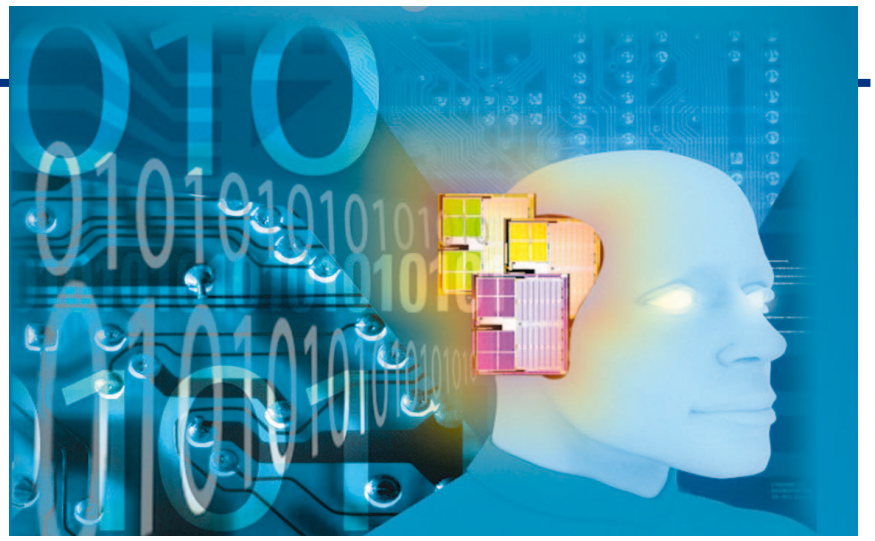
# Multicore debug and performance optimization IP for complex SoCs



By Christian Lipsky

System-on-chips (SoCs) are used in a wide range of application spaces from automotive to consumer electronics, and they are getting ever more complex. Multicore, multibus designs with a multitude of discrete signals and loads of embedded software are already commonplace. Today's high-end embedded systems already contain several million lines of software code and the number is growing. General Motors Corp. estimates that, by the year 2010, its cars will each contain 100 million lines of code[1].

Customers expect bug-free products; software failures can lead to loss of reputation, customer loyalty, and money or, even worse, loss of life in certain safety-critical applications. As a consequence, fast and effective debugging capabilities are a key success factor for embedded systems. To overcome the challenges arising from increased SoC complexity, new powerful debugging approaches are required. Advanced debug solutions also help with software optimization – for example, calibrating parameters in the field.

Before embedded systems were integrated in one single chip (SoC), the different functional blocks of the system were typically distributed as chips across a printed circuit board.

For debugging, logic analyzers and oscilloscopes were attached to the chip interfaces and buses on the board. This approach couldn't be employed any more when the different cores and buses moved into a single-chip SoC. Accessing the signals buried deep within the chip created new demands for appropriate debugging solutions. Traditional software and system debugging methodologies for an SoC include C-modeling, rapid prototyping, and run control.

C-models (or other high-level models) can be used to simulate a complete system, including hardware, by mapping the design to the memory of a simulation computer. This approach allows full visibility of the system internal states and signals. But this debugging methodology has major drawbacks. A model doesn't always behave like the real system will in a real-life environment because the environment impacts the system.

Additional effort would be required to model such environmental influences, and still not all effects can be anticipated upfront. Apart from that, the simulation runtime is much higher than running the real system hardware.

Even the continuous performance gains in computer platforms cannot keep up with the more rapidly growing complexity of the systems to be simulated. C-modeling can be useful at the early stages of a project when SoC hardware is not yet available.

