CompMon: Ensuring Rigorous Protocol Specification and IP Compliance

Robert Adler
Intel Corporation
3600 Juliette Lane
Santa Clara CA 95054
408 765 0072
robert.p.adler@intel.com

Sava Krstic
Intel Corporation
2111 NE 25th Ave
Hillsboro OR 97123
971 214 1708
sava.krstic@intel.com

Erik Seligman
Intel Corporation
2111 NE 25th Ave
Hillsboro OR 97123
503 712 3134
erik.seligman@intel.com

Jin Yang
Intel Corporation
2111 NE 25th Ave
Hillsboro OR 97123
971 214 1735
jin.yang@intel.com

ABSTRACT

Intel defines numerous forms of reusable IP that are leveraged by many projects across different divisions and business groups. In order to ensure the successful reuse in the various system topologies demanded by Intel design teams, compliance of the IP to the specification is critical. In this paper, we describe a compliance methodology and flow developed for a generic interconnect fabric protocol. We show how to annotate and automatically extract specification compliance rules in a pre-1.0 evolving specification. We describe a standalone tool-agnostic SystemVerilog (SV) compliance monitor **CompMon** that implements all the compliance rules that can be checked on a single interface of the protocol. The monitor is written in SystemVerilog and can be used in any pre-silicon validation flow to ensure a single, cohesive compliance standard. We describe a rigorous approach to check that the monitor and the set of compliance rules it implements are complete, consistent and correct. We show how we enabled consistent compliance checking across groups and easy adoption of the compliance checking methodology into any design team validation environment. CompMon has been deployed to seven disjoint verification environments so far, both SoftIP providers and CPU design teams. Their experiences demonstrate the success of this methodology for ensuring compliance and successful reuse.

General Terms

Standardization, Verification.

Keywords

Validation, Formal Verification, Simulation, Standards, Compliance, Protocols

1. INTRODUCTION

IP blocks designed to be compliant to our IO fabric interface specification are meant to be reusable with minimal incremental effort. Our internal standard defines the interface signals and key architecture elements: interface instantiation, the protocol used for information exchange between compliant IP blocks, the arbitration and flow control mechanism to initiate and manage information exchange, the address decoding and translation capability supported, the way power is managed, and the hooks required for validation/debug. Figure 1 shows a generic architecture using our I/O fabric.

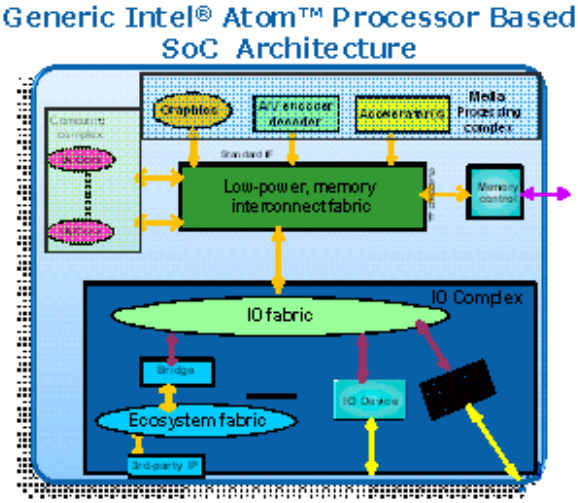

Figure 1: Generic Intel® Atom™-based SoC Architecture

In our terminology, IP blocks are called agents, and they are connected to the fabric via our standard interfaces. The topology and internal workings of the fabric are product specific. A design that integrates compliant IP blocks has the flexibility to implement topologies that meet specific requirements and constraints for that product.

The wide adoption of our IO interface standard, with the ensuing proliferation and reuse of compliant IP blocks, puts an emphasis on the quality of the architecture, its specification, and its validation methodology. When we establish that an agent is compliant to the message interface specifications, we want this to guarantee interoperability with arbitrary fabrics and agents designed to the same standard. The validation collateral provided by the protocol team includes a compliance rules document, bus functional models (BFMs), and a compliance monitor. Figure 2 illustrates the basic structure of an interface using this standard.

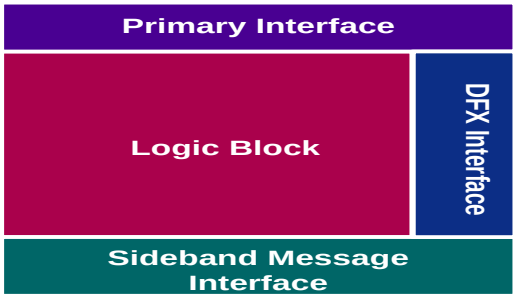

Figure 2: Basic interfaces in our standard.

