Originally posted at EETimes.com on Feb. 12, 2007 at:
http://www.eetimes.com/showArticle.jhtml?articleID=197005268
and reprinted with permission of EE Times.

Achieving completeness in IP functional verification

Wolfram Buettner and Michael Siegel, OneSpin Solutions

This article formalizes the concept of best possible verification quality – completeness – and describes a methodology, field-proven on many complex module and intellectual property (IP) designs, that tells you when verification is complete. To demonstrate the results, it summarizes the verification of a protocol processor IP from Infineon.

How do you know when functional verification of IP is complete? Functional verification is complete when the IP's implemented behavior and specified behavior correspond with each other. At this point, the design is error-free.

But how do you know when that is? Is it when every line item in the verification plan is checked off? Is it when 100% of the RTL code is covered? Or is it when last week's verification runs have detected no further errors? And does achieving these "completeness" goals imply that the design is free of functional errors?

No. These "completeness" definitions are relative metrics that track verification progress and "guesstimate" achieved verification quality. Generally, they do not ensure that the design and specification are error-free.

"Completeness" is an absolute concept, not a relative one, so it needs an absolute, and tool-independent, definition: verification is complete when every possible input scenario has been applied to the design under verification (DUV), and every possible output signal has been shown to have its intended value at every point in time.

Does this mean you must adopt a special design-for-verification methodology? Or, does it mean you must re-implement the existing design in your verification environment? And is this definition at all practical in a world of limited resources? After all, a pipelined processor, for example, has trillions of trillions of different circuit states. However, simulation-based verification executes at only a tiny fraction of real-time circuit operation speed. Consequently, simulation mandates a trade-off between verification coverage and verification resources that argues against the very concept of completeness. Moreover, in the case of IP, simulation-based verification does not adequately define the hardware/software environment(s) into which the IP can be easily and reliably integrated – a serious impediment to IP re-use.

Generally speaking, formal verification has similar problems. Single properties are exhaustively proven with respect to all possible input scenarios. However, the properties typically have an implicative structure: for a specific input pattern (for example, a write request is received by a bus arbiter), the property makes statements about the resulting DUV state and output behavior (for example, the arbiter transmits an acknowledgement). Consequently, a single property typically verifies only a fraction of the possible input scenarios and their associated output behaviors. Therefore, a set of properties must be developed to capture the DUV's entire behavior. This leads immediately to the central questions in formal verification: "Have I written

enough properties, or are there gaps in the property set? Is every possible input scenario inspected and is its effect on the states and outputs verified by at least one property?"

Consequently, this concept of "completeness" leads to a new definition of quality for formal verification – the property set must be gap-free. Combined with the exhaustive verification of single properties, a gap-free property set ensures the most rigorous verification possible. Historically, however, no formal verification methodology has been able to address the issues of how to identify gaps in property sets analytically and automatically, and how to provide the necessary diagnosis information to systematically close these gaps by writing additional properties. And none have adequately addressed the integration environment issue. Can these problems be solved?

Yes, but before answering "how?", it's important to address the issue of completeness and its implications in greater depth.

Completeness and its implications

Design verification is really design/specification co-verification. The module or IP design is verified against the specification while, simultaneously, errors and omissions in the specification are (hopefully) detected and corrected.

An implementation error that escapes verification and ends up in the final chip is generally one of three types: unstimulated, overlooked, or falsely accepted errors. An unstimulated error is one in which the input stimuli fail to trigger the error, and propagate its effects to the outputs; an overlooked error is one in which the error is triggered, but there is no monitor or property to observe and flag the erroneous behavior; and a falsely-accepted error is one in which erroneous behavior at the outputs is not detected because both the RTL implementation and the monitors or properties deviate in the same way from the specified behavior. Of course, the specification itself may be incomplete or incorrect, and thus not reflect the originally intended behavior.

Clearly, a methodology for complete verification must detect these errors reliably and consistently. Therefore, verification coverage is a combination of:

- Input scenario coverage: the ability to avoid unstimulated errors;
- Output behavior coverage: the ability to avoid overlooked errors; and
- Specification compliance: the ability to avoid falsely accepted errors.

