IP core-centric communications protocol Introducing Open Core Protocol 2.0

by OCP-IP Specification Working Group
www.ocpip.org

Abstract
Talking about IP core plug-and-play reuse, it is now generally admitted that cores need to be decoupled both from the on-chip interconnect and from one another using a clearly-specified core interface protocol. This interface must allow an IP core developer to focus on IP core generation without detailed knowledge of the device in which the IP might be instantiated. In other words, the core must be portable from one SOC design to the next without integration rework. Today this is all the more desirable as highly integrated designs are often developed in a complex multiple-locations multiple-teams environment.
The Open Core Protocol, initially introduced by Sonics and widely known as OCP, has been pioneering that design methodology for several years. OCP is effectively core-centric and thus applicable to all on-chip interconnection systems. As a natural result of this success, the OCP specification has been moved under governance of the OCP-IP consortium. This association has been created in 2001 and is driven by a pool of semiconductor industry leaders, including Nokia, Texas Instruments, STMicroelectronics, UMC and Sonics, the original founder. A lot of other major players have joined the consortium since its creation; hence today more than 70 member companies are merging their effort for the widest OCP adoption.
This paper provides first an overview of the SOC challenges; and demonstrates how the latest release of the Open Core Protocol fulfills the requirements.
The complete specification of the OCP-IP2.0 release [5] can be freely downloaded from www.ocpip.org.

1 Motivation background

Taking advantage of the increasing density of IC process technologies remains extremely dependant on a formidable challenge: How to complete the design and verification of these increasingly complex chips, while shrinking the project schedules as expected by the market? A key element of the solution is a massive reuse of pieces of existing pre-verified designs through intellectual property (IP) core reuse.
Adapting cores from chip design to chip design to make them fit with the rest of the system-on-a-chip (SOC) has become for a while a totally inefficient and unproductive methodology. Each time a core is to be integrated into a new system the system integrator is hampered by massive rework that reduces productivity and dangerously stretches the project schedules. Considering the variety of cores and the proliferation of on-chip interconnect systems, this painful work quickly grows exponentially, making the adoption of a standard mandatory. This standard must define a common interface and communication protocol between IP cores and system-on-chip interconnects, resulting in a standard way for an IP core developer to deliver his product. The IP core becomes naturally split between the core logic, which includes the native functionality of the core, and an interface module, which aims at facilitating the integration phase. Not only this allows the core developer to design an IP core to a known delivery standard without having to consider the destination platform of the IP core, but multiple interface standards can also be targeted, simply by changing the interface

module.
On the other end, by choosing one specific core protocol for his platform, instead of spending a lot of effort adapting cores to the new system, the system integrator can focus on system level design issues. Since the cores are now decoupled from the on-chip interconnect and hence from one another, it becomes also easy to swap one core for another to match system requirements changes.
To meet these decoupling goals, a standard core interface must be core-centric as well as system interconnect neutral, so that the entire IP core rework can be virtually eliminated.
A candidate to a core interface protocol standard must also:

- Capture all of the core-system signaling
- Be process independent, yet have timing guidelines
- Have a scalable performance, adapted to the core
- Be configurable

Many unsuccessful attempts in the past were restricted to the definition of the data flow signaling, ignoring how crucial it is to capture exhaustive core information including control wires such as interrupts, error signals, flow-control signals, test and debug signals, or endianness behavior for instance.
Finally last requirement but not least, the protocol must be freely available and non-proprietary to promote the widest adoption by IP core developers, system integrators, and EDA tool providers.

2 Open Core Protocol 2.0

The Open Core Protocol version 2.0, released in September 2003 by the OCP-IP specification working group, adds many enhancements to the 1.0 specification currently used by the OCP community [1][3]. That makes OCP2.0 be the best fit to the requirements list discussed in 1.
As its predecessor, OCP2.0 is a point-to-point, master-slave interface between two communicating entities. The master sends command requests, and the slave responds to them. All signaling is synchronous with reference to a single interface clock, and all signals except for the clock are unidirectional, point-to-point, resulting in a very simple interface design, and very simple timing analysis.
OCP captures dataflow, as well as control and test signaling. To that purpose, an entity can be given either a system attribute, or a core attribute, independently of its master or slave property. This is illustrated in Figure 1. OCP is also fully scalable: A minimal OCP configuration is defined as the basic OCP, and a set of extensions can be enabled as needed to accommodate a particular core's requirements.
Major 2.0 improvements with respect to 1.0 include a new burst model, the addition of in-band signaling, endianness specification, enhanced threading features, dual reset facilities, lazy synchronization, and additional write semantics.