The compliance monitor (CompMon) is the subject of this paper. It implements all the compliance rules that can be checked on a single interface. It is a standalone SV module and is written in the synthesizable subset of SV with SV Assertions (SVA) [1]. Consequently, it can be used with simulation, formal verification, and any verification/validation tool that supports SV and SVA. The most important features of CompMon are: (1) the tight linkage between the compliance rules that it implements and the actual text of the specification, and (2) the use of formal verification on its implementation. Formal verification of CompMon and its tight linkage with the specification give us high confidence that the set of rules it implements and the specification itself are close to the ideal of being fully complete, correct, and consistent. By enabling the integration of the compliance monitor into all pre-Si validation flows and by using it to verify the specification, the working group is able to provide a single, cohesive compliance standard that provides an overall validation strategy with high confidence in compliance.

The rest of this paper is organized as follows. Section 2 is an overview of the compliance methodology and flow. Section 3 describes our method for annotating and automatically extracting compliance rules from the evolving specification in order to provide tight linkage between it and the compliance monitor. Section 4 focuses on the architecture and key elements of the compliance monitor and Section 5 on its formal verification. In Section 6 we discuss the modes of use of the monitor, and in Section 7 we show how it was integrated into various validation flows and deployed to several projects, with results and impact. Section 8 concludes the paper.

2. Overview of CompMon Development

The heart of our specification is a set of explicitly annotated rules. Each rule is a property (statement) that is either true or false for any given waveform on the interface wires. The rules are intended to define the interface. If both the agent and the fabric obey the rules, then they will be able to properly communicate.

Along with the specification document itself, our working group provides a compact list of all compliance rules that are defined in the specification. This list is extracted from the specification by an automated procedure described in Section 3. Another automated procedure cross references all comments in the CompMon code against the compliance rules list. Thus at any point in time, we are able to get a concise picture of the rules implemented by CompMon and how they align with the rules in a given specification release. When a new specification revision is made, all compliance rules are automatically extracted into the compliance rules list and the rule list is automatically aligned to the rules implemented in CompMon in order to get an idea of the work required to update CompMon. The middle part in **Figure 3** shows the dependence between the specification, the rule document, and CompMon.

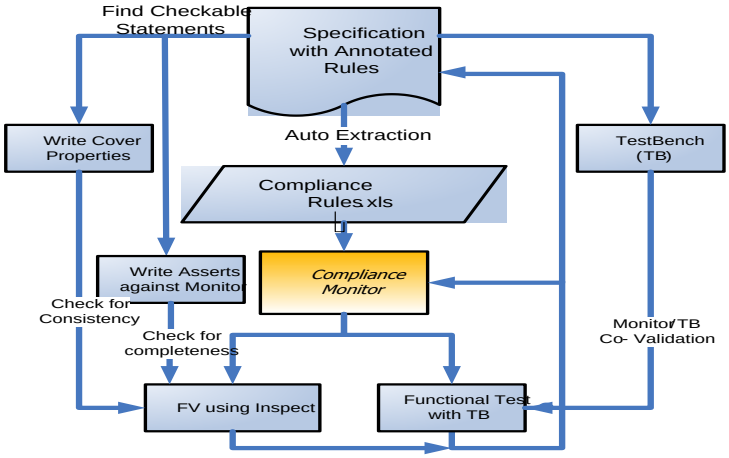

Figure 3: Development flow of the compliance standard.

To ensure that the rules implemented in CompMon are a faithful representation of the rules in the specification and to examine the specification itself against logical deficiencies, we use a thorough validation process that involves co-simulation with dynamic simulation test collateral (**Figure 3**, right side) and formal verification (FV) using the Inspect tool (**Figure 3**, left), a Formal Property Verification tool developed at Intel. The FV checks are based on a large number of checkable statements that we extracted from the specification (assertions and cover properties) and are an independent set from the set of compliance rules implemented in CompMon. Every failed check requires a fix in either CompMon or the specification itself. The whole process is detailed in Section 6.

3. Specification and Automatic Extraction of Compliance Rules

Like many specifications, we define the protocol via a textual description of the intended functionality provided by the interface as well as via a distinct set of compliance rules that compliant devices must follow. To be compliant, an IP block is required to obey all mandatory protocol rules and any rules that are

applicable to optional features implemented by the block. Compliance rules applicable to optional features not implemented by the block are not applicable to the block's functionality, and therefore are not a part of compliance testing for the block.