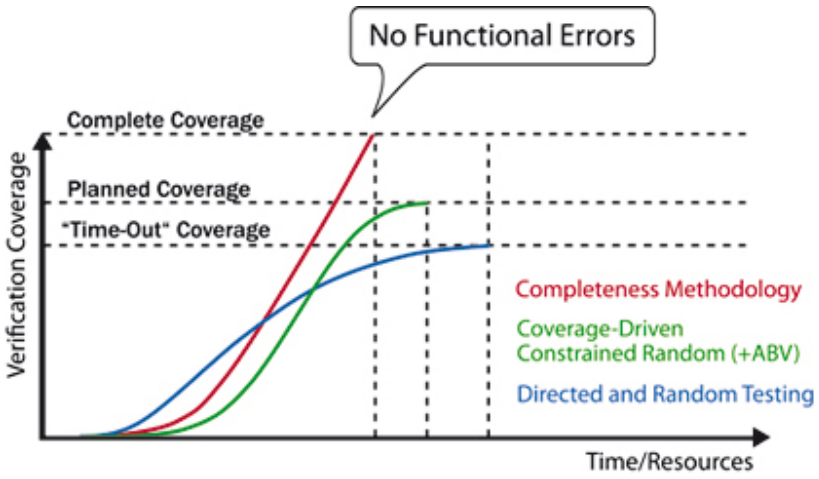
Approaches including testbench reviews, property reviews and executable specifications are used to address specification compliance. Established coverage metrics, and the tools that produce them, address the input and output coverage dimensions. However, they leave much behavior unexplored, and thus fail to achieve completeness.

Inevitably, the question arises in all verification projects: "How much coverage do you have?" Richard Goering of EE Times poses the question another way: "Let's say you get a metric that says you've achieved 98.6 percent verification coverage on a given intellectual property (IP) block. What does that really mean?"

The issue underlying these questions is "What behavior, intended or unintended, have you not covered, and how can you close this gap?" One formal verification methodology exists today that efficiently produces a "complete" -- or "gap-free" -- property set; it achieves 100% input scenario coverage, 100% output behavior coverage, and evaluates specification compliance analytically.

This methodology:

- Requires no design-for-verification methodology
- Requires no recording and analysis of other coverage metrics, such as code coverage, functional coverage, transaction coverage, finite state machine (FSM) coverage, toggle coverage, fault-injection coverage, etc.
- Eliminates the need to anticipate corner case scenarios in advance, because complete verification covers all corner cases, anticipated or not.
- Detects and analyzes previously unknown, and thus uncovered, behavior without time-consuming manual labor.
- Systematically detects specification errors and omissions, because a property error or gap often reveals a specification error or gap.



1. Figure 1 shows how coverage, duration and effort vary for different verification methodologies.

Comprehending well-defined requirements for integration into the target hardware/software environment(s), engineers can integrate the module or IP into the chip. Coverage-driven chip-level simulations then can be performed with confidence that the modules and IP are error-free, thus eliminating the late error detection that threatens tape-out deadlines.

This IP verification completeness corresponds to full equivalence between a property set and an implementation, and results in both a corrected specification and a complete, specification-compliant set of exhaustively proven properties. In other words, completeness achieves true functional sign-off. How is it possible to achieve this verification completeness?

Achieving verification completeness

Completeness is achieved in three steps:

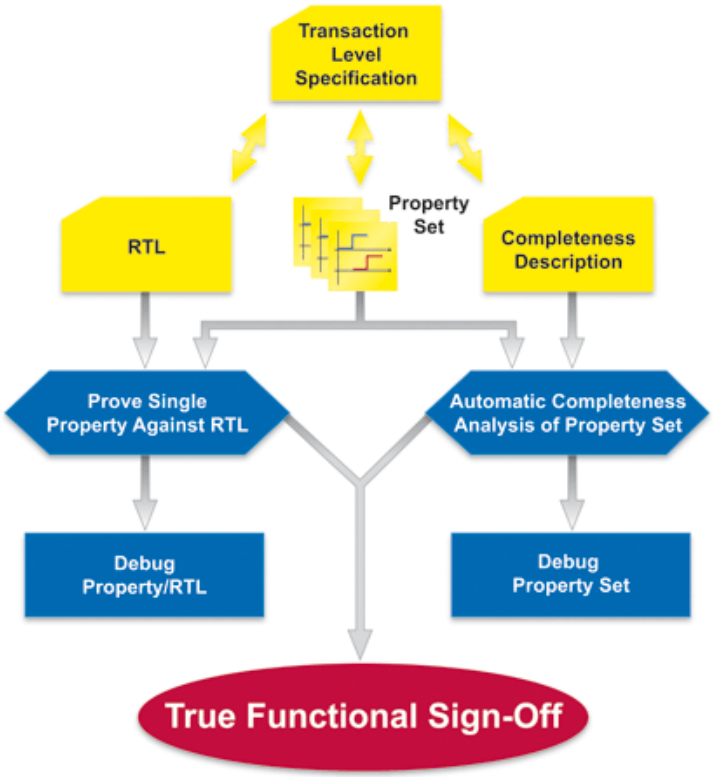
- Verification planning
- Single property development
- Complete, gap-free property set development