2.1 Basic Open Core Protocol

The basic OCP is a very simple interface, allowing simple cores to be plugged into a system. Besides the clock, it includes the request, which consists of command, address, write data signals from the master to the slave, and command accept, response, and data read signals from the slave to the master. Figure 1 shows a simplified view of an OCP interface. Address and data bus widths are configurable to match the core's requirements.

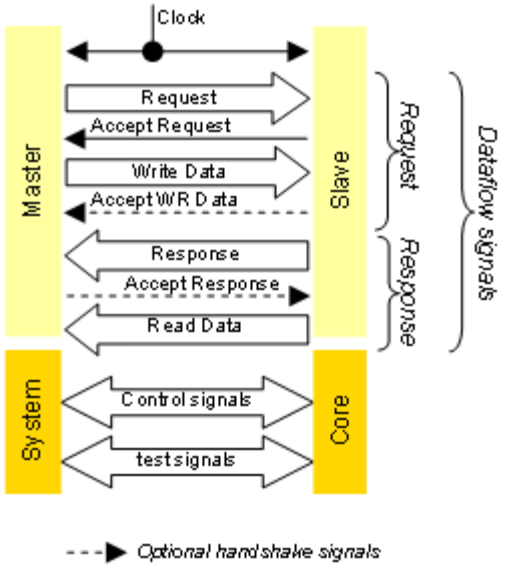

Figure 1: Open Core Protocol Overview

Commonly an OCP transfer is made of two separate and temporally decoupled phases, one for the request, and one for the response. Progression within a phase is controlled by a two-way handshake between the communicating entities. A phase begins with one side asserting the signals associated with that phase. In the illustration shown in Figure 2, the master asserts Request 1 during cycle 1, and Request 2 during cycle 3. The phase completes when the other side acknowledges that phase with a dedicated accept signal. In the example, the slave accepts Request 1 immediately (in cycle 1), while Request 2 is accepted a cycle after it is presented (in cycle 4). Any phase can be handled using a similar two-way flow control if required. The initiator of a phase is required to hold all signals associated with that phase steady until the target of that phase signals acceptance. This feature reduces the number of storage elements required to build an OCP interface.

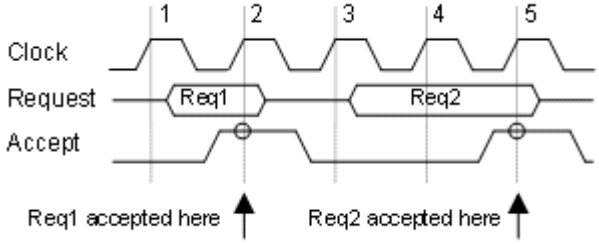

Figure 2: OCP Handshaking

The request phase always includes some flow control, based on a command/command accept pair of signals, whereas flow control (response/response accept) is optional for the response phase.
During a write operation, some cores may benefit from a separate phase for the data transfer in order to issue the transfer address as early as possible. To that purpose, a third independent and decoupled phase known as the data handshake phase can be added. This phase is controlled by a similar set of flow control signals and is optional. OCP phases are summarized in Figure 3.

By decoupling the different phases from one another using a separate handshake signal, an

OCP master can send multiple requests before it has received the first response. This pipelining and true split transaction capability allows for increased performance in the face of very long latencies that might, for example, be encountered when accessing off-chip DRAM. Pipelining is a key performance enabler in complex systems, and is generally used in conjunction with OCP burst transactions, described in section 2.3.
During the response phase, an indication about the transfer completion is sent back to the master. Report possibilities are “Valid”, indicating that the transfer has completed normally, “Fail”, reserved to the Write-Conditional command and indicating that the write has not been done, or “Error” if a transfer error condition has been detected.
To simplify the interface, responses must be returned in order. For high performance cores that require out-of-order responses, OCP also supports multi-threading. This concept is more complex and discussed later in the document, section 2.4.

