

# Standard Debug Interface Socket Requirements For OCP-Compliant SoC

Neal Stollon  
*MIPS Technologies Inc.*

Bob Uvacek  
*Pixelworks Inc.*

Gilbert Laurenti  
*Texas Instruments Inc.*

## Abstract

Debug for SoC adds new requirements and challenges in terms of adding visibility and control to a system, simplifying integration of hardware and software instrumentation into design flows, and supporting emerging needs for architectures incorporating complex network on chip buses, multiple cores, multithreading, embedded security, power management and other issues. The OCP-IP Debug Working Group addresses the definition of debug resources and integration to enable comprehensive debug of OCP based systems. Contributors include OCP-IP companies (IP venders, systems integrators, end customers) addressing the need for debug solutions and/or debug of the OCP Infrastructure.

The initial focus is on definition of debug signals at the OCP Socket and fabric levels, but leaves their specific implementation open to IP and tools venders. The scope does not address in detail the external debug interfaces between an OCP **Debug Interface Socket** with on-chip Debug environment and external components (probes, debuggers, etc). These pin level interfaces are considered separate from the OCP Debug System and are being addressed by other industry working groups (Nexus, MIPI, IJTAG, Multicore Association Debug Working Group, etc). The OCP Debug Infrastructure defines requirements for interfaces between the Debug and EDA infrastructure and features, but does not address specific debug to EDA interfaces, which are again specific to different implementations.

**Interested parties wishing to provide feedback or contribute with specifications and models are invited to contact the authors or any OCP-IP group members. Contacts can be made via [admin@ocpip.org](mailto:admin@ocpip.org)**

## 1. Introduction

Debugging and Monitoring of SoC are important capabilities to supporting better quality and time to market hardware and software solutions. Debug is an emerging area in SoC design, with many facets to consider. The OCP-IP Debug Working Group is focused to standardizing a set of OCP debug interfaces to enable new and more rapid application of system debug/observation/cross-triggering IP and software solutions.

In recent years, the Open Core Protocol (OCP) paradigm has become an accepted solution for delivering high-performance on-chip interconnect fabrics for massive multi-core systems-on-chip. A special class of this paradigm is the multi-processor systems-on-chip (MP-SoCs). This document outlines a matching concept of system wide debug of heterogeneous MP-SoC using a standardized OCP bus interface for all IP blocks.

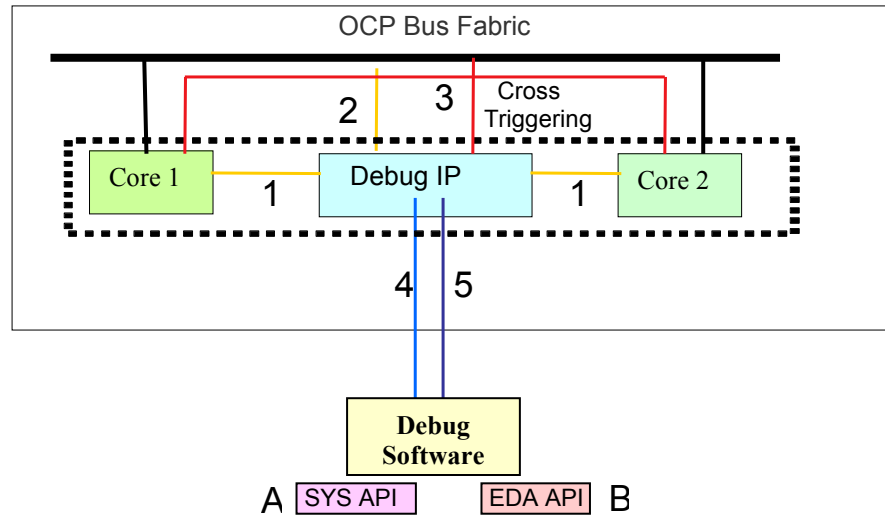
The essence of the proposed debug solution is an optional OCP port, known as the Debug Interface Socket, which may be added to all cores and IP blocks that support or need debugging access. The Debug Interface Socket is an OCP port with a restrictive configuration that is limited specifically to support debug interactions. The Debug Interface Socket may communicate off chip via a number of mechanisms; most common being a test port or memory mapped interface to processor. If debug blocks are memory mapped, then the normal OCP Data Socket is added to the debug socket to interface to on-chip IP blocks and cores. The Socket IP is ideally supported by a commercial infrastructure of tools, SystemC and other transaction based ESL and virtual debug block models, which complement the emerging trends in SoC design flows, and which interact with other OCP IP to provide comprehensive IP based debug solutions.

### 1.1 OCP Debug Overview

This paper describes the approach to a standardized OCP-bus compliant debug interface based on the **Debug Interface Socket**. The debug interface has several layers of extended functionality to address the disparate and increasing needs of three major debug communities: Software, hardware, and mixed SoC prototyping.

One goal is to enable debug companies to offer a library of proven debug IP blocks for many kinds of debug situations and purposes.

In this document an overall debugging framework is described which is the base of the OCP debug interface. In the same way that the OCP data was a superset for the various bus interfaces and data structures, the OCP Debugging Framework defines an OCP Debug Interface Socket that can connect to a superset of debug solutions, including those developed outside of the OCP-IP membership. As a quick overview, the following shows the key components of a simple (dual core) SoC Debug environment.