A rule that appears in the specification can either be categorized as a design rule, as a signaling rule, as a protocol rule, or as a transaction rule. Design rules are important specification rules that IP designers must meet in order to properly design their IP block. Design rules are either informatory or have to be checked manually by someone with intimate knowledge of the design's micro-architecture. Design rules are not checked by the compliance monitor and IP designers are required to provide information about the status of design rules. Signaling rules are rules that cover transitions on signals at the interface. It is possible to check signaling rules at the interface using simple assertions. Protocol Rules are rules that identify protocol requirements and define how a transaction is exposed by the interface. Transaction Rules are rules that identify agent requirements and expected behavior across multiple transactions.

Here are some example compliance rules in each category:

- **Design rule (not checked by CompMon)**: Before enabling an agent to initiate credit re-initialization, software must ensure that the re-initializing agent and all agents that can issue transactions to it are quiesced.
- **Signaling Rule:** When driving a command, all initiators must drive all reserved fields to 0.
- **Protocol Rule:** All agents must check address[0] when decoding Type 0 configuration cycles to make sure that it matches the value of its BusSelect strap.
- **Transaction Rule**: During transactions, the ID and Tag sent with the original request are returned with the completion.

In order that a distinct set of compliance rules be given to IP designers and validators, the design, signaling, protocol, and transaction rules are extracted from the specification. Originally, the extraction process was entirely manual, and verification engineers manually combed and interpreted the specification in order to extract the rules. For the dedicated rules sections, this manual process only involved replicating the dedicated rules into the compliance rules document. However, many of the in-line rules required rewording in order to make them into compliance rules because, in many cases, they were either not worded as compliance rules or they were combined with more informative text that was not appropriate for the compliance rules document. In such cases, the verification engineers were forced to create their own rule text that captured the essence of compliance rules that were contained within the specification text and not an exact copy of the text itself.

As the specification evolved, not only were additional features and rules added, but previous features and rules were tweaked, removed, or moved to different pages. When the manual process that originally created the compliance rules document was utilized to update the compliance rules document, a manual audit of each and every rule was required in order to make sure it was still consistent with the specification text and location. While this process was straightforward for the standalone rules that did not change, it consumed a large amount of time for any rule that was in any way different from its corresponding text in the updated version of the specification, and it was exceptionally difficult for the rules that were captured from the essence of the specification's text.

In order to address this maintenance problem, we created a novel way to auto-generate the compliance rules document directly from the specification Microsoft Word document. Any text that defines a rule is tagged with a comment and the body of the comment contains XML that defines important information about the rule such as its rule number and rule type, which can be extracted by a script. Because of this auto-generation method, maintenance of the compliance rules document has become the specification maintenance that the working group does anyway plus marginal incremental work to maintain the rule tags and to run the rule extraction macros when the specification is published. Beyond the maintainability benefits, this approach allows for easy extraction of the rule's location in the specification. The biggest disadvantage of the use of auto-generation is that it requires that all text in the specification that defines rules be written so that the text can stand on its own without any context from the specification. However, while this does require some extra work by the specification's authors, we believe that it is an improvement to the specification and that the extra effort is well worth the efficiency and accuracy provided by the auto-generation.

As an example of the auto-generation, there is a compliance rule in the document text stating

- **On a credit put, the maximum values of the credit command and data fields are 1 and 4 respectively.**

This is linked to the following piece of XML, as an annotation in Microsoft Word:

- **<RuleInfo><RuleNumber>**PRI5#7**</RuleNumber><RuleType>**Signaling**</RuleType></RuleInfo>**

This enables us to clearly link rule number **PRI5#7** to the text sentence. Regardless of future changes to the document format or content, as long as that sentence remains, our extraction script will always generate a rule like this:

Rule Type	ID	Spec Loc	Text
Signaling	PRI5#7	2.2.1.1	On a credit put, the maximum values of the credit command and data fields are 1 and 4 respectively.

A release of the specification published in December 2009 was the first revision of the specification to make use of auto-generation for the compliance rules and this method has been used in every release since. For a recent release in 2010, our working group was able to release a compliance rules document that contained 438 rules that were 100% in sync with the specification within minutes of the final approval of the draft.

