Bringing Formal Property Verification Methodology to SoC Interconnects

Mohamed Ayoub DHOUIB* and Ridha DJEMAL **

*Science Faculty of Monastir, 5019 Monastir
Telnet Corporation Tunis
ayoub@yahoo.fr

**King Saud University, Electrical Engineering Department
PO Box 800 Riyadh 11451 KSA
rdjemal@ksu.edu.sa

Abstract: With many system bus alternatives in telecom, signal processing, etc, chip designers face the prospect of having to support multiple interfaces to meet interconnect requirements. Designers must then build next-generation chip architectures that deliver reliable interconnect architectures and ensure interworking between SoC heterogeneous IP blocks. In this article we show how formal verification associated with assertions can help and automate the functional verification of SoC interconnects, and particularly bridges: transactions translation checking, functional performance analysis, address/data consistency checking. With our proposed methodology, these tasks can bring more automation and robustness to the functional verification of SoC interconnects.

Key words: DUT: Design Under Test, SoC: System on Chip, SVM: System Verilog Assertions.

INTRODUCTION

The process of developing appropriate verification environment for SoC designs becomes nowadays really tricky, due to the fact that they involve a number of different protocols and include numerous heterogeneous IP blocks. These IPs range from general purpose RISC processors, DSPs, MPEG/JPEG processors, and distributed memories to communication cores and specialized instruction-set processors. With such an increasing design complexity, verification tends to consume up to 70% of project resources and often represents a bottleneck.

In today’s SoCs, the most frequently used on-chip interconnect architecture is the shared medium arbitrated bus, where all communication devices share the same transmission medium. In a bus-based SoC, multiple IP blocks share the transmission media, have generally heterogeneous interfaces and communicate through different communication protocols such as AXI, OCP, AHB, etc. For SoCs consisting of tens or hundreds of such IP blocks, interconnect architectures will lead to serious bottleneck problems and likely communication errors. As the number of connected IPs increases, their global synchronization will require more effort spent on the interconnect verification.

To cope with the rising interconnect complexity and the increasing time spent on its functional verification, research in applied formal verification has become a hot topic in system design. Major hardware bugs found may cause expensive project delays when they are discovered during system test on the real silicon chip. The consequences are severe, from cost overruns to lost market opportunity. Simulation and emulation tools, which are traditionally used to find bugs in a design, often cannot find the corner cases or hard-to-find bugs that may occur only after hundreds of thousands of cycles, and are well beyond the reach of conventional simulation and emulation technologies. Formal methods have emerged as an alternative approach to ensure the quality and correctness of hardware designs, overcoming some of the limitations of traditional validation techniques such as simulation and testing.

1. Formal Verification Background

Formal verification associated with assertions is a well known approach to functional verification of SoCs [DAL03a]. An assertion is typically a statement about how a particular design should or should not behave at the RTL level. Assertions have the
expressive power to enable concise descriptions of complex behaviours. To describe a design feature within an assertions’ framework, first its functionality must be understood, then its behaviour must be explicitly described, and finally, the conditions under which this behaviour is applicable must be stated. If an assertion is violated, it means that a bug has been found (either in the design or in the assertion).

Unfortunately formal verification and assertions are not so easy to bring into operation. First of all, assertions can be hardly and not thoroughly extracted from the specification, and very hard to elaborate. This becomes really problematic if assertions are wished to take a central part in verification, in which case a high amount of assertions must be extracted from the specification.

When using formal verification, there is in addition the necessity to create assertions that model the environment. In fact, the environment represents the set of stimuli that are legal at the boundary of the DUT. This description is called constraints, since it constrains the formal engine to analyze only legal stimuli. Without constraints, a formal engine will report false errors instead of design bugs. Therefore, there is a necessity to create assertions to model the environment, called environment constraints. This task gets complex when using formal verification on a wider selection of properties. The environment behaviour needs to be extracted from the written specification. The created assertions are then verified, and are automatically transformed into constraints for formal verification. Getting constraints correct is important for quality verification results. Erroneous assertion coding leads many times to dead-locking the environment [DAL03a]. Moreover, under constrained inputs lead to false counterexamples and wasted debugging time. In fact, when the verification engines are not constrained sufficiently, illegal input sequences can produce false counter examples.

The fundamental idea we propose here is to build ready-to-use a formal verification kit to automate the verification of interconnects, and specifically bridges. A formal verification kit is much more than a simple library of assertions and performance properties, as it is highly configurable. We detail further the interconnect formal verification kit aspects and features in section 3.