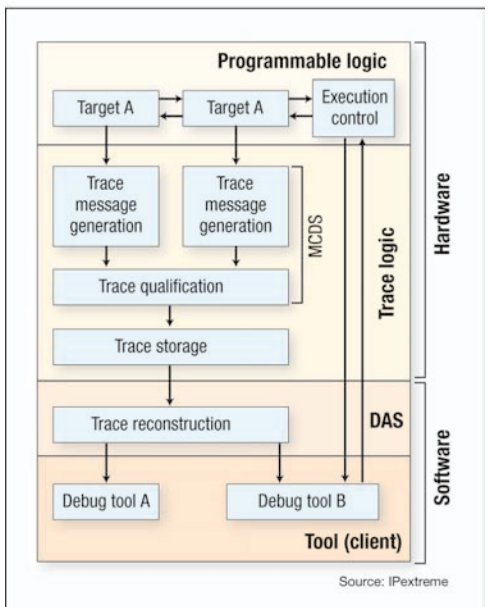



**Fig 1: Data flow through a MCDS-based debug system**

The complete emulator from trace message generation down to trace storage is located within the device package.

Rapid prototyping is an alternative method to C-modeling. In this approach, the debugged system (for which silicon is not yet available) is mapped onto programmable logic devices such as FPGAs. The fact that the RTL code of the debugged system is used in the emulating hardware allows for a more accurate debugging methodology than modeling.

Rapid prototyping features much higher execution speed than C-modeling, but it is still slower than the real SoC silicon. Typically, the FPGA emulation of a complex and fast system is at least four times slower than the final silicon version[2]. As with modeling, issues still arise around mismatches with the real system due to environmental influences. Also, dedicated environment devices like data sinks and sources or traffic generators are needed. Rapid prototyping also requires bulky and more expensive hardware, and the RTL description of the debugged system must be available.

Run control moves debugging closer to the production system by running system software directly on the SoC. In the run control approach, the system is halted when a certain user-defined breakpoint or watchpoint occurred. Most of today's processor cores contain additional debug logic that enables run control by an external debugger tool. For a multicore SoC, every core has to support run control. A multicore break and suspend switch is required in order to halt the system within an acceptable slippage[2].

The run control debugging methodology is restricted to postmortem debugging (i.e., the system is inspected after it has halted rather than while it is running).

For certain mechanical application spaces (for example, engine control modules and hard disk drives), this approach is not applicable because such systems can be damaged when they are suddenly halted before the system enters a state where it can be safely shut down. Run control will definitely remain a standard debug feature in the future. But its capabilities are not sufficient for real-time debugging requirements in complex multicore SoCs.

In the tracing approach, dedicated debug support hardware generates streams of information that has been obtained by observing certain system nodes of interest. The trace hardware can be located on-chip (on-chip tracing) or in external hardware (e.g., debugger box or emulator box).

Tracing is performed at runtime and enables the observed information to be time stamped. For instance, the instruction pointer or data/address values on the system bus are traced and then forwarded to an external debug tool. Appropriate means have to be applied to get the trace information off the chip.

The debug tool can reconstruct the obtained trace information and, for example, map the instruction pointer values to the known source code of the program that has been running on a certain system core, in order to analyze the program flow. In contrast to the run control approach, tracing is non-intrusive; that is, it has no impact on the program execution. Sporadic bugs, such as bugs that cannot be reproduced or bugs that only occur after a program has been running for a long time, can be discovered by the tracing approach much more easily than by single-stepping (run control). For debugging complex multicore SoCs in real-time, the most powerful approach is definitely tracing combined with run control.

In the bond-out based ICE, a special emulation device that contains the same logic as the target SoC bonds out internal buses and signals for increased visibility using additional pads and pins. Development effort is required to create an ICE probe that provides the adaptation of the emu-

**Christian Lipsky, is a senior design engineer at the IPextreme design centre in Munich.**

lation device to the target device's footprint.

Additional debug hardware required, such as an ICE box and a bulky probe, make this solution inapplicable for certain target systems and for debugging systems in the field. Furthermore, the clock speed of the observed buses or signals must not exceed the limit given by conventional chip pins designed for harsh environments. As the system bus is typically bonded out, this limitation directly refers to the system speed. A possible solution would be to run the emulation device at a lower frequency. But this might lead to behavior that is slightly different from the production SoC's behavior. Other factors may also cause possible differences – for example, the analog behavior of an A/D converter[2]. Using a bond-out based ICE is the most powerful and efficient debugging solution for single-core systems running at operating frequencies well below 100 MHz.