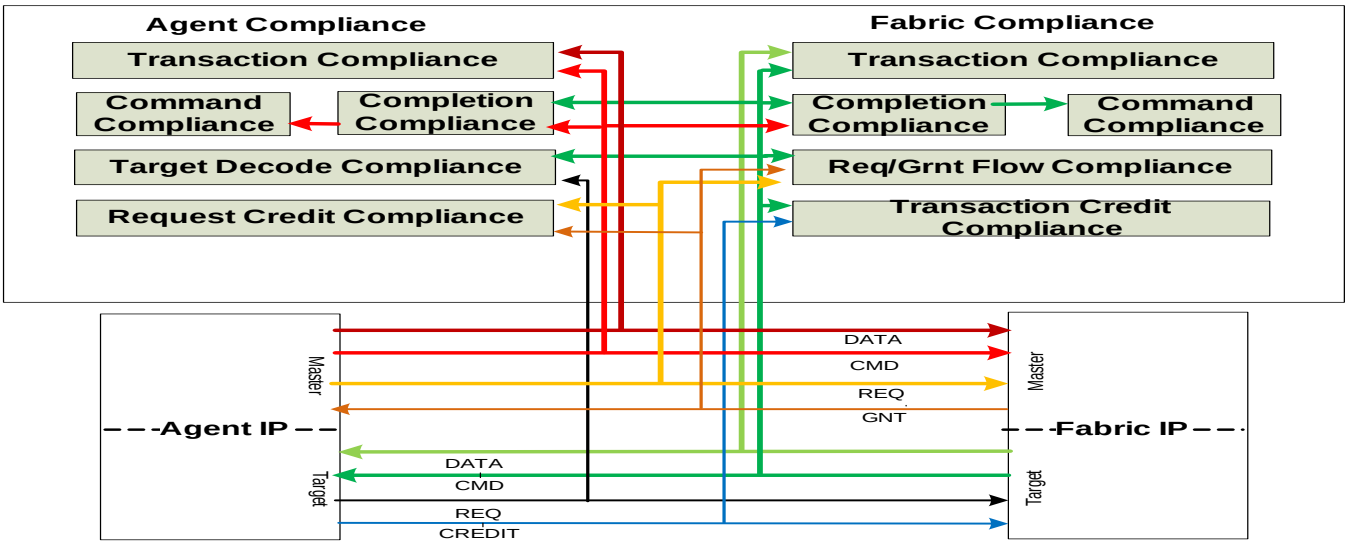

Figure 4: Organization of the Compliance Monitor

4. Compliance Monitor Organization

As shown in Figure 2, our protocol contains two main interfaces, the "primary" and "sideband". For each of these interfaces (primary and sideband), CompMon contains a standalone SystemVerilog module that implements all the compliance rules that can be checked on a single interface. The monitor is written in the synthesizable subset of SV with SVA. It is organized in a modular, layered fashion based on the protocol stack in the architecture specification. The following text provides a detailed discussion on CompMon for the primary interface. The monitor for the sideband is constructed similarly.

Figure 4 shows the main sub-modules in the monitor and how each sub-module monitors the signals on primary interface between the agent block on the bottom left and the fabric block on the bottom right.

The **Request Credit Compliance** module ensures that the request credits for the agent master are reset properly, and that they are correctly incremented. Similarly, the **Transaction Credit Compliance** module ensures that command and data credits for the agent's target interface are initialized and handled correctly. The **Target Decode Compliance** module ensures that the target decoding scheme works according to its compliance rules. The **Req/Grnt Flow Compliance** module ensures that the request-grant flow between the agent master and the fabric follows the set of flow protocol rules. The **Command Compliance** module applies to both master and target sides of the interface. It ensures that any command sent out on the interface is for a valid command type and that for each valid command type, the command is properly formatted. The **Completion Compliance** module also applies to both sides of the interface. It ensures that each command is completed properly according to the completion compliance rules. At the highest level, the **Transaction Compliance** module assembles a transaction from its command and data parts and ensures the integrity of the transaction and the correct pipelining of back-to-back transactions. By partitioning the compliance rules into these submodules, each submodule is responsible for a particular protocol in the architecture. For instance, the Request Credit Compliance module only cares about the proper handling of request credits.

There are several advantages of this modular approach. First, if some changes are made to one protocol or a new protocol is added to the specification, the only module that needs to be updated or added is the one corresponding to the protocol. Second, this approach enables a much more scalable approach for formally verifying the monitor in the future, as we can focus on one module while hiding all other modules by replacing them with necessary environmental assumptions. For example, if a change in a new protocol version does not modify the credit handling, we might want to abstract out that part of the protocol, assuming its assertions hold true, instead of re-verifying.