Once the interconnect kit is available, formal verification can be applied for automatic transactions translation checking, automatic address/data consistency checking, and automatic functional performance evaluation.

2. Assertion-Based Verification

As formal research matures and approaches a level of sophistication required by industry, a change of design methodologies is becoming a must. A move from ambiguous natural language forms of specification to forms that are mathematically precise and verifiable is then required. Furthermore, these languages must lend themselves to automation. Formal property specification is the key ingredient in this methodological change.

Formal verification is the natural approach for an assertion-based methodology. Due to its underlying algorithmic and exhaustive capacity, formal verification can compute proofs (usually bounded) that a property always holds, or will automatically compute a short violation sequence. The coverage scenarios are computed automatically without the need of test-benches [DAL03b].

Assertions have been initially written using Verilog or VHDL to describe design properties and have been combined to the design to be simulated [LAH06a/ LAH06b]. As HDL descriptions of assertions increase quickly, complex to develop and hard to maintain, more suitable means emerge to permit a wide and flexible use of assertions. First, generic HDL components for assertions have been first gathered into libraries OVL [ACC01a]. Then, well defined languages dedicated to assertions have emerged and conquered the users. These languages are based on appropriate constructs among which temporal operators with an accurate semantics and lead to concise description that can be efficiently treated by verification algorithms. Major assertion languages are PSL [ACC01c] and SystemVerilog assertions (SVA) [ACC01b]

Several books are dedicated to assertions [FOS04]. In this article we define assertions as a description of a design’s functionalities in terms of either:

- Behaviour properties : that must always hold on the design (e.g. the design always translate INCR16 bursts into 16 consecutive single transfers)
- Performance analysis properties : that target a performance evaluation on the design (e.g. an incoming WRAP4 transaction on interface x takes at most 16 clock cycles to appear as an outgoing transaction on interface y);

Both assertion types are important in order to check which design functionalities are correct.

3. Interconnect Verification Kit

With recent on-chip protocol generations such as AHB, OCP or AXI that enables reuse, IP blocks are not aware of the whole system in which they are integrated, and the interconnect is the functional backbone of the entire system. Any control-related problem, being a functional bug within an IP (control error, etc) or a functional incompatibility between IPs (latency, etc) shall be detected either at the bus protocol level or at the interconnect level. The bus protocol layer is one focus where some SoC design functional verification activities can be monitored. Nevertheless, the interconnect layer can be also a corner stone for any functional verification task.
A formal verification kit is for formal verification and assertions what a Verification IP is for simulation [DAL03b]. It includes complete documentation, automated environment and assertions creation, automated scripts and report generation, all targeted at analyzing the various interaction between the different ports of a SoC interconnect.

The formal verification interconnect kit is with a great interest for checking the interaction between two interfaces present on the same DUT (interconnect, bridge ...). The interconnect kit enables the verification of two main assertion types:

- Transaction and address/data consistency assertions which describe the properties that should always be true when checking the interaction between two interfaces on the DUT.
- Transaction performance assertions which describe the properties that should always be reachable, and that include functional performance goals which evaluate the longest time spent on various transactions while interfaces interaction.

The interconnect kit creates automatically properties that check the interaction between two interfaces on the same DUT. This kit is appropriate for testing SoC bridges and interconnects, and can be used with or without interface regular Hardware Protocol Kits (HPK) such as the AHB, the OCP and the AXI HPKs. The interconnect kit just require stating the DUT first interface and the second one types (interface protocol and port direction), it will then automatically map all the available properties accordingly to the used HPK interfaces configuration files.

Some parameters are required to be defined to select which kind of transfers are to be checked such as bursts translated in singles, in bursts of same size, or in bursts of different size, etc. The Figure 1 shows how the interconnect kit can be used to check the interfaces interaction on an AHB to OCP bridge DUT. On this example, the interaction kit will provide properties and scenarios that evaluate how an AHB transaction will go through the OCP side. For instance, we can check if any AHB burst is translated as many singles on the OCP side, or also evaluate the best/worst time needed to see the burst going through the OCP side.

![Figure 1. Interconnect kit used for an AHB2OCP bridge verification](image)

The interconnect kit is also appropriate for verifying SoC interconnects by checking the interaction between the different interfaces of the interconnect DUT in a coupled manner. The Figure 2 shows how the interconnect kit can be used to verify an AXI interconnect. Regular HPKs, namely the AXI HPK, builds the verification environments for the different AXI interfaces (master and slave ports), whereas the Interconnect Kit spies on the different transactions exchanged between the appropriate couple of interfaces and checks the consistency of the transaction translations.