The two basic operations, i.e. read access and write access, have been mapped onto seven OCP commands: Read, Write, Write-Non-Posted, Read-Exclusive, Read-Linked, Write-Conditional, and Broadcast. Using specific command encodings, different flavors of the two basic operations can be supported. For instance, a write operation may be given posted or non posted semantics, depending on the system requirements. When an OCP interface only uses a posting scheme, the presence of a response to Write command is optional. Support of Write-Non-Posted and/or Write-Conditional always assumes a response is given to any write issued on the OCP interface. The Broadcast command is a special write operation used when more than one entity is being targeted with the request.
To address system synchronization needs, i.e. between multiple tasks running on a single or several processors, OCP offers two different mechanisms: Read-Exclusive commands are used for atomic read-modify-write accesses, and therefore are always associated with a write transfer. This scheme is known as “locked synchronization”, as the target shared resource (typically a memory) is not accessible by the other requestors during the whole transaction. For systems including the latest generation of DSP or RISC processors, a “lazy synchronization” scheme is also supported using the Read-Linked and Write-Conditional commands. In that particular case, the shared resource is not blocked between the read and the write accesses, resulting in better overall system performance.
For Write, Write-Non-Posted, Write-Conditional and Broadcast commands, the write data accompanies the request. For Read, Read-Exclusive and Read-Linked commands, read data accompanies the response. Since the OCP interface is unidirectional, there are two separate data buses for read and write data.
Relationship between commands and phases is summarized in Figure 3.

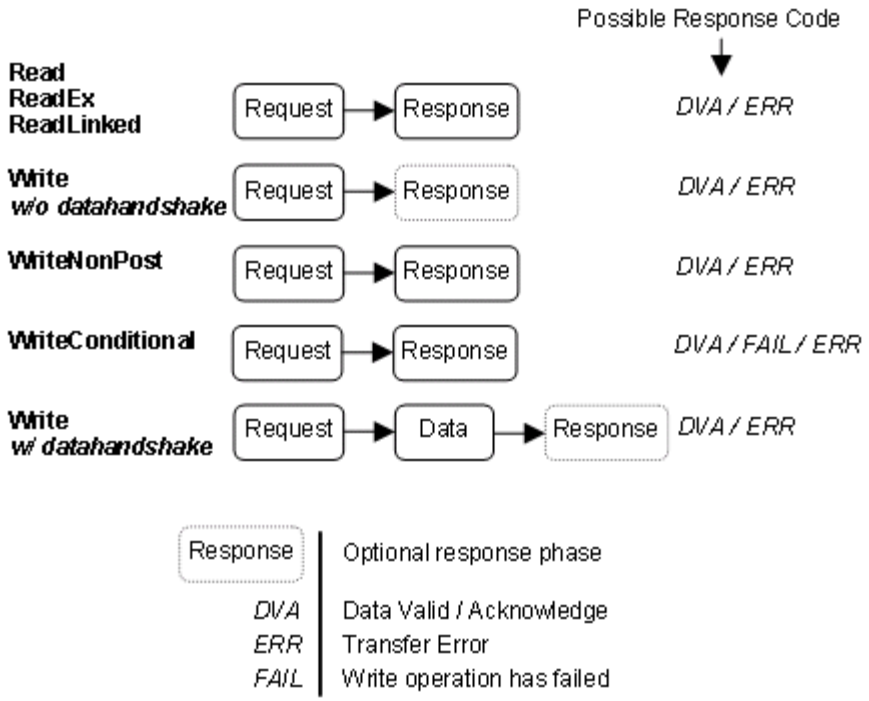

Figure 3: OCP phases within a transfer

The OCP commands are generally accompanied by an address field. In a standard system, it is the responsibility of the chip interconnection network to route the request to the appropriate target, based on this address. As an illustration of the OCP flexibility, it is interesting to point out here that it is perfectly legal for a local OCP interface not to include any address information.
More generally, an OCP interface is defined with a very comprehensive set of parameters, which allows enabling/disabling most of the signals individually. As a consequence, commonly-used interface models such as read-only, write-only, synchronization-only or FIFO interfaces can be easily described. The OCP interface parameters are defined in a set of configuration files, shared by the different tools in the development environment as a fully executable specification.