5. Verifying the Monitor and Specification

As mentioned in Section 2, the IO fabric compliance standard is expressed by the set of rules contained in the IO fabric specification. There are two fundamental ways in which this set may be deficient; it can be insufficiently constrained or it can be overly constrained. If the rules are insufficiently constrained, there is a behavior (waveform) that is undesirable but does not violate any of the rules. For the purpose of this paper, we consider this issue to be caused by the rules being incomplete. In order to make the rules more complete, either new rules need to be added or existing rules need to be strengthened. If the rules are over constrained, there exist behaviors that are desirable, but that violate rules. For the purpose of this paper, we consider this issue to be caused by the rules being inconsistent. Making the rules consistent requires that the offending rules either be weakened or deleted. Since CompMon is another expression of the compliance standard, the same completeness and consistency concerns apply to it as well. In addition, there is a correctness concern about CompMon: its set of rules should be a faithful expression of the standard, in the sense that CompMon would raise a red flag when checking a waveform if and only if the

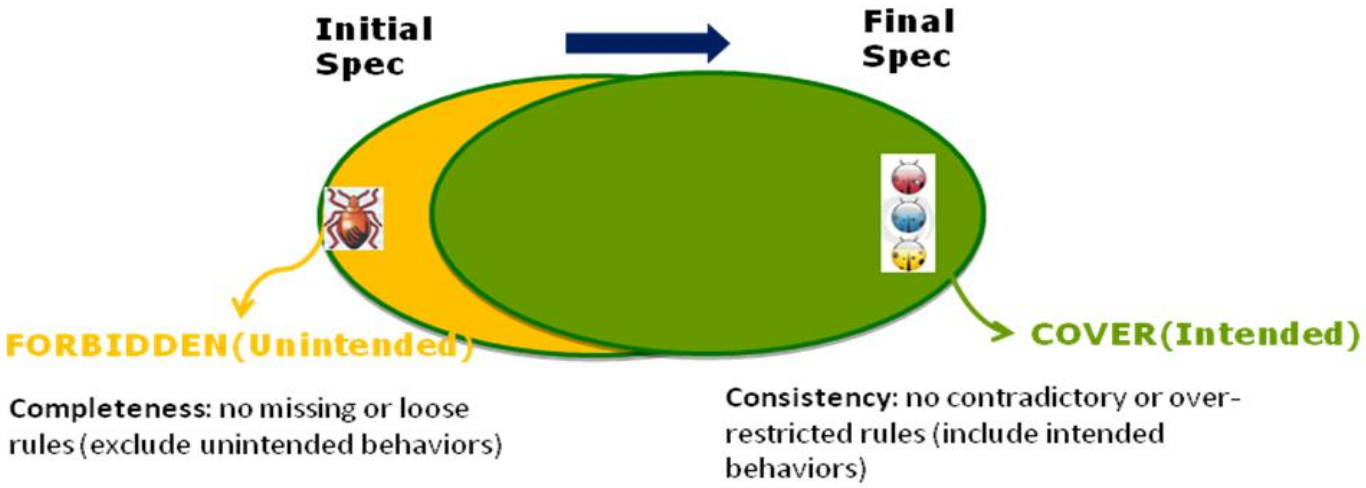

Figure 5: Towards Completeness And Consistency

waveform violates some of the rules in the specification. We refer to the completeness, consistency, and correctness as our 3C verification goals.

For illustration, consider Figure 5. The green set consists of waveforms consistent with the intent of the specification---an ideal that represents the architects' notion of what behaviors on the interface should be considered legal. The yellow set consists of waveforms that would pass the CompMon test. We would like these sets to be the same. Any yellow-but-not-green behavior (bug) indicates an incompleteness bug, and any green-but-not-yellow behavior (ladybug) indicates an inconsistency bug. Fixing any (lady)bug brings CompMon a step closer to matching the specification and the architectural intent.

Below we describe how we use simulation and formal verification (FV) to get CompMon closer to the ideal. Simulation exposes ladybugs; FV exposes both bugs and ladybugs.

- **Validating the monitor through simulation:** During CompMon development, we validated it by co-simulating with the team's simulation test generation collateral. CompMon flagged errors in cases where the observed traffic violated some of the assertions contained within CompMon.
- **Formally verifying the monitor:** In Formal Property verification (FV), one provides a design with a set of *assumptions*, a set of *assertions*, and a set of *cover points*. (All three are sets of properties.) An FV tool such as Inspect checks each assertion and each cover point individually. An assertion passes the Inspect test if and only if it is implied by the assumptions. A cover point passes the Inspect test when a waveform exists that complies to the assumptions and the cover point as well.