![Figure 2. Interconnect kit use for an AXI interconnect verification](image)

To make more flexible the interconnect kit usage, assertions are automatically selected and sized according to a set of parameters (the kit configuration file) that can be filled in in text mode or through graphical GUI. A configuration file is needed per a couple of in/out ports.

The interconnect kit supports many types of master/slave ports including AHB, OCP, AXI and APB. A regression script is then generated and a detailed report enables to quickly evaluate which assertions passed or failed.

4. Application

The interconnect verification kit has been applied with formal verification on many blocks ranging from a few tens of Kgates to several hundreds of Kgates, most of them are bridges. These blocks are typically bridges that present a unique couple of interfaces. The first one is typically a slave interface that adheres to a given communication protocol, whereas the second interface is a master one communicating using the same or a different protocol. The interconnect kit can also be applied on blocks having more than two interfaces, and this can be ensured by applying the kit on each couple of in/out interfaces.

The typical setup time of the interconnect library, when a good description of the DUT interface characteristics is at hand, is of a few tens of minutes. The formal verification environment is automatically inferred by the regular HPK libraries with the set of parameters that describe the interface characteristics and the behavior of the environment entities. We present some samples of the verification results...
obtained by applying the interconnect formal verification kit on an AHB2AHB and an AXI2OCP bridges.

4.1. The AHB to AHB Bridge IP

The AHB to AHB bridge allows designers to perform a hierarchical AHB bus organization in order to distribute the communication flow among some child AHB interfaces working in a parallel way instead of concentrating all the communication across the main AHB bus. This bridge can be typically used for creating child AHB buses in order to improve the whole system performances [DJE05]. In fact, each child AHB bus can manage local communication without the need to access to the main AHB bus. The AHB to AHB bridge supports only incrementing bursts [ARM02a].

After preparing a configuration file that specifies the parameters characterizing the bridge behaviour in terms of translation rules and latency values, a complete regression has been launched to check the interaction of the two bridge interfaces. Verification environments, on either side of the bridge, have been built automatically by the AHB Hardware Protocol Kit [DAL03b].

Figure 3 shows an example of violation sequence that was highlighted through the application of the interconnect kit. The violated property states that the bridge should bring out any incoming transaction within two clock cycles at most. The AHB to AHB bridge, which translates bursts of length N to bursts of same types and lengths, took three clock cycles as depicted in figure 3. All of transactions translation consistency properties have been successfully proven on the bridge DUT.

Figure 3. A violation VCD of a worst case performance property

4.2. The AXI to OCP Bridge IP

The AXI to OCP bridge is a communication converter from the AMBA AXI protocol to the freely available Open Core Protocol (OCP). OCP is a highly-configurable bus-independent protocol that meets all of an IP core’s communication requirements. This bridge allows the conjunctive use of AXI-based and OCP-based IP blocks in the same SoC design. The use of such communication translators between different communication protocols is becoming a must, since IP designers strive to ensure their IPs can be used by the widest possible range of applications.

The AXI to OCP bridge presents an AXI slave interface and a master OCP one, and supports single transfers and all types of bursts : incrementing, streaming, and wrapping. This bridge transforms any incoming incrementing or wrapping burst on the AXI side to a burst of the same type and length on the OCP side. It translates however any streaming burst (fixed address bursts) of length N to N consecutive OCP single transfers [ARM02b][OCP07]. Figure 4 shows a violation VCD of a transaction consistency property which states that : any incoming wrapping burst of length eight should be computed as a wrapping burst of length 8 in the OCP interface. This property did not hold on the AXI to OCP bridge and is considered as a design bug.

Figure 4. A violation VCD of a transaction consistency property

5. Conclusion

The increasing complexity of nowadays SoC designs has created a verification crisis. Engineers cannot imagine all of their design’s possible corner-case behaviours. We showed that the application of formal verification to verify interconnects behaviours and interfaces’ interaction has been proven. We illustrated also that, when appropriately applied, formal verification strategies are a powerful verification methodology which contributes to increasing design quality and shortening time to
market. We described how formal verification, when using a verification kit can efficiently solve design verification tasks by checking heterogeneous interfaces interaction and performing functional performance analysis. For such areas of the verification flow, formal verification is now a must-have.

REFERENCES


[DAL03b] S. Dellacherie: Bringing Automation to the Verification of SoC Based Designs, GSPx, 2005.