2.2 Open Core Protocol simple extensions

OCP simple extensions include byte enables, multiple address space capability, as well as the addition of in-band socket- specific information to any of the three OCP phases, i.e. request, response, and datahandshake (if any).
Required by many SOC applications, byte enables signals can be optionally added to the interface, and are driven during the request phase, for both read and write operations. This allows for byte addressing capability, or partial OCP word transfer. When a separate datahandshake phase occurs, a separate set of byte enables can also be associated with that phase.
Multiple address spaces can also be defined within a target. Often, there are address spaces with unique properties or behavior that tend to get located in different places in the system address map. Address spaces are typically used to differentiate a memory space within a core from the configuration register space within that same core, or to differentiate several cores within an OCP subsystem. From the top level system perspective, the multiple OCP cores that are part of the subsystem can then be mapped at non-contiguous addresses.

The third simple extension is the possibility to attach core-specific in-band information to any of the transaction phases. These custom signals are not assigned explicit semantics by the OCP protocol, but are assumed to behave as any other signals of the phase they belong to. This flexible extension is extremely powerful for building platform-specific signaling, while maintaining a well-defined protocol. Typical usages are cache signaling information, debug versus application differentiation, dynamic endianness management, or access permission control.

2.3: Open Core Protocol burst extension

The concept of burst exists to tie multiple OCP transfers into an OCP transaction. As a superset of the OCP1.0 burst, the OCP2.0 burst model targets both high flexibility and high efficiency:

- Ability to handle precise bursts (the length is known) and un-precise bursts (the length is unknown).
- Ability to specify standard address sequences (incrementing, wrapping, streaming, XOR) as well as custom address sequences.
- Ability to support single request/multiple data transaction models.
- Ability to define atomic sub-units within a burst for fine control of the request interleaving throughout the system interconnect.
- Ability to add complete framing information with all transfer phases.

In a multiple request/multiple data burst, each transfer within a burst is a complete OCP transfer (including command, address, etc.) but also contains information that specifies the number and addresses of future transfers in the burst. In a single request/multiple data burst, only the first address is transmitted, the burst sequence code is used by the target to reconstruct the expected address sequence.
An OCP target can use the burst information to prefetch data not yet requested. Bursts raise the efficiency of transfers to such clients as DRAM controllers, where the latency between accesses to different banks is extremely context-sensitive.

2.4 Open Core Protocol threading extension

For complex cores that may require multiple outstanding requests to targets with different latencies, such as a multi-channel DMA engine for instance, the OCP supports the concept of threading. In a single-threaded OCP, requests can be pipelined, but responses must be returned in the order of the associated requests. A multi-threaded OCP retains this strict ordering rule within a thread, but requires no ordering between different threads. Cores can launch requests on different threads and have responses returned out of order. Individual requests and responses are tagged with a thread ID in order to associate them with a given thread. While the concept of a thread applies only to the ordering across a single OCP, the concept of a connection goes end-to-end between cores. It allows information relevant to a request to be sent from initiating core to target core. To that purpose, OCP defines another ID, the connection ID.
Multithreading is also a very powerful way for reducing the number of wires in a system, as routing congestion and physical effects become a major issue at the backend stage of the ASIC process. In practice, many functional connections between initiator/target pairs do not require the full bandwidth of an OCP link; thus, sharing the same wires between several connections, based on functional requirements as well as device floor planning data, is an

extremely attractive tool for the system architect to make the appropriate gate count versus performance versus wire density trade-off.
Multithreading comes with its own set of signals to identify the different request, response and data threads within the system, and to provide thread status information (OCP "thread busy" extension). Treated as a hint only in the OCP1.0 protocol, the "thread busy" information can be given tighter exact semantics in OCP2.0: If the exact behavior is enabled, it becomes illegal to send a request or a response to a busy thread. As a consequence, the flow control on such an interface changes from the two-way handshake described in section 2.1, to a model driven by the receiver ability to accept the transfer within the cycle.
The exact semantics is vital to construct complex multithreaded interconnection networks, while guarantying that no dead lock condition can occur.

2.5 Open Core Protocol signals list

As an illustration of the protocol overview covered in the previous sections, Figure 4 provides the list of the OCP signals. There is no attempt here to describe in detail each of these signals. In order to get such an in-depth understanding, the reader should refer to the Open Core Protocol specification version 2.0 [5].