For CompMon FV, we configure Inspect as follows:

- **Assumptions:** The assumption set consists of all the compliance rules that are implemented in CompMon. Even though many of these would be considered assertions by end-users trying to prove compliance of their IP, they are all considered assumptions here, for the purpose of verification of the monitor. This is because want to show that if every compliance property we supply is true, then validity of expected secondary assertions and coverage points (see below) will be guaranteed.
- **Assertions:** The assertion set for compliance monitor FV consists of properties that are not stated as explicit rules, but that are expected to always be true on an interface that obeys the IO fabric specification. As an example, "the fabric must be IDLE when the agent transitions into IDLE" is not explicitly stated in the specification, but is implied by the set of rules that describe when an agent may become IDLE. These properties are logically redundant with respect to the specification, so not necessary to include there as explicit rules, but are useful for proving that the compliance monitor is enforcing the conditions we intend.
- **Cover Points**: In the cover points set, we place examples of expected protocol behavior. These include reaching every defined state, carrying out each legal type of transaction, and mimicking every waveform given as illustration in the specification document. We need to ensure that the compliance properties, when assumed to be true, will not rule out expected behaviors.

The CompMon FV process is illustrated in Figure 6. When CompMon FV fails, each failing assertion presents a completeness issue, and each failing cover point is a consistency issue. They all need to be fixed. With a failing assertion, we look at the Inspect-produced waveform for it and work to find a too-weak (or non-implemented) rule that is responsible for the failure. We then strengthen the rule, or add it if it was not there at all. With a failing cover point, we need to determine which single rule or a small set of rules contradicts the cover point. We then weaken that rule or set of rules accordingly.

When an Inspect FV run on CompMon passes without red flags, we are guaranteed that every coverpoint is reachable by some legal simulation and that no legal simulation will violate any of the declared assertions.

The CompMon FV effort helped not only in the identification and fixing of several bugs in the monitor implementation, but it also led to the discovery of several ambiguously stated clauses in the specification. As a result, some language was tightened or design-specific parameters were introduced into CompMon to capture the intended behaviors. To this point in the CompMon project, 19 such improvements have been made to the specification in order to improve its completeness, correctness, and consistency.

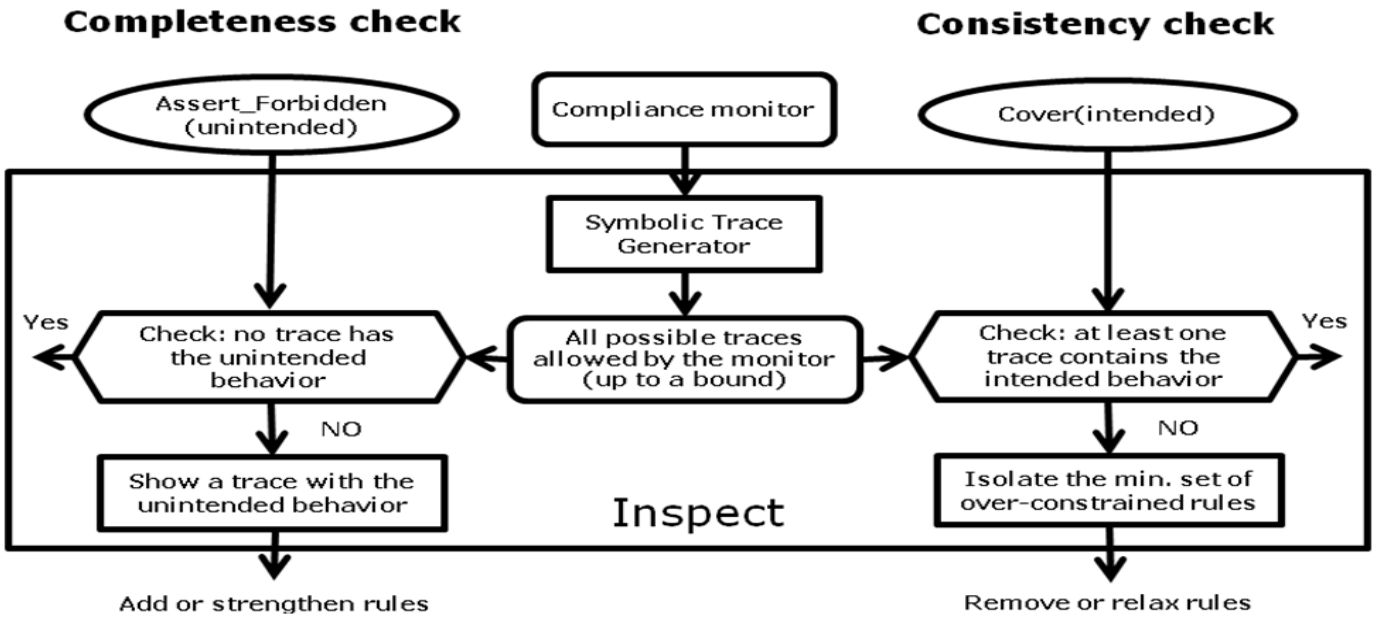

Figure 6: Formally Verifying The Compliance Monitor

6. Compliance Monitor Usage

Recall that CompMon implements all the compliance rules that can be checked on a single interface and that the core of CompMon consists of properties that have been partitioned into *agent rules* and *fabric rules*. Monitoring a single interface can cover over 90% of the compliance rules that are not design rules. The remaining 10% of the non-design rules require information beyond what is visible on the interface wires. To validate such rules, a combination of dedicated test scenarios and internal white-box assertions are required. The other common categories of rules that are not checkable by CompMon are multi-interface rules and system rules. Implicitly, these rules require visibility across multiple interfaces.

