis is discussed elsewhere.[7,8]

## Silicon experience

Table 1 presents the characteristics of four devices implemented with our reconfigurable platform. Table 2 summarizes the details of the instrumentation

**Table 2. Instrumentation characteristics of implemented devices.**

| Characteristic | Device A | Device B | Device C | Device D |
|---|---|---|---|---|
| Signals tapped | 744 | 1,350 | 3,030 | 4,680 |
| PTEs | 14 | 1 | 1 | 1 |
| Trace buffer size | 2K × 36 | 4K × 64 | 2K × 132 | 2K × 44 |
| Signals wrapped | 1,230 | 218 | 512 | NA |
| CapStims | 4 | NA | 1 | NA |
| Pattern buffer size | 2K × 36 | NA | 256 × 32 | NA |

used. All devices worked at their specified clock frequencies (given in Table 1) in the presence of our instrumentation.

Logic analysis distributed over multiple clock domains was a common application targeted in all four devices. Monitoring transactions on the system bus was implemented in all the devices featuring an embedded processor core (B, C, and D). Simple assertions were implemented on devices B and D.

The main goal of device A was rapid in-system validation of the link, transport, and physical-layer blocks of a serial Advanced Technology Attachment (ATA) IP core developed in house by an IP design group within a large integrated device manufacturer (IDM). To achieve this goal, the implementers relied extensively on four SPN-PTE-CapStim instrument subsystems, targeting several IP blocks in different clock domains of the 90-nm test chip. The same instrumentation used by IP developers can be used later by IP integrators to validate the IP cores in their SoCs. Overall, the postsilicon applications significantly simplified and accelerated the work of the validation and debug team.

Device B is an ARM926 platform SoC. The IDM design team's objective was to provide product groups with a reference design comprising the ARM, basic peripherals, a functional configurable-logic block, and the reconfigurable infrastructure. Device B was implemented in a 90-nm test chip and had an instrumentation area investment of 4.8%. One advantage of instrumenting a platform SoC is that all platform SoC adopters can easily reuse most instrumentation to speed up their silicon validation process. The embedded logic analyzer successfully located a design error in the implementation of a functional fault injector and also a bit-swap error, inserted for testing purposes, in an image-processing block. Both errors were fixed using reconfigurable wrappers.

Device C is a 65-nm ARM926-based device. This project's primary objective was to investigate postsilicon validation solutions that could ease the difficulties experienced when moving the SoC ''ecosystem'' to new process nodes. Although the existing ARM embedded trace macrocell (ETM) and embedded trace buffer (ETB) system provided instruction-level visibility, our platform provided more granular and low-level insight as well as on-chip functional-test and diagnosis capabilities. The instrumentation was designed to inject faults and validate the fault tolerance mechanism's response to the introduction of soft errors in 65-nm cache memories. The validation team created error conditions by using static values or values applied dynamically when specified transactions were detected on the bus. One simple error scenario inverted select data on the bus during certain clock cycles. Another dynamically replaced data on the bus with values from the CapStim memory, using a pulse injection mechanism.

Device D is a 90-nm HDTV controller (a production device). Designers targeted extensive observability with an instrumentation area investment of approximately 4%. In addition, the SPN was connected to a low-voltage differential-signaling interface providing real-time off-chip streaming of on-chip data. The overall solution proved invaluable: two functional problems—a protocol error and an incorrect buffer address wrap-around—were found using logic analysis and simple assertions. The design team acknowledged that at least one of the two problems was a corner case that was virtually impossible to detect presilicon.

**Silicon results show** that the combination of embedded logic analysis, on-chip functional-block test, at-speed assertion checking, and in-system scan dumps for complete register observability creates a very powerful breakthrough solution for in-system, at-speed silicon validation and debug. The end results are a significant reduction of the silicon validation and debug time, and faster discovery and root-cause determination of integration problems, design bugs, and chip defects. This approach will likely be widely deployed in complex SoCs at 90 nm and below. The reconfigurable instrumentation's flexibility provides the key for developing additional innovative applications. Possible extensions include integrated hardware-software co-debug, chip security, and online testing. ■

## Acknowledgments

I would like to acknowledge the entire DAFCA team for contributing to the development of this technology.

## ■ References

1. B. Vermeulen and S.K. Goel, ''Design for Debug: Catching Design Errors in Digital Chips,'' *IEEE Design & Test*, vol. 19, no. 3, May/June 2002, pp. 35-43.
2. M. Abramovici et al., ''A Reconfigurable Design-for-Debug Infrastructure for SoCs,'' *Proc. 43rd Design Automation Conf.* (DAC 06), ACM Press, 2006, pp. 7-12.

3. DAFCA, http://www.dafca.com.
4. B. Vermeulen, T. Waayers, and S.K. Goel, "Core-Based Scan Architecture for Silicon Debug," *Proc. Int'l Test Conf.* (ITC 02), IEEE CS Press, 2002, pp. 638-647.
5. P. Dahlgren, P. Dickinson, and I. Parulkar, "Latch Divergency in Microprocessor Failure Analysis," *Proc. Int'l Test Conf.* (ITC 03), IEEE CS Press, 2003, pp. 755-763.
6. V.K. Reddy, A.S. Al-Zawawi, and E. Rotenberg, "Assertion-Based Microarchitecture Design for Improved Fault Tolerance," *Proc. 24th Int'l Conf. Computer Design* (ICCD 06), IEEE CS Press, 2006.
7. M. Boule, J.-S. Chenard, and Z. Zilic, "Assertion Checkers in Verification, Silicon Debug and In-Field Diagnosis," *Proc. 8th Int'l Symp. Quality Electronic Design* (ISQED 07), IEEE CS Press, 2007, pp. 613-620.
8. M. Abramovici, *Method to Locate Logic Errors and Defects in Digital Circuits*, US patent 7,296,201, to DAFCA, Patent and Trademark Office, 2007.

**Miron Abramovici** is cofounder and chief technical officer of DAFCA, an electronic design automation company focused on postsilicon SoC validation, debug, and in-system bring-up. His research interests include reconfigurable infrastructure for SoCs and its applications. He has a PhD in computer engineering from the University of Southern California. Abramovici is a Fellow of the IEEE.

■ Direct questions and comments about this article to Miron Abramovici, DAFCA, 10 Speen St., 2nd Floor, Framingham, MA 01701; miron@dafca.com.