**Figure 1 – Debug Environment with 5 Typical SoC Hardware Connections**

**Debug Hardware Interfaces**

1. Core Interface – interface to core or IP block (run control and debug data /control)
2. Bus Interface – interfaces to a bus socket or fabric (to collect traffic data, bus event/trace, bus triggering)
3. Cross Trigger Interface – allow debug block related and controlled actions and event synchronization
4. Pin Interface Control – interface to package pins (JTAG for debug control, simple trace port for data export to external analyser. Debugger software). This differs based on types of interfaces and is outside of the OCP-IP Debug specification
5. Pin Interface Data – other higher speed data/trace interfaces (also defined outside of the OCP-IP Debug specification)

**Debug Software Interfaces**

- A. API interface to a System Debug Software tool
- B. API interface to EDA tools – Verification, Simulator, ESL

With the occurrence of multi-core architectures, new requirements for debugging have emerged. It is often necessary to observe the interacting activities of two (or more) cores in a “co-debugging” or comparative debugging situation. For this reason, we define the Debug Interface Socket to support at least dual channel event synchronous debug. Dual channel is used as a minimum to enable comparison, and event synchronous allows that instructions and on-chip events be displayed in correct temporal relation. Multiple channel approaches are supported by direct extension. Temporal relationships between channels may be established by several means, time stamping during collection of trace information being a simple and direct approach. To enable coherent, system-wide start and stop of debug tracing and other real time on-chip control, a hardware-light cross-trigger concept is proposed.

The OCP Debugging Framework concepts are extensible and can be reduced to single core debugging without cross-trigger and time-stamping and can be extended to more debug channels by duplicating hardware.

## 1.2 OCP Statement of Need for Debug Standardization

OCP compatible IP blocks typically need debug interfaces that are not comprehended in OCP Data Socket specification. Additional debug signal interface definitions make OCP an “Even More Complete Socket” by addressing the visibility and control needed to better analyze the operation and interaction of OCP architectures, at different stages in design flows while providing a common set of debug options and consistent signal interfaces. Debug signal interfaces are not currently being addressed by the OCP 2.2 specification. Enabling a robust on-chip debug capability is being recognized as an important Design for Debug (DFD) capability for complex SoC and having DFD standardization makes OCP more attractive as a SoC platform.

OCP Debug definitions also provide a common set of interfaces to improve incremental support from debug probe and tool vendors and EDA tool vendors and to allow debug and verification convergence. In most cases in this document, the interfaces and requirements are discussed, while the specifics of detailed design are treated as a black box to be implemented in different ways by IP companies and other OCP-IP members.

## 1.3 OCP-IP Interaction with Other Debug Activities

As mentioned earlier, debug related problems have several facets, not all of which are or will be addressed in detail by this working group. There are several other industry working groups that are addressing aspects of DFD and related on chip debug problems. The work addressed by the OCP Debug Group complements these other efforts. Some of the debug related industry activities include

- Nexus (IEEE 5001) Forum, focused on high performance trace related interfaces based on the IEEE 5001 (Nexus) standard
- MIPI Alliance Test and Debug working group addressing a range of debug interface efforts
- JTAG (P1687) , a working group looking at JTAG extensions
- Multicore Association (MCA) is looking at integrated set of Debug and Communication APIs for multicore architectures.
- AJTAG (P1149.7) is looking at 2 wire JTAG architectures
- SPIRIT Consortium has a Debug working group looking at debug register description within the SPIRIT IP-XACT frameworks.