The IEEE-ISTO 5001-1999 Standard for a Global Embedded Processor Debug Interface, familiarly known as Nexus, features a message-driven on-chip tracing system. The idea is to have high-quality debug support on mass production SoC devices. This approach became practicable with the ever increasing scale of integration. The standard eases migration between processors from different vendors and enables support by a broad range of tool developers.

Nexus specifies a scalable hardware interface; JTAG is typically used as the control interface. An AUX interface with additional control signals (3 to 5 pins) is used to transfer trace data (usually 8 or 16 pins). The additional pins increase the device cost. Furthermore, the required SoC trace bandwidth scales with Moore's Law and rising clock frequencies. But trends show that costs for package pins don't decrease fast enough[2]. In addition, bulky connectors are required next to the chip.

As the Nexus architecture is intended for low gate count, the capabilities for trace qualification and trigger generation are usually limited. External breakpoints and watchpoints are heavily delayed by compression and buffering, which makes them impractical for finding complicated bugs. Another issue relevant for multi-target debugging is the fact that exact time correlation between different trace sources is nearly impossible[3]. Nexus is a suitable solution for SoCs with a certain complexity and frequency level, but not those requiring maximum trace bandwidth.

Nexus 5001 is too expensive for high volume, complex multicore SoCs and it doesn't offer a solution for applications where mechanical or environmental constraints preclude the use of bulky headers and short cables. Therefore, Infineon Technologies decided to develop a new concept in cooperation with leading automotive suppliers. In the MCDS based debug architecture, the complete emulator from trace message generation down to trace storage is located within the device package – see figure 1. So the debugging process takes place in the real target system and the "What you debug is what you ship" requirement is fulfilled.

At the heart of this emulator logic is Infineon's on-chip MCDS, which features trace, triggers, and performance monitoring. The architecture is independent of the physical interface between chip and debug host, and no additional hardware is required except for the host PC. Instead of an additional emulator box, a software layer running on the host computer can be used

*Continued on page 32*

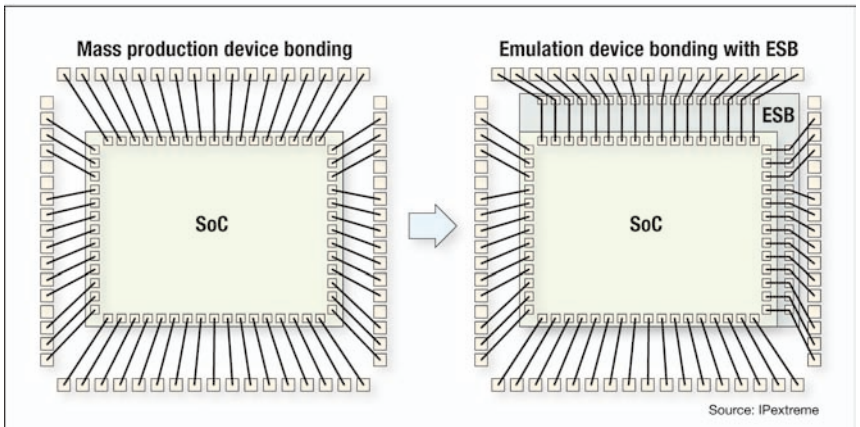



**Fig 2: A possible method using an emulation side booster**

The MCDS-based concept scales with integration density and clock frequency trends (Moore's Law).

# Multicore

from page 31

to configure the MCDS and to read the stored trace data. Infineon developed and uses such a layer, called device access server (DAS).

