



April 5, 2005

The Smart Approach to Designing with the ARM® Architecture

Intelligence

Technology

Design

Consumer

Market

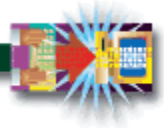
Tools

Resources

[Home](#) > [IQ Print](#) > [Version](#) > [Current Issue](#) > [Design Strategies and Methodologies](#)

- » [Home](#)
- » [IQ Online News](#)
- » [IQ Print Version](#)
- » [Members](#)
- » [Current Issue](#)
- » [Next Issue](#)
- » [Regional Issues](#)
- » [Past Issues](#)
- » [Subscriptions](#)
- » [About IQ Print Version](#)
- » [ViewPoint](#)
- » [Partners](#)
- » [About IQ Online](#)

## DESIGN STRATEGIES & METHODOLOGIES



### Multi-Core Embedded Debug - Volume 3, Number 5, 2005

#### Author:

Dr. Neal Stollon, First Silicon Solutions, Inc.

#### Synopsis:

A distributed debug architecture replaces processor-centric debug for more robust and faster system-level debug of multicore devices. Engineers who are involved in embedded system development know that getting the processor operations correct is often only the tip of the iceberg in getting a system up and running. In most ARM core-based systems, the processor core is just one part of a multi-core architecture; where it may be communicating and interacting with other processors, custom logic specific to the design and/or peripherals, and other IP blocks. Aside from the other complexities that multicore architectures introduce into the design and verification flow, they present new challenges in debugging the design. This article looks at the complexity of design issues with today's multi-core devices and how First Silicon Solutions offers a robust debug architecture for addressing them.

### Multi-Core Embedded Debug

Any system with deeply embedded cores and subsystems must factor in capabilities for debug purposes and can rarely assume that the necessary information needed for debug gets out to the device's pins.

System designers and processor core developers intimately understand this debug problem after only a few design cycles comprised of sleepless nights in the lab trying to infer, rather than observe, what is going on in their system. A range of debug approaches and tools are available for specific cores and chips that attempt to address the issue of getting "under the skin" of a complex architecture. Many rely on JTAG-based emulation and debug, using the ability of the JTAG-based run control and scan paths to put a system in a debug-friendly mode, gathering snapshots of the processor operation. This approach has a lot of value, but it can be very slow and tedious to gather the required information, and it does not address many of the issues that are specific to multi-core systems - such as interoperability and communications of the processor subsystems and synchronization of debug information from multiple processors.

Many modern debug approaches are based on embedded instrumentation - adding IP blocks into a design specifically to facilitate debug and provide a conduit for improved visibility into what is going on inside the embedded system. For many ARM designers, the Embedded Trace Module (ETM) is the most familiar onchip instrumentation block, which adds a processor trace capability to an ARM processor. ETM software and other debug blocks interface to the outside world by a combination of ways. Many rely on extending the JTAG port on a device for additional debug functions, or for cases where a higher debug throughput is required (and more pins can be tolerated), having the debug blocks interface with the outside world through a dedicated test access port.

The main focus by IP vendors for embedded debug instrumentation has been on trace and run-control of the processor core itself. However, a processor-centric debug approach does not support the systemlevel debug needs of designers who are also interested in debugging their whole system-level device. A system debug approach means that in addition to processor debug, designers need unified access to embedded bus operations, other third-party IP, and signals in their custom logic specific to their design.

First Silicon Solutions (FS2) focuses on debug solutions for embedded system developers, and in particular, developing debug solutions using embedded instrumentation to address system-level debug. In order to address a diverse range of debug requirements, system debug typically includes several types of customizable

debug and control blocks, well beyond simple trace and debug functions of a processor core.

There are two types of debug activities. The first, and simpler type, is gathering and extraction of on-chip information, broadly referred to as real-time trace. For trace operations, the primary concerns are appropriately triggering the start and stop times, compressing the information for most effective use of the debug resources, and streaming the information between the on-chip trace logic and off chip probes and analyzer tools. On-chip debug blocks that utilize this approach include processor, logic, and bus analyzers.

The second type of debug activity is onchip analysis, where the debug blocks run autonomously to gather system-wide information, performing more complex debug related activities. Examples include controlling and cross-triggering between multiple cores and tracking and analyzing core(s) to measure system considerations such as bus utilization or bus saturation. FS2 refers to this as a HyperDebug capability (Figure 1).