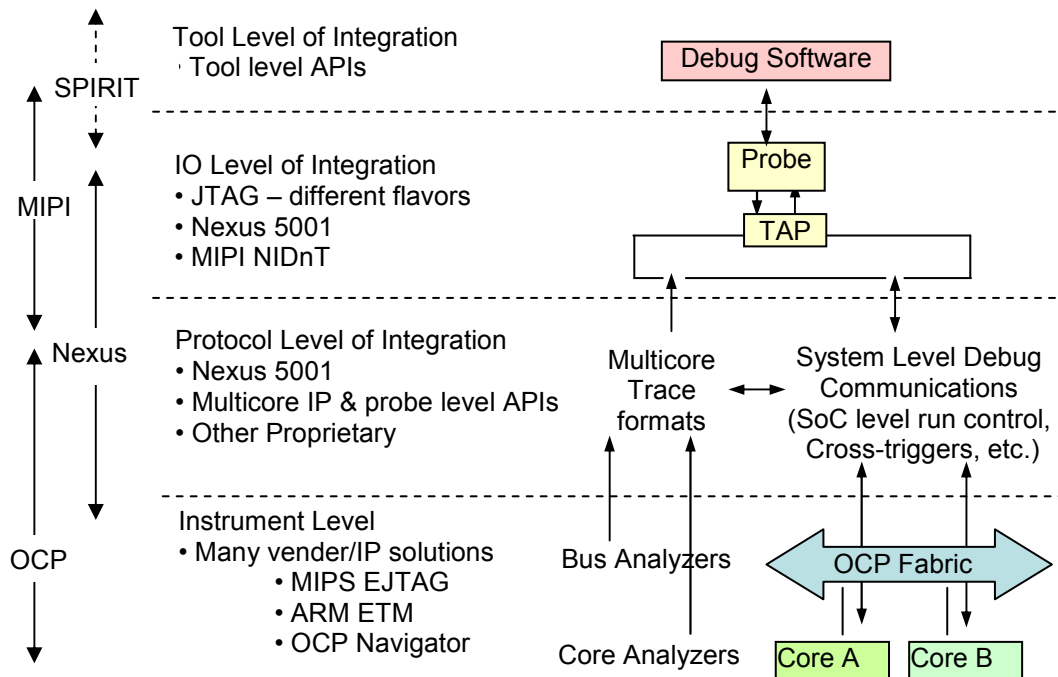
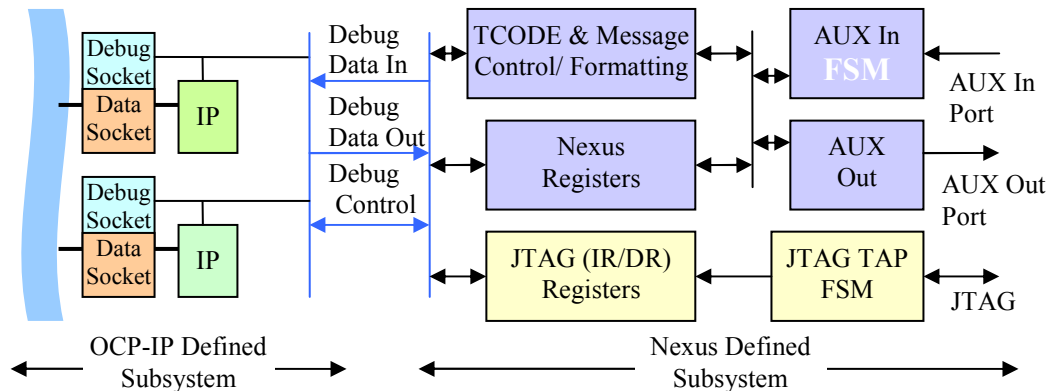


Figure 2 – Debug Infrastructure Activities

There are, in addition, other industry debug activities also investigating addressing debug issues. In most cases, there is an understanding and appreciation of the value of collaboration and cooperation between these industry activities in developing more comprehensive solutions.

One key point that needs to be emphasized is that the other discussed efforts are looking at the debug interface from the chip access or trace protocols at the port and pin level or at addressing related software and modeling requirements. None are looking at standardizing at the IP socket level interface and its directly adjacent instrumentation and wrappers, which is the primary focus in the OCP-IP working group. Figure 2 shows one interpretation of the relative areas of focus of some of the different working groups.

The OCP Debug working group focus differs from Nexus and MIPI activities by focusing on local debug of IP interfaces and operation standardization at the socket level, i.e. at the core and bus interface level, to address OCP compliant IP or subsystem debug control and visibility. These interfaces are interoperable with and should not overlap the IO level definitions that are the primary focus for Nexus, MIPI, and others.



**Figure 3 – Integration of OCP-IP Debug Sockets and Nexus**

#### 1.4 OCP Debug Working Group Strategy

The initial goal of the working group is to document a common set of Debug Guidelines and socket level signal models that address the range of simple to more complex debug of OCP based systems. Many approaches were evaluated based on contributed prior art. Others identify new capabilities that are needed for debug configurations and strategies that incorporate embedded analysis that comprehends multiple clock domains, power management domains, security domains, etc. required in modern SoC and embedded systems design. We avoid defining a separate debug bus to keep a simple modular IP structure on the chip, in line with the OCP-IP architectural philosophy.

In general, the goal in defining a debug interface socket is to be comprehensive in terms of providing a minimal set of baseline debug features for known debug issues and to keep the infrastructure scalable and configurable to address a range of alternative implementations. Where possible we leverage signals and interfaces defined in other OCP specifications. As an example, JTAG signals were defined in OCP2.0 as a primary debug interface. The intent however was not that implementation should be necessarily limited to 1149.1 JTAG. Due to (largely bandwidth) limitations of JTAG for SoC debug, other options are emerging, to enable alternatives with better debug support. As an example, for debug control interface, memory mapped debug control options that use an embedded processor for debug configuration and control are an alternative to JTAG. Trace port interfaces are higher bandwidth trace alternatives to JTAG serial interfaces (at their lowest level be compatible with Nexus and MIPI interfaces). With a separate interface layer, debug socket interfaces can be compatible at a TAP and protocol level with most debug IO interfaces – AJTAG, SerDes (Aurora), other single/dual wire debug interfaces, etc.

## 2 Debug Component Overview

In this section, an overall debugging framework is described as a base of the OCP debug interface. In the same way the OCP data socket is a superset for the different bus interfaces and data structures, an OCP debug socket can be a superset of the debug solutions based on standardized libraries of debug IP blocks that interact with the debug sockets signals, allowing

- Signal level observation (bus and system trace) and control (triggering)
- Consistent (multiple) processor software debugger and bus traffic observation interfaces
- Special debug features for security islands, voltage islands, gated clock islands etc.
- New classes of debug errors (which are different from system errors)

### 2.1 Three Views of Debugging

So, how will a debug environment be applied by OCP-IP users? We recognize that the purpose to debug in a chip can differ widely between companies, projects, and points in the design cycle and at least three variants want to be satisfied by a standard:

#### 2.1.1 Hardware Centric Debugging

Hardware debugging concentrates on additions in hardware to expose chip internal signals on the pins (JTAG or other) or in registers used to prove correct internal functionality or correct design parameters. This may include observation of signals from diverse parts of the SoC, including signals inside IP blocks. Triggering and trace capture should be precise to a clock cycle, to correlate debug hardware to simulation based verification. Software debuggers may not be required; however there are analysis advantages to including the display of such extra information for processor based architectures.