The specification provides exhaustive documentation information, most notably about:

- The OCP concepts
- The signals and encoding description
- The protocol semantics
- Examples of timing diagrams
- Behavioral models
- Configuration files format and syntax
- Developer's guidelines
- Timing guidelines

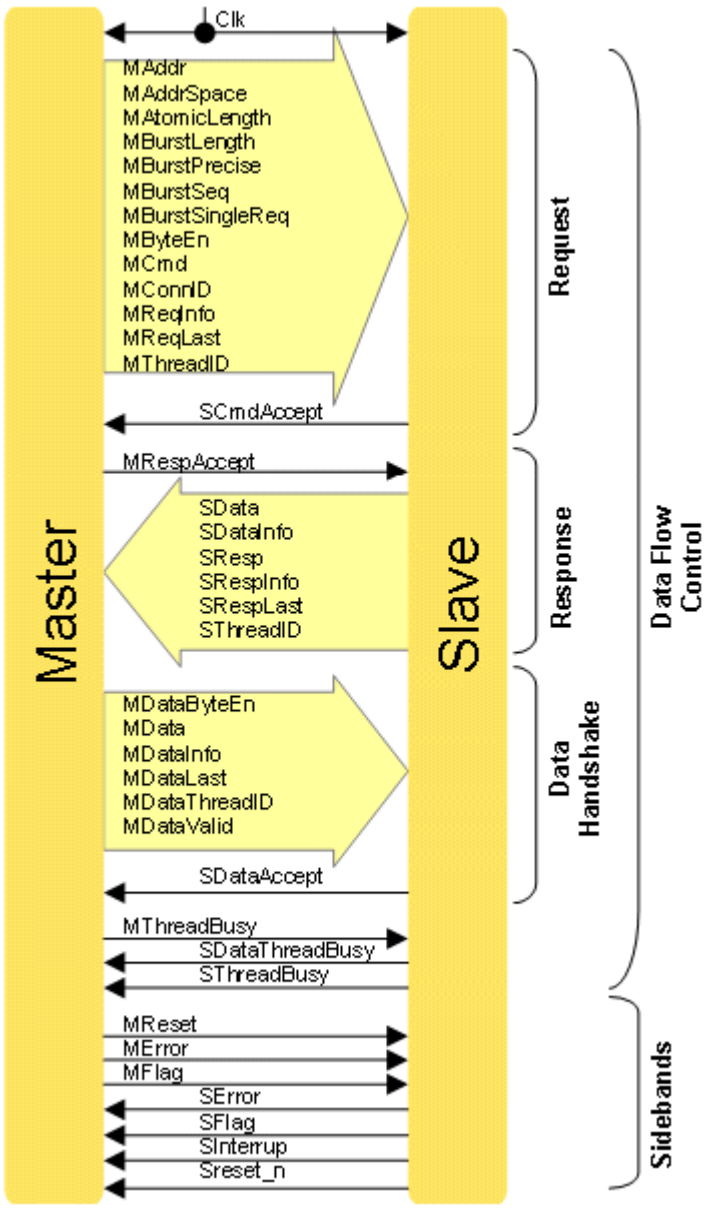

Figure 4: OCP signals

3 Open Core Protocol as a system socket

So far we have only discussed the dataflow portion of the OCP interface. To satisfy a true system socket definition, sideband signals and test signals must also be considered. This aspect is often left aside by typical computer-bus style interface specification, requiring the system integrator to deal with them in an ad-hoc manner for each system design. The OCP explicitly supports sideband signaling. Special signals are set aside for interrupts, errors, and control/status information. Reset is also part of the sideband signaling: A dual reset model can be implemented, so that the master (respectively slave) OCP can be aware of the reset status of the slave (respectively master) OCP which it is connected to. In addition, a generic flag bus can be used to accommodate core or platform specific control signaling needs.
The test interface is also part of the optional OCP extensions. This interface supports scan, JTAG, and clock control, providing a complete interface for manufacturing test and debug of the core when integrated into the system-on-a-chip.
As a socket, OCP is typically used to interface a set of cores to an on-chip interconnect

system. As illustrated in Figure 5, OCP acts as the boundary between the IP cores and the interconnect.
While it would appear to be a lot of work for the system integrator to provide a specific OCP interface implementation for each different core in the system, the generation of these system OCP wrappers is limited by the choices offered by the OCP protocol. Only a regular set of wrappers needs to be provided, rather than an individual adaptation to very different core interfaces. The wrapper structure is regular enough to be generated from an automated tool, leading to a full automation of the complete SOC interconnection network. The protocol used internally to the interconnect IP is not necessarily OCP-compliant, but usually an OCP-derivative optimized for micro-network architecture.
This automation capability, key benefit inherited from the standardization approach, can be extended to the verification phase, resulting in a highly productive design process, in a true plug-and-play environment.