Verification Planning

The verification plan defines top-level verification goals that drive partitioning of the verification effort into parallel and serial tasks. However, verification planning is not really a one-step activity. Instead, it is an iterative process using feedback about previously unanticipated behavior from the subsequent verification process. In fact, verification planning is finished and "complete" only when completeness analysis has confirmed that the property set is gap-free.

Single Property Development

Following the verification plan, the designer or verification engineer initially deploys the formal tool's property development and debug environment to describe expected behavior of selected functionality step by step, in terms of transaction level properties (see the left side flow of *Figure 2*). Typically, one property captures one transaction. The tool automatically performs property checking to verify the RTL code with respect to that property.



2. This flow supports the development of a complete, proven property set.

Where mismatches between a property and the code are detected, the debug and diagnosis environment enables the user to quickly locate and eliminate them. Single-property development and debug identifies not only implementation errors but also mismatches caused by specification errors and omissions, allowing the design team to correct the original specification. The output of this flow is a set of exhaustively proven single properties.

Complete Property Set Development

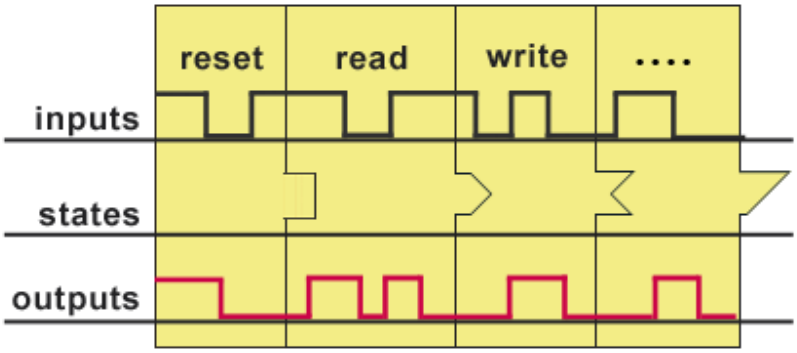
The objective is to develop efficiently a "gap-free" property set that achieves 100% coverage of both input scenarios and their associated output behaviors. An automatic completeness analysis identifies and displays input scenarios and output behaviors not yet covered by properties (the right side flow of Figure 2). These identified verification gaps effectively guide the development of further properties using the single-property development flow. When the final property set is confirmed as complete, it provably captures every possible behavior of the module or IP, and thus avoids faults due to unspecified, unverified or unknown behavior.

Completeness analysis

A single property expresses and captures a single transaction of the DUV, which is a transition between high-level design states. This transaction view of the design is a highly abstracted view, free of implementation detail, that can be easily compared against the natural language specification. It is also highly re-usable in other, similar projects.

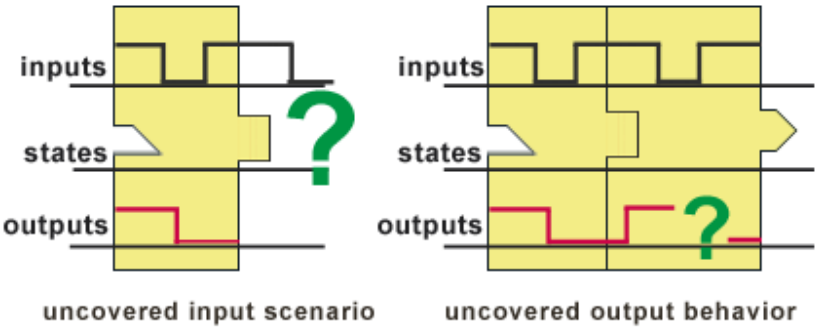
The transaction view is mapped to a specific implementation via a set of mapping functions. The same mapping functions are used in all transaction-level properties, and are easily readable and more compact than the RTL itself. For instance, a transaction-level property can express the expected register-level behavior of an ADD instruction in a processor design in a few lines. The mapping functions that describe all data forwarding consist of a few hundred lines of code. The underlying RTL code comprises several thousand lines. The transaction-level view thus affords the designer a high-level functional view that considerably simplifies verification.