#### 2.1.2 Software Centric Debugging

Software debugging concentrates on minimum additions in hardware to provide a debug environment for software analysis (under the assumption that the hardware is implemented correctly). Many processors include a software centric debug capability (MIPS EJTAG as an example). In many cases, the debugger connects to the processor(s) to provide processor run control (and breakpoints) or execution trace. The processor may be used as a debug controller, with special instructions and triggering signals.

#### 2.1.3 System on Chip Debugging

System on Chip debugging concentrates on software tracing and hardware observation requirements common in SoC. Observation of the on chip hardware interaction is essential to complete the software application and verification. Comparative debugging of any two cores is equally important for multicore systems. The debug-system is independent of the target hardware and captures both “pre-reset” and “post-crash” events as well as bus traffic bottlenecks. Debugging must proceed even if the major parts of the system are in power down or a core is in sleep mode. The debug hardware may be disabled during normal chip operation, for security or power improvement, either under software control or by making parts of the debug hardware inoperable in production chips by burning fuses.

Debug features need to support the system level verification and analysis of OCP based systems. Where possible, RTL or other (System-C) blocks should be available for EDA analysis for JTAG and DFT, BIST, and other debug structures, even when these are implemented as physical (post synthesis gate level insertion) macros. From a system point of view, debug blocks should ideally support the same model abstractions used in other areas of a design, in order to support ESL simulators and high level software debuggers. This simplifies interaction of the hardware and software personnel being involved in the debug process.

## 2.2 Technical Scope of OCP Debug Interface

In the sections 2.4, we define signal groups that are considered basic to an OCP debug interface (debug control, debugger interface, cross-triggering interface, etc.), presenting them in the context of an example of a dual channel synchronous debug interface using the widely implemented JTAG port. These are typical interfaces required in many multicore analysis scenarios. In section 2.5, we also discuss additional (extended)

groups that are optional based on specific debug and analysis requirements. The optional Extended Debug Signals in this interface are defined for debug features such as time stamps and performance analysis and to simplify definition of special “debugging aware” functionality in designs that have security domains or power management with voltage islands.

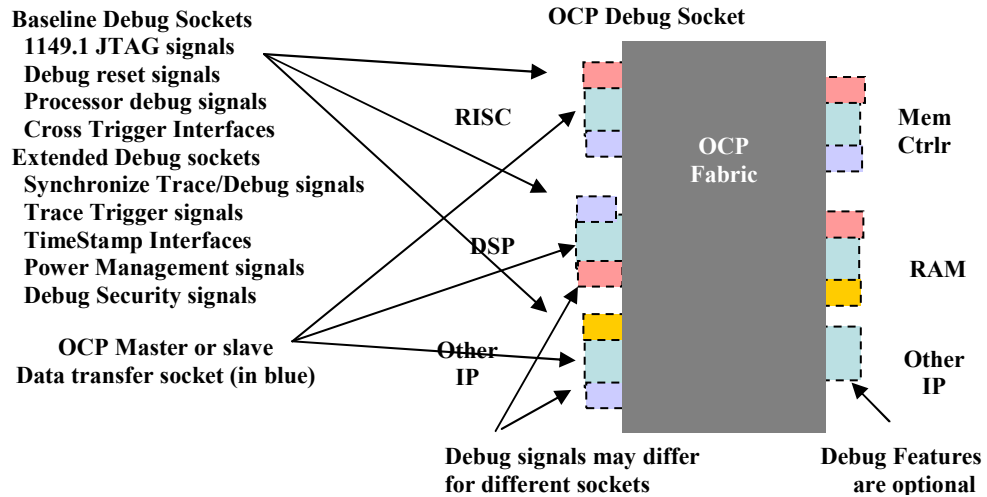


Figure 4 – OCP Multi-core Interfaces Including Debug and Data Sockets

### 2.3 Debug Components and IP Interfaces

The basic signals and definitions for an OCP Debug Interface Socket may be added to all cores and IP blocks that support or need debugging access. The OCP Debug socket may be implemented as additional signal to the OCP Data (master and/or slave) port (simplifying where debug blocks are memory mapped and controlled through the OCP Data socket) or as an independent OCP port configuration.

Figure 5 shows a simple system where debug IP blocks are implemented and integrated into a modular OCP system interconnect structure that is created from library IP blocks around a bus fabric (as described in the SPIRIT XML conceptual framework). Debug wiring goes through the system bus fabric and is contacted through an OCP debug port. OCP debug ports sockets may be implemented independent of data sockets, including at different points in the OCP system from where a Data port may be implemented. In general, the debug socket functions are passive and not intrusive to system operations or performance, however some debug block operations (cross triggering, etc.) may have interaction with other parts of the system operation.