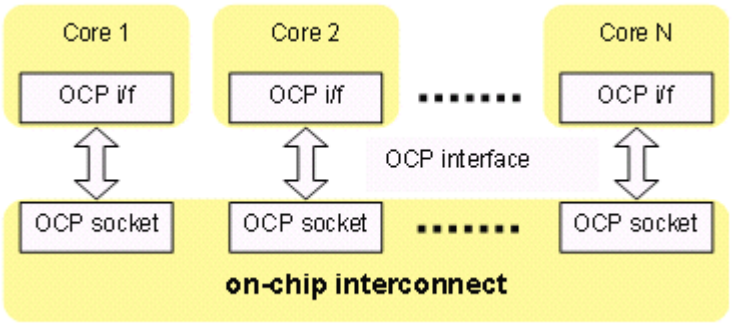

Figure 5 *OCP-based interconnection*

4 Related Work

The idea of a standard core interface and protocol has been originally promoted by the Virtual Socket Interface Alliance (VSIA), aiming to "dramatically accelerate system-chip development by specifying open standards that facilitate the mix and match of Virtual Components from multiple sources" [4].
The VSIA on-chip bus development group has been putting forth a virtual component interface specification (VCI) [2]. But for some reasons (the VCI specification is limited to the dataflow portion of a core's interface, thus new custom interfaces once again have to be invented on a per-design basis for control and test signals, lack of tools, …), VSIA failed into establishing a worldwide core interface standard. A recent press release [7] has announced a strategic alliance formed by the VSIA and OCP-IP organizations: VSIA has endorsed the OCP interface, and OCP-IP has become the first VSIA adoption group. Such alliance will considerably boost the OCP adoption in the coming years.
It is also interesting to notice that other potential candidates for a standard, such as the AXI protocol recently unveiled by ARM [6] (AMBA Rev2.0 does not meet all the requirements for next generation SoC devices), is sharing many basic concepts with the Open Core Protocol, reinforcing the pioneer role of OCP in the field of core interface protocol.

5 Summary

This paper has presented the concepts and features driving the Open Core Protocol 2.0, a core interface aimed at capturing all of a core's system communication needs. The key concept is "decoupling". The Open Core Protocol is simple, synchronous, and point-to-point. It is highly

scalable and configurable to match the communication requirements associated with different IP cores. A basic OCP is appropriate for simple cores, while complex high-performance cores can be accommodated efficiently with OCP extensions.
Referring to a well-specified interface between IP cores and on-chip communication systems, the work of the IP core developer is decoupled from that of the system interconnect developer and system integrator. Cores with OCP interfaces and wrapped interconnect systems enable true plug-and-play approach and automated design processes, thus allowing the system integrator to choose the best cores and best interconnect system. Such capability is a major foundation of a platform-based design methodology.
The Open Core Protocol is on-chip interconnect independent and its specification is freely available to the design community. The Open Core Protocol is a living standard, and will receive the appropriate updates in the future to continue meeting the SOC design requirements, in terms of performance and design methodology, including design, verification and modeling tools.

The paper presented results from the work of many members of the OCP-IP Specification Working Group, most notably Wolf-Dietrich Weber (Sonics), Drew Wingard (Sonics), Anssi Haverinen (Nokia), Tarek Zghal (Texas Instruments) and Franck Seigneret (Texas Instruments).

References

[1] OCP-IP. Open Core Protocol Specification. Version 1.0, 2001, www.ocpip.org

[2] Virtual Socket Interface Alliance, On-chip Bus Development Group. Virtual Component Interface Specification, www.vsi.org

[3] Wolf Dietrich Weber, Enabling Reuse via an IP Core-centric Communications Protocol: Open Core Protocol. IP 2000 conference.

[4] Virtual Socket Interface Alliance. Vision Statement. www.vsi.org/benefits.htm#vision.

[5] OCP-IP. Open Core Protocol Specification. Version 2.0, September 2003, www.ocpip.org

[6] ARM. AMBA Specification Rev2.0, www.arm.com

[7] EETIMES. OCP, VSIA join forces for SoC interconnect, www.eetimes.com/story/OEG20031007S0017