While single properties describe the effect of single transactions, a chain of properties captures a sequence of transactions. An automatic completeness analysis then determines whether every possible input scenario, corresponding to a transaction sequence, can be covered by a chain of properties that predicts the value of all states and outputs at every point in time. An example of such a property chain is shown in *Figure 3*.



3. A property chain predicts the value of all states and outputs at every point in time.

Figure 4 shows examples of the type of property set gaps that completeness analysis detects. In the first example, the analysis identifies an input scenario that is only partially covered by a property chain. The expected matching property at the end of the chain is missing, so a portion of the input scenario clearly has not been covered. This gap could allow erroneous behavior to escape detection.



4. Figure 4 shows two types of gaps detected by completeness analysis.

The second example shows an input scenario for which a full property chain exists, but the properties do not determine a unique value for a given output(s) at some given point in time. In this case, the gap could allow erroneous output behavior to be overlooked and thus escape detection.

In both examples, the design may well behave as intended, but completeness analysis shows that the property set does not capture all possible input scenarios with their associated output behaviors. Consequently, the property set cannot exclude unintended behavior. Therefore, further properties must be developed to fill the gaps.

What about falsely-accepted errors that result when the RTL implementation and properties deviate in the same way from the specified behavior? Using independently developed transaction-level properties minimizes their probability of escape. These properties are simple and concise, especially compared to exceedingly large testbenches, making them easy to review against the specification.

Completeness analysis effectiveness

Infineon has applied completeness analysis to the verification of an IP: its PPv2 protocol processor. This compact, high-performance, configurable 32-bit RISC processor has 40 application-specific instructions, a seven-stage pipeline, and fine-grained multi-threading.

Using OneSpin's 360 Module Verifier, which utilizes completeness analysis, the verification team devised a complete set of only 40 properties that achieved error-free functional operation of the entire PPv2 across all possible configurations. This methodology:

- Ensured the correct pipelined processing of multiple instructions, guaranteeing no undesired interferences between instructions; and it ensured the correct operation of unpredictable behaviors such as traps and interrupts.
- Guaranteed transparency of pipelining together with related forwarding and stalling behavior which is essential for the software programmer.
- Comprehensively verified data paths with complex bit-manipulations.

- Met hard real-time requirements such as trap- and interrupt-execution within permissible latencies.
- Ensured independent execution of multiple threads under all possible combinations of instructions, thread switches and expected, but unpredictable, behaviors.
- Ensured that IP operated error-free in hardware/software environments that met three well-defined customer integration requirements.

Verification found 10 serious errors and 17 other issues that required specification modification and redesign of the context switch logic. One error detected was that instruction words had been modified while stored in the context switch buffers. Initially, this situation had not been considered in the specification and, consequently, had not been covered in the verification. The completeness analysis of the property set revealed that the behavior of instruction words in context switch buffers had not yet been verified, a clear verification gap. An additional property that described the intended behavior revealed the error, and both the RTL and the specification were corrected. This is one example of an unanticipated, unspecified and unintended behavior that, without a completeness analysis, most likely would have escaped any verification approach.

The project consumed 4 engineer-months of effort: 40% less than that of the simulation-based verification of a previous protocol processor that employed a significantly simpler architecture.

The PPv2 IP exhibited no functional errors during integration and chip-level verification, and has been integrated into four SoCs.

Summary

Only with automatic completeness analysis and supporting methodology can you know for sure when IP verification is complete. By eliminating IP verification uncertainty, completeness analysis removes a major impediment to reliable design and deployment of "plug and play" IP. Ensuring that IP is error-free, completeness analysis enables a highly effective blend of formal verification and chip-level simulation.

About the Authors:

Dr. Wolfram Buettner is Managing Director and Chief Technology Officer of OneSpin Solutions.
Dr. Michael Siegel is Senior Manager, Product Development of OneSpin Solutions.