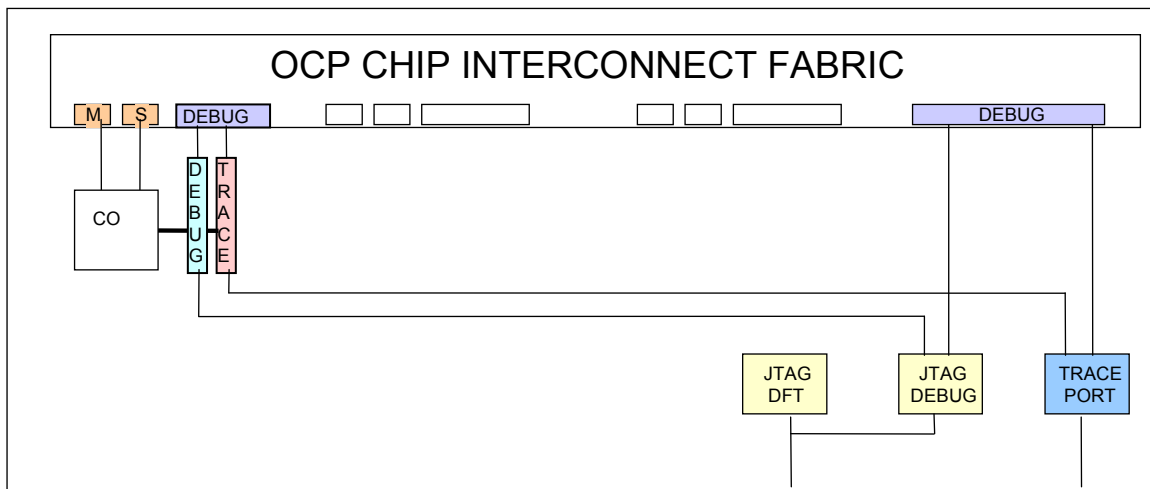


Figure 5 – Single-core Debug Solution with Wiring through OCP-Debug Sockets

## 2.4 Debug Interface Definitions

The programming of registers that contain either configuration or status information in the debug IP blocks may be JTAG-mapped or Memory-mapped. Either one or both modes of control and access are acceptable, based on specific system requirements as shown in Figure 6. For simplicity, debug ports discussed reference JTAG and Nexus interfaces. This restriction is to simplify the interface discussion, and is not intended to preclude integration with MIPI, IJTAG, or other interfaces.

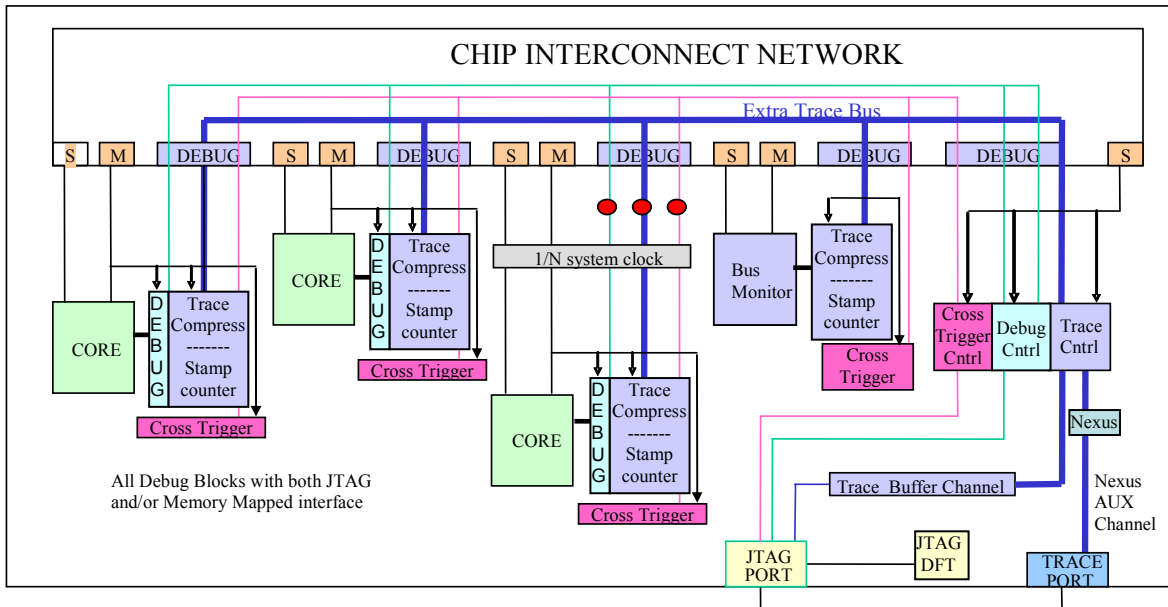


Figure 6 – A Multi-Core Debug Socket Implementation

### 2.4.1 Basic Socket Level Debug Interfaces

Processor run control is typically implemented via the JTAG interface and by debug mode signals in an IP. The use of JTAG debug interfaces is supported via OCP specifications and it is assumed that any JTAG signals are decoded at one or more JTAG TAPs (Test Access Points). A JTAG-only debug interface is adequate for many non-real time debug operations, including the ability to (serially) interface debug components on different cores and to set up and synchronize an OCP system into a debug mode. Even in the case of “memory mapped” debug blocks the processor control can operate over this JTAG port. As a lower speed serial interface, JTAG can limit performance on data intensive debug operations such as trace, which is driving other higher performance test and debug interface efforts.