For the rules that are checkable by CompMon, using CompMon to check for interface compliance is as simple as instantiating a SystemVerilog module. This will allow CompMon to monitor the interface wires for violations of the agent and fabric rules that it implements. In the previous section we showed how these rules, treated as assumptions in the Inspect tool, allow formal verification of the monitor and the specification to which it is tightly linked. However, one of the key goals of the compliance project is to allow integration of CompMon into the formal verification and dynamic simulation environments of any complaint design. The following text describes the way in which CompMon can be used to enable compliance checking in either dynamic or formal verification.

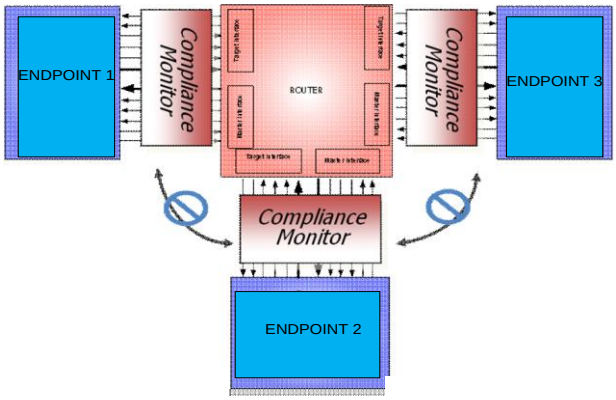

Figure 7: Compliance checking.

Dynamic Simulation: Consider the system in Figure 7, with three agents connected to a router via sideband links. In this scenario, the router and the endpoints might be real RTL or BFMs and an instance of CompMon is instantiated on each interface in the design. During simulation runs, the three instances of CompMon observe all the interface wires on their specific interface and report compliance violations of any rules on their interface. The agent and the fabric rules are both treated as assertions in this usage mode of CompMon.

Endpoint Formal Verification: When formally verifying an endpoint, we need to ensure that if the fabric is obeying the protocol, the endpoint will correctly respond. Thus, properties relating to signals arriving from the fabric need to be assumptions, while properties on signals generated by the endpoint need to be assertions. For endpoint IP FV, the IP is combined with CompMon and given to an FV tool such as Inspect, with the fabric rules given as assumptions and the agent rules given as assertions. Failing assertions would indicate bugs in the IP under test.

Fabric Formal Verification: To verify a fabric, we need to add an instance of CompMon at each of the fabric's interfaces, treating properties on signals arriving from the endpoints as

assumptions, and properties on the fabric signals as assertions. Referring to Figure 7, the design given to an FV tool would consist of the fabric and the three monitors. The agent rules in the monitors would be given to the tool as assumptions, the fabric rules as assertions. This situation is the dual of the endpoint formal verification case above.

Configuring CompMon for flexible FV use: The SystemVerilog language requires that each property be designated as an assertion or assumption in the code. This created a problem for us because, as described in Section 5, we needed the flexibility to enable four different modes of using CompMon, where agent rules and fabric rules can be independently treated as assumptions (conditions on the external environment, assumed to be true), or assertions (properties of the device under test that we want to prove). We solved the problem by adding to each module parameters of the form FABRIC_IS_DUT, ENDPOINT_IS_DUT, and CHECKER_IS_DUT (the latter to support CompMon FV runs as described earlier.) We then stated all properties in CompMon using a wrapper macro, with an extra argument indicating which module it was checking. The combination of parameters and macros allows us to flexibly pass in top-level parameters, and generate the desired formal configuration.

The use of this configurable FV environment was pioneered on various early models. We successfully demonstrated the viability of these FV configurations, running initial proof-of-concept verification to bound 25. This work is still ongoing.

7. Deployment, Results, and Impact

Since its creation, CompMon has seen deployment on seven different Intel design projects, spanning both Soft IP providers and SoC CPU designs.