With this concept, any existing chip interface can be shared to get the trace data off-chip; so no additional pins are required. A popular solution is to use the standard JTAG pins as debug interface. Especially for debugging or calibration measurement in the field (e.g., engine control in a car while driving), a long thin JTAG cable is much more convenient than the bulky cables, headers, or adaptors which are required for other debug architectures. As the trace memory is located on-chip, the new approach does not have any bandwidth bottleneck issues.

The limiting factor is the size of the trace memory, which is constrained by production cost factors and die area requirements. Today's SoCs already generate several tens of Gbits/s of raw trace data. Even if the huge amount of information that results from program execution durations in the range of several seconds or minutes was stored (on-chip or off-chip), it would not be practical to find the bug by searching manually. Appropriate filter mechanisms are required in order to qualify special sections of the trace where the bug resides. The traditional approach (for example in Nexus 5001) is to record a lot of trace information and, afterwards, filter non-relevant sections of the trace.

The MCDS based approach takes the opposite path. The trace information is qualified before it is stored; i.e., only relevant information is written to the trace memory. In addition, this information, which consists of trace messages, is compressed. Memory sizes in the range of several tens of kilobytes are sufficient for simple program flow debugging. The trace memory can be configured as a circular buffer to collect trace messages either continuously or before and/or after a certain condition (e.g., watchpoint occurred) is met.

In some application spaces, security is a critical requirement. The software running in the SoC shall not be accessible to everyone. In order to support such requirements, many SoCs already contain dedicated logic that enables certain debug features only after an authentication process has successfully been executed. For such SoCs, the MCDS logic block can be locked by default and has to be unlocked by the described authentication process.

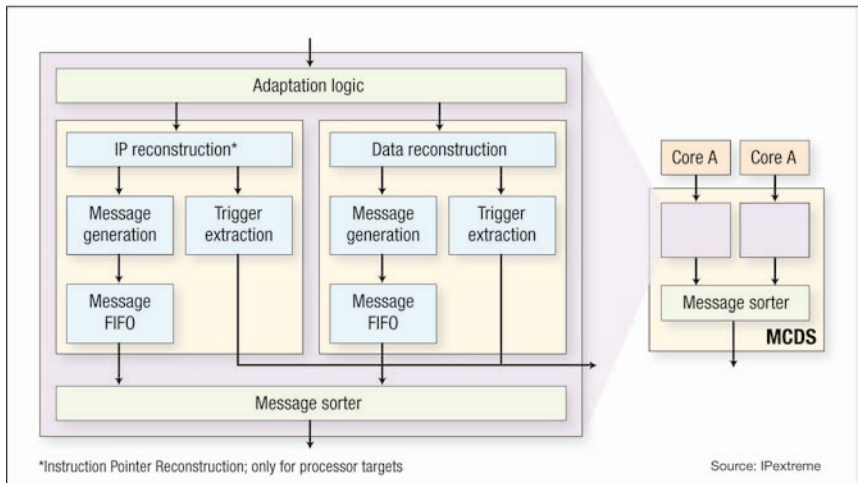



**Fig 3: MCDS functional blocks in two-target example**

**Depending on the target type (processor, bus, set of signals), the relevant traced information can be different for cores A and B.**

For devices where chip area or production costs are a critical factor (e.g., high volume products), a dedicated emulation device can be produced, which contains the logic of the production chip as hard macro, the emulation logic, and the emulation memory[3].

Because of its on-chip trace storage and because it uses the same integration technology as the production SoC, the MCDS-based concept scales with integration density and clock frequency trends (Moore's Law) and is therefore the right solution for complex SoCs in the future[2].

The MCDS-based concept is targeted to manufacturers of complex multicore SoCs who want to add highly sophisticated debug and trace capabilities, while maintaining reasonable production costs. Different markets from the automotive industry to consumer electronics will benefit from the extensive debug and code optimization features of the new concept.