Debug control signals are independent of the target system and may duplicate many basic controls. The basic debug signals include an independent reset and independent clock signals for the debug system synchronized to the debugger interface. An independent clock allows more flexible support of asynchronous or clock gated systems. An independent reset allows analysis of the target during system reset sequences. The reset and clock signals for time stamping counters may be common or independent from debug control interface. Debug reset, debug and timestamp clock and debug and time stamping reset, may also be common with system clock or reset.

### 2.4.2 Core Debug Socket Interfaces

OCP debug programming models should allow a user defined debug configuration based on the debug scenario. SoCs with multiple cores and buses have different requirements from simpler systems. A standardized set of debug interfaces that address system level debug of run control and debugger tools interfaces are perhaps the most significant innovation being discussed, since they allow bi-directional debugger accesses to be consistently controlled via these debug interface signals. Special signals that are included in this group are a debugger initiated debug mode request (rd/wr) signal and a core acknowledge

signal to the debugger that the core has entered a debug state. These signals may be aliased to existing signals in processor IP providing similar function (as example; for MIPS cores these are DINT, and DM interfaces). The core interfaces should also support unlocking of stuck-at situations and forcing completion of locked actions (Force, abort, suspend). Not all signals may be required for all cores or systems.

OCP peripherals interfaces would also need to be “debug aware”, to recognize and synchronize with processor cores or other bus masters that enter debug mode. This would require:

1. Processing an OCP read transaction initiated by debugger slightly differently than an OCP read transaction initiated by the application software.
2. Optionally stopping a local hardware process as soon as the host processor enters the debug state.

A peripheral debug interface implementation shall insure that, for Debugger OCP transactions, a debugger initiated debug mode operation reads peripheral information transparently, while preserving the system state. The debug operation disables any application based peripheral read side effect that may be present (ex. flag clear, post-increment, state machine update] as part of the access.

Depending on debug scenarios and the relationship between the local hardware process and the software process, it may be desirable to stop peripheral or other local hardware process when a processor enters the debug state. The peripheral can monitor the debug state to freeze local hardware processes when the host processor is in debug state. The OCP peripheral may be potentially shared or accessed by several OCP masters in the architecture. This may be by a parameter passed into a system debug register via JTAG or under software control or it may be implemented as part of the debug hardware.

To accommodate the diverse debug scenarios, a peripheral debug programming model may implement the two or more debug control parameters in a system debug register as -

1. FREE allows either keeping the local hardware process free running or make it sensitive to the debugger input.
2. SOFT allows waiting for a clean boundary before stopping the local hardware process when extra latency is acceptable.

### 2.4.3 Cross Triggering Socket Interfaces

Cross triggering and their associated system level control are another important feature for debug of complex SoC. Cross triggering allows global and distributed event recognition and multi-core triggering needed to identify and isolate events occurring throughout the system. Event recognition and triggering is widely used in conjunction with trace to capture information on events and data in the SoC. Trigger in conditions are monitored and compared to generate real time triggers in a Cross-trigger Manager. Examples of trigger signals might be debug request or interrupt request. These triggers in turn can be used to control event actions such as configuration changes, setting breakpoints or interrupts, trace collection or user defined requests. More complex trigger implementations might be programmed to trigger on specific values or sequences such as sequential combinations of bus address region and data read or write cycle type accesses.

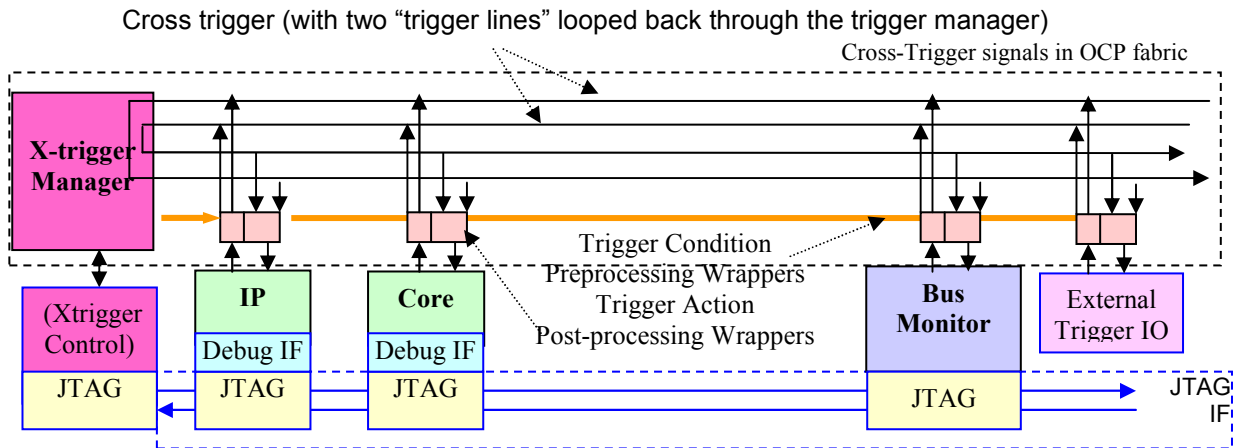


Figure 7 –Cross-triggering in the OCP-bus

The cross-trigger operation may be distributed among different IP connections to improve performance and support clock conversion and synchronization. Level (as opposed to edge) triggering is recommended for widely varying clock domains. Trigger-out (action) & Trigger-in (condition) pre/post-processing wrappers at each OCP interface point can be used to simplify the cross-trigger. Condition and action pre/post-processing may be of varying complexity and may be configurable via JTAG or processor control. Trigger signal routing may be addressed using sideband signals in the OCP interconnect. Trigger events may be routed to Trace components. An external device shall be able to contribute to cross-triggering and the trigger should support external triggers.