The deployment to the first major SoC CPU project was divided into three phases in order to minimize environment downtime caused by assertion firings that prevented designers from performing code turn-ins due to failed tests. Minimal downtime was critical due to fact that the deployment happened very close to RTL freeze. The first phase of the deployment was to deploy CompMon to the Primary and Sideband BFMs delivered independently to the project. Because of the deployment of CompMon to the BFMs, 18 issues were found in a combination of the BFMs and their tests. The second phase of the deployment was to integrate the compliance monitor into a private testbench model, to mitigate risk. The third phase of the deployment was to permanently deploy the compliance monitor into the testbench so that all regressions utilized the compliance monitor. A stretch goal was to deploy CompMon into other cluster testbenches. We successfully completed all three phases and the stretch goal.

By inserting CompMon into the various simulation environments of this SoC design, a total of 15 compliance violations were discovered. While many of these cases were waived as an acceptable risk, a few were fixed including a critical showstopper credit management bug found in their main fabric. While most of the violations have not been fixed, their discovery has enabled us to explicitly document the project's compliance exceptions. This is very important as future Intel projects that derive from it will need to be aware of compliance violations so that they can make informed decisions on what is required to be fixed in order to integrate IP blocks from other providers around Intel.

In parallel with this initial deployment, CompMon was deployed to private models of another early project's validation environments. This led to improvements in CompMon's robustness as this project's implementation used features which the other implementation did not exercise. We successfully caught known non-compliance points for the IP.

All implementations claiming compliance with our protocols are now required to include CompMon as part of their test environment and to document how they check for the few rules that require IP-specific information. A design is considered compliant if it passes IP-specific functional testing without violating any of the rules. While waivers might be granted, some of the assertions will require coverage to ensure interoperability. If an IP exits functional testing without exercising these assertions, a hole in functional testing is revealed that must be filled by additional testing.

8. Summary

In this paper, we described the compliance monitor that has been created by our working group to enable testing of compliance with a reusable I/O fabric specification. In order that the same consistent reference is used for compliance testing of any IP block, CompMon is a standalone SystemVerilog with Assertions module that can be used by any design team. By basing the module on SVA, we enable standard simulation, formal verification, and arbitrary future tools to be used in an overall validation strategy which provides high confidence in compliance. We showed how we solve the problem of specification and specification rule linkage in a pre-1.0 evolving specification, and we showed how we used formal verification to achieve a high level of assurance that the compliance rules and the specification are correct, complete, and consistent.

The success of our approach and the compliance monitor we produced have been demonstrated by our co-validation with simulation collateral, our formal verification checks for monitor self-consistency, and the use of the monitor in production simulations. To date, the CompMon project has resulted in the discovery of 20 specification issues, 20 collateral issues, and 15 compliance violations in an initial SoC CPU project, as well as countless bugs in the later IP blocks and designs that began using CompMon near the beginning of their development.

Finally, the rigorous compliance methodology we developed for our IO fabric is applicable to any communication interface specification. Furthermore, we strongly believe that by capturing and verifying an interface specification formally and validating that every IP is compliant to the verified specification, we can help accelerate SoC development and validation in a truly modular and reusable fashion.

9. Acknowledgements

We would like to acknowledge Dave Bertinetti, Sridhar Lakshmanamurthy, Balaji Venkataraman, Pranabesh Dash, Andre Oliver, Kam Zamani, Shweta Shah, Tony Jacobs, and Jaya Chaganti.

10. References

[1] SystemVerilog Org page: http://www.systemverilog.org/

[2] A. Isles, J. Sonander, M. Turpin, "AMBA Compliance Checking Using Static Functional Verification," DesignCon, 2005

[3] OCP web page: http://www.ocpip.org/

[4] AMBA Open Specifications available at arm.com website: http://www.arm.com/products/system-ip/amba/amba-open-specifications.php

[5] H. Foster, L. Loh, B. Rabii, and V. Singal, "Guidelines for Creating a Formal Verification Testplan", Design & Verification Conference (DVCon) 2006, February 2006.

[6] Erik Seligman, Rami Naqib, Ram Koganti, and Kapilan Maheswaran, "Bringing Formal Property Verification to an ASIC Design", Design & Verification Conference (DVCon) 2006, February 2006.

[7] C. Edwards, "Make Less Work For Yourself". Engineering and Technology, Vol. 5 Issue 4, March 2010.

[8] A. Datta and V. Singhal, "Formal Verification of a Public Domain DDR2 Controller Design", 21st International Conference on VLSI Design, January 2008.

[9] Andrea Fedeli, Matteo Moriotti, Umberto Rossi, and Franco Toto, "Addressing IP Reuse With Formal Verification and Assertion Based Verification", Design and Reuse, February 2004, http://www.design-reuse.com/articles/9511/addressing-ip-reuse-with-formal-verification-and-assertion-based-verification.html