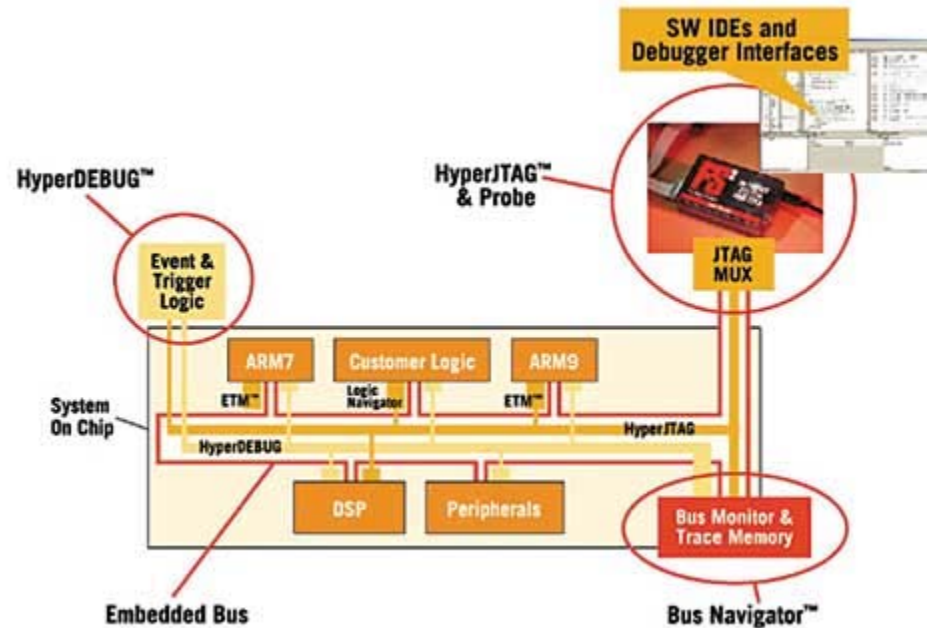


Figure 1

The integration of these instrumentation types and their interaction with processors and other IP, provides an integrated "Multi-Core Debug Backplane". This backplane works in conjunction with processor-specific debug blocks and JTAG interfaces to provide a central communications focus for system-level debug.

Some of the features of this debug backplane architecture are to enable a single point of control for:

- Accessing core-specific events and status from a variety of cores for a more system-wide view of operations
- User-defined, core specific or global actions such as run control over starting and stopping trace resources
- Synchronization/timestamps to all of the cores to enables coherency in debug of blocks that might otherwise be asynchronous.

For maximum flexibility, overall functionality of this "Multi-Core Debug Backplane" is typically defined as a combination of IP, its connections to important debug signals, and the off-chip tools that are customized for specific monitoring and debug operations. These tools include debug-programming interfaces to enable the user to dynamically define and control the combinatorial and sequential relationships of the monitored signals and how they interact for debug purposes.

The structure of debug architectures can vary depending on system size, debug performance requirements, and whether debug resources need to be dedicated or shared. For example, the amount of debug information gathered for even a small system can frequently outstrip the communications bandwidth available for bringing the information off chip. As a result, the debug information needs to be buffered on-chip. The memory resources for this buffering, especially for trace operations, can be significant. The tradeoff of whether to use a single trace block, or a larger number of distributed trace blocks can determine the overall trace memory size

needed, the speed of operations, and the efficiency of the debug solutions, especially for systems with different processors all having their own debug approaches.

For systems with only a few blocks, having global debug resources may be simple, but as the number of cores in the system increase, a large number of global debug signals or very long serial chains can impact timing closure or overall system performance during debug. While there are no hard and fast rules, FS2 found when developing its multi-core HyperDebug(tm) architecture, that in many cases, distributed debug approaches are less intrusive and can be best customized to interface with system debug requirements. (Figure 2) A distributed debug architecture enables smaller and faster trace blocks that can be customized to specific processor requirements and can be placed local to the processor cores for reducing full-speed operation debug issues. The distributed debug approach uses fewer global signals requiring timing closure, and provides the designer with the option to divide the debug architecture into subsystems. Debug subsystem can target specific debug requirements and simultaneously improve physical placement, which, for larger architectures, can improve JTAG performance. FS2 utilized this approach when it developed a JTAG superset architecture (HyperJTAG) to support heterogeneous processor debug over several debug chains.