#### **2.4.4 OCP Synchronized Run Control**

The Sync Run Control supports clock synchronized program execution of two cores that would run asynchronously in normal case. That makes it possible to time align the instruction streams to study interdependency. This is a simplified method to avoid the hardware for time stamping.

#### **2.4.5 OCP Traffic Monitoring and Trace Interfaces**

Traffic monitoring and trace are in many cases, perhaps the most critical debug feature in terms of being able to analyze on chip behavior. In typical cases, system monitoring and trace is being done on signals on the data socket and the minimum trace control that needs to be defined is a trigger for trace enable. This enable is application dependent, it may be implemented in a trace block from monitored bus traffic events occurring within the bus operations, it may be extracted from the system cross-trigger information, it may be provided by a processor or other on-chip IP. Specific features for Bus Monitoring and trace are product dependent; however the following are features that have been implemented in existing products:

- User defined and predefined OCP transaction filtering and alignment of requests & responses
- Filtering based on OCP operations (ex. Initiator, thread, address range, DMA logical channel)
- Trace capture of both OCP transactions and non-OCP qualifying events
- Continuous (or at least long duration) System monitoring
- Non-invasive to preserve the OCP System bus behavior
- Secure at both the JTAG and other interface levels
- Buffering to support elastic trace bandwidth at OCP System traffic peaks
- Support for SW instrumentation interleaving
- Support system trace data reads from the JTAG or from application SW
- Support interleaving several trace flows from different trace points or channels
- Support Multi-threaded data observation
- Optionally support trace-packets for different (MIPI/Nexus) protocols

### **2.5 Extended (Optional) Debug Interfaces**

Extended debug signals support application specific or optional functionality designed into the target system (Power Islands, Secure subsystems, others) or the debug subsystem (Performance Monitoring, Timestamps, others). In all cases, they follow the point-to-point communication protocol and expose the OCP signaling to the debugger via JTAG-controlled or memory-mapped registers.

#### **2.5.1 Performance Monitoring**

Performance monitoring enables observation of selected parameters (ex. bus channel bandwidth, system event latency, monitor transactions to a specific Initiator or target, thread or tag characteristics, etc.) to identify data traffic and measure data bandwidth based on user defined Start & Stop triggers. Performance monitoring may be a standalone function or an option in a traffic-monitoring-and-trace block.

#### **2.5.2 System Time-stamping**

For distributed systems, a timestamp provides a means of temporally correlating events that may be occurring in different systems or asynchronous domains. We define the timestamp implementation – a gated clock and reset that can be used to run timestamp counters at different blocks. The timestamp clock and reset may be the same or different from system clock/reset, debugger or trace clock/reset, etc.

### **2.5.3 Power Management Monitoring**

Power management is integral requirement for many SoC products. Variable or intermittent power can impact systems operations such as debug IP by locking the IP into a state or interrupting the debug process in a debug session. Ideally in case of gated clock domains and voltage domains, debug continuity and functionality should not be interrupted if any IP block switches off clocks or voltages, goes into sleep mode, etc.

Power management debug signals help to avoid confusion and set guards against unexpected power related operations on the chip. As one of the few changes in the OCP\_IP Data Bus Specification, we recommend an extension to the SRESP field in order to add 2 new default responses (NOPWR, NOCLK) that indicate the condition on the IP. These can be used along with optional power management triggering to generate system triggers when a power situation changes in a domain (switching off, waking up, switching frequency, switching operating voltage, etc.) to adjust operations to support continuous Power Management monitoring and optionally preserve the OCP System bus behavior without software intervention.

### **2.5.4 Security Debug Interface**

One emerging issue of SoC security is that debug features can be used as a backdoor into many system functions. The most common way of addressing SoC debug security today is via fuses or bond out wires that are connected during chip validation and permanently disabled in the production chips. The disabling of security is typically irreversible and can adversely impact the integration of the SoC and in the later system lifecycle stages. Security concepts require selective enabling debug of sensitive locations at different times and with different levels of authorization. This is best implemented as part of chip level or system level security environment. OCP security enabling can be extended to the debug socket so debug IP blocks can implement selective access and debug feature security features.

## **3 Addressing the Debug Requirements to SoC Design**

Debug and analysis related tasks for SoC systems are useful in different phases of a design effort, and many diverse engineering activities would profit from improved access to debug IP, observation IP, verification IP, assertion IP etc. In this section we address some of the more global issues associated with debug and how the OCP-IP debug socket, along with supporting models and corresponding tools, can make a positive difference in the design and analysis efforts. We look in particular at 7 activities where improved on chip debug can make a difference.

### **3.1 System Verification of SoC Designs Before Tapeout**

Verification of designs prior to SoC tape out and providing better linkages between post-silicon and in-silicon analysis can have a dramatic impact on SoC design flows and first pass success of SoC designs. Improved verification is in many cases the critical path before tapeout. By standardizing an OCP debug interface we enable a more uniform and less of a development environment for hardware debug solutions to be used in prototyping a design for validation. By having consistent interfaces and models of the debug components, the correlation between simulation, emulation, and other hardware prototyping can be improved for IP and system hardware and software co-debug and co-verification. In particular debug capabilities in a prototyping or emulation system may be the quickest and simplest way to verify software performance and interactions for complex multiple embedded processor architectures.