The features in the MCDS-based concept as described previously are based on trace message generation, trace qualification, and trace compression. These tasks are realized by the MCDS logic block. Infineon partners with IPextreme to make the MCDS technology available to SoC designers as IP in order to create an attractive market for associated tooling, which benefits tool vendors (market size) and users (tool cost and quality).

MCDS can trace one or more cores in parallel at run-time and at full operating speed. Scalable time-stamping down to cycle level and exact time correlation between different trace sources (cross-target) allow accurate tracing of concurrency-related bugs. The IP solution supports any type of processor, bus, or set of signals as debug targets.

Bus tracing becomes more and more important in complex SoCs because a significant amount of transfers cannot be observed at the processor cores. Figure 3 shows the functional blocks of the MCDS in an example system for two debug targets (cores A and B). Depending on the target type (processor, bus, set of signals), the relevant traced information can be different. For instance, the instruction pointer can obviously only be traced at a processor target. For other target types, information such as data, status, process ID, can be of interest.

Highly configurable, scalable, complex trigger generation is used for trace qualification and run control of the processor debug targets. Triggers can be generated internally in the MCDS block – for example, by using state machines based on counters. As figure 4 shows, triggers from the debugged cores can be fed into the MCDS where they can be combined with internal or external triggers from other cores (cross-target). A configurable break and suspend switch allows control of multiple processors as a group by user-defined combinations of triggers.

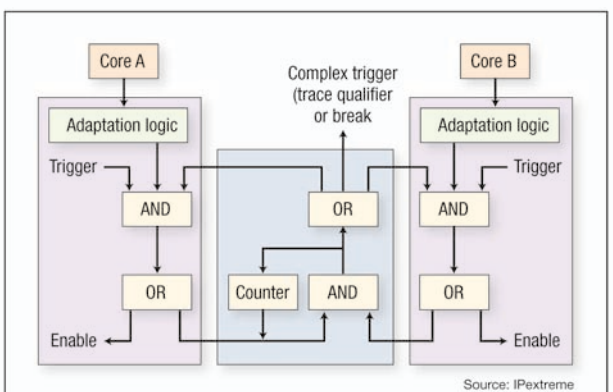



**Fig 4: MCDS trigger control cross-connect**

**A two-target example of how triggers from the debugged cores can be fed into the MCDS.**

Apart from the debugging features of MCDS, the solution also offers capabilities for code profiling and performance optimization. Dedicated performance counters evaluate specific signals from the debug targets. Corresponding trace messages allow analysis of attributes like cache hits/misses, number of executed instructions, number of stall cycles, or number of finished bus transactions per emulation clock cycle.

MCDS is already being used in the automotive industry, where calibration and corresponding measurement tasks are performed. For instance, software variables and program flow are traced over time while certain software parameters are being varied.

For debugging complex multicore SoCs, on-chip trace is the only long-term sustainable real-time debug approach. The new MCDS-based concept, comprising the MCDS logic block, trace memory, and access logic, offers the required features. The solution scales with integration density and clock frequency trends (Moore's Law) and is, therefore, the right solution for the future.

MCDS is available as an IP-based solution for multi-processor run control and trace of all types of processors, buses, and sets of signals. Using MCDS does not require an external emulator box and the trace data can be transferred through an existing interface (e.g., JTAG) to avoid adding high-speed pins. The solution has no bandwidth bottleneck and efficient trace memory usage is given by complex configurable trace qualification and trace compression.

**References**

**1. Robert N. Charette, "Why Software Fails", IEEE Spectrum, Sep. 2005, www.spectrum.ieee.org/sep05/1685**

**2. A. Mayer et al., „Debug Support for Complex System-on-Chips", 2003 Embedded Systems Conference San Francisco Paper, www.techonline.com/learning/techpaper/193103950**

**3. A. Mayer et al., "Boosting Debugging Support for Complex Systems on Chip", IEEE Computer, Oct. 2006**

**Online:**

**Multicore, compilation are key on embedded roadmap.**

**www.eetimes.eu/199203040**