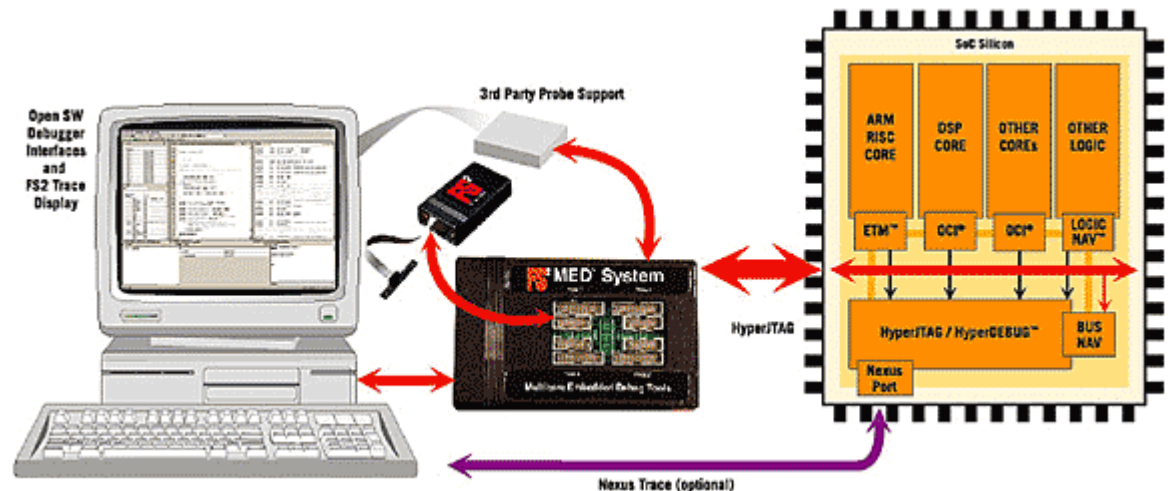


Figure 2

FS2's architectural approach to debug supports the integrations of different types of instrumentation blocks and analyzers. Heterogeneous multi-core designs (RISC and DSP, for example) typically have a range of native debug instrumentation, with ETM software as the most common approach used with ARM cores for debug control, triggering, and trace. Other types of cores and processors that make up the multi-processor architecture must be supported within the same debug structure. Since debug interfaces for cores may have different methods of control, signal definitions, and types of embedded resources, some type of transaction layer is often required to ensure a common set of signaling between the system debug controller and the individual debug blocks for different cores. This layer should support the debug block interfaces, as well as provide a translation layer for extracting and preprocessing status information from different cores and for sending and customizing control "actions" to the processor core for debug purposes.

In addition to trace and on-chip analysis, two other major types of debug instrumentation are often required for system debug. Most systems contain a combination of on-chip buses and user defined logic blocks with unique debug requirements. Bus and logic debug are often addressed using configurable logic analyzers that are specialized for event monitoring, triggering, and trace of the bus or block-specific operations.

As an example, since buses have large amounts of traceable data, it is often useful to automate selective dropping of bus cycles where no interesting transactions are taking place (idle or not ready cycles, as examples) to extend the effective trace depth. Another option for bus debug and trace viewing simplification is alignment of different phases of a transaction (i.e. transfer and response) to simplify viewing and analysis.

Logic analyzer instrumentation for design-specific logic blocks presents different debug challenges. There is often a wide range of custom logic in an embedded system, ranging from communication interfaces to hardware accelerators. This IP may be tightly or loosely integrated with other blocks (or processors) in a design. Configurable logic analyzer blocks allow a wide range of options in supporting digital event monitoring, triggering, and trace, and can be used in conjunction with processor- and bus-level debug blocks.

Here are some features to look for in an embedded logic analyzer solution:

- Support for a suitably wide range of signals for the specific debug needs. For example, to debug 100 signals, the logic analyzer must support the appropriate trace width.
- High enough operation speed to support the logic being debugged. If your logic is running 200 MHz, can the debug interface keep up?
- On-chip versatility and features to support the triggering, tracing, and debug operations required for design analysis. If the conditions that you are interested in debugging are sequential events, does the logic analyzer support sequential event triggering?

A final consideration is the choice of debug interfaces and methods of getting debug information off the chip and interfacing to probes and host-based tools. Most debug systems require some type of debug probe to handle the on-chip to offchip communications without using host system resources. The only standards that address the issues of generic probes compatible with a wide range of processors and with multi-processor debug is the Nexus 5001 standard, which has received limited industry support. Other approaches are based on JTAG or some superset thereof (First Silicon Solutions' HyperJTAG, for example), and on custom trace ports, which require corresponding custom trace probes and related tools support.

Multi-core debug is an important consideration in emerging architectures and will continue to grow in importance as multicore architectures become more complex and sophisticated.

| [Privacy Policy](#) | [Legal Statement](#) | [Site Map](#) |