### **3.2 Software Debugging of SoC Processors**

Standardized OCP debug interface will enable software debug solutions for common OS or RTOS environments. Embedded processors often running RTOS typically have trouble simulating for hardware/software (HW/SW) and require emulation on other hardware platforms. Higher abstracted ESL design methodologies work with processor debug using cycle-accurate software and hardware modeling techniques to complement on chip instruments to improve visibility of internal events and data streams. Including debug IP models in the ESL flow allows developing tests and routines that can be reused during emulation and silicon validation. Modeling also allows the debug system itself to be debugged and tested at the ESL level.

### **3.3 Functional Validation and Real-time on-Chip Monitoring of SoC**

Standardized OCP-debug interfaces take a large measure of the guess work out of the data collection. In conjunction with standardized interface definitions (as used in SPIRIT or in other XML-text files) it allow simpler data description, and display for software integration, system optimization, bottleneck analysis, high level models predictability, correlation of hardware and software events, etc. Standard debug interface descriptions in on-chip functional monitoring and functional validation allow "XML-Based" Assertion Generation for triggering and "virtual instruments". The latter are used for reconstructing bus events, history, commands, data bursts, etc. which are typically information needed in real time multi-processor and multi-cache systems (i.e. processor + cache+ other buffers) debug, and performance analysis and management.

### **3.4 Manufacturing Testing in the Functional Mode**

Manufacturing test issues are increasing with SoC size and with new DFT (Design for Test) techniques used for high volume production of chips. Design for Debug (DFD) and DFT Development overlap on many of the same concerns and requirements, such as functional configuration and visibility into the SoC which are only addressable by standardized interface information. Functional testing using a debug block infrastructure is defined in DFT approaches such as IEEE1500 and is beginning to supplement test coverage in manufacturing test. Having standardized internal debug interfaces and observation IP (along with EDA tools to insert and tester software to exploit on-chip information) simplifies many ongoing issues in DFT adoption and use.

### **3.5 Supplementing Documentation of On-chip Data Flows and Events**

Many decisions during and after the SoC design process are based on documentation. The accuracy of the information is as important as the quality of the RTL code. Often the quality and accuracy of documentation is an issue in many SoC flows. Debug systems supporting consistent, logical and automated flow for debugging of designs provide designers with views and insight into intended design behavior, and these insights are improved by standard interfaces that reduce the learning time required to understand how debug is being implemented for a given design. When supported by an underlying set of debug-specific databases to store and organize design information, the debug capability becomes part of a documentation scheme that allows better understanding of both the hardware software operations. In more complex SoC systems, documentation often falls short of capturing all possible combinations of system events.

### **3.6 Automating SoC Internal Descriptions**

While multi-processor systems differ, they typically include huge amount of registers. Standardizing on interfaces to these registers improves the ability to quickly and possibly automatically create the relevant debugger views and structural cross-trigger views for access to registers and events needed to enable and automate capabilities such as assertion-based design, network-on-chip mapping, ESL design methodologies, test modeling, etc.. Developing the software and data support for this type of debugging interface is being addressed by XML standards such as SPIRIT. However groups such as SPIRIT do not address the standards of the on chip infrastructure. Applying debug interface standards to the on chip registers, such as in an OCP-system, help extend the support of automation of these descriptions.

### **3.7 Co-debug of Multiple Processors on System Boards**

Many of the same debug requirements for debugging of multi-core SoCs also apply for boards with multiple processors. The debug problems involved are often an extension of those encountered when a SoC is being prototyped or emulated. Debug tools such as trace and cross-triggering are often needed at the board level and being able to apply standards based approaches facilitates the design process. Standardization of debug interfaces allows debuggers to be aware of the structure of the system board and offer more resources and capabilities and simpler debugging activities.

## **4 Future Directions**

The technologies involved in developing advanced debug techniques and consistent Design for Debug methodologies needed for more complex SoC are still emerging and evolving. There are an increasing number of organizations looking at the problem from different facets, all of which provide a needed piece of the puzzle. In the OCP-IP Debug Working group, we are addressing the facet that ties debug of OCP compliant IP blocks in a standardized manner to other activities. This does not result in a solution for SoC

debug by itself, but facilitates development of comprehensive debug solutions in conjunction with IP and tool vendors. We expect that progress in developing these debug solutions will require increased interaction with other debug related activities.

A key advantage that the OCP architecture provides is a standardized and robust set of interfaces that reduce the risk of the IP vendor in developing interoperable IP and the system developer in applying that IP into systems. However, current OCP-IP standards do not address this emerging issue of enabling complex on-chip debug. We have presented several on chip debug scenarios based on the use of an OCP Debug Interface Socket that interfaces with existing chip level debug interfaces, discussed the specific component signal types of the Debug Interface Socket and discussed how the integrated approach can be applied in design and automation related scenarios that are relevant to current and future SoC designs.

### **Acknowledgements**

The authors would like to thank OCP-IP and their respective employers for their continued support and contribution. The authors also thank other workgroup members and industry partners for the fruitful discussion and their useful comments during this initiative.