

# The IP Quality Revolution

*Michael Keating  
DesignWare IP, Solutions Group, Synopsys Inc.  
keating@synopsys.com*

## Abstract

*The future of IP quality lies in IP developers and IP users working together in order to create environments and methodologies that will maximize IP reliability and performance, while reducing defects.*

## 1. Introduction

Over the last two or three decades, the automobile has undergone a dramatic, almost revolutionary, improvement in quality and reliability. Driven by strong market forces – competitive pressure and consumer demand – car manufacturers have used technical innovation to improve their products greatly. Thirty years ago, cars had points, condensers, and carburetors, needed tune-ups every three months or so, and seemed to break down regularly. Today they have electronic ignitions, fuel injection, dozens of microcontrollers, and can go 100,000 miles without a tune-up. Breakdowns, although they do occasionally occur, are dramatically less frequent.

Driven by the same strong market forces – competition and consumer demand – IP developers are making equivalently significant investments in improving the quality of commercial IP. The result may well be a equivalent revolutionary improvement in the quality of commercial, third-party IP.

The importance of improving IP quality is obvious to anyone who has talked to SoC designers who use IP. SoC designers have been increasingly using third-party IP; they must in order to do the complex SoCs they are designing in an acceptable amount of time. However, they have often found bugs, poor documentation, and inadequate support. To grow the market for commercial IP, and to enable the development of more and more complex SoCs, IP developers must consistently deliver IP that meets customer expectations.

Much has been written on the required deliverables for IP. The Reuse Methodology Manual, OpenMore, and the VSIA Quality working group have all documented what IP providers need to deliver in order to make integration into an SoC as easy as possible. This aspect of IP quality is a reasonably well-understood problem.

Unfortunately, IP customers want more than a com-

plete set of deliverables; they also expect completely correct functionality. This problem is much harder.

IP users would like perfect, defect-free IP, but current experience indicates that there is no interesting (that is, complex) piece of code, either in software or in hardware, that is defect-free. In [1], Poulin quotes studies that indicate that mature code (code that has been used over a long period of time and on several projects) still has on the order of one defect per thousand lines of code. In [2], Jones indicates that even the best software developers using the best methodology and best tools can eliminate only about 99% to 99.9% of bugs. This level of quality is quite good for software, and probably for most hardware designs as well, but it still does not reach the level desired by IP users (and IP developers).

The reason that complex designs have bugs is fairly straightforward. Human beings write the code, and human beings make mistakes. No matter how carefully engineers specify, design, and code IP, they will make mistakes. In the future, formal methods for specifying and verifying code may well add a great deal of mathematical rigor to design, but even the best of mathematicians occasionally make mistakes. (For instance, there were several false, flawed “proofs” of Fermat’s last theorem by major mathematicians before Andrew Wiles finally got it right – and even his first proof had holes in it.)

Verification is similarly imperfect. A reasonably complex design (say, with 100 or more state flops) has a state space that cannot possibly be completely tested. (Even at two billion states per second, covering  $2^{100}$  states would take longer than the life of the universe). So any verification methodology involves trying to find the optimal subset of tests for exposing as many bugs as possible. Inevitably, some bugs will get through.

The next section of this paper describes what appears to be achievable in IP quality today, and how this level of quality can be achieved. It will then describe how IP developers and users can best cope with the fact that IP does contain defects. Finally, it will speculate on what can be achieved in the future.

## 2. Quality by design

IP quality in particular, and design quality in general, starts with good design; the best way to reduce bugs is not

to introduce them in the first place. A clear functional specification and verification plan are essential for creating a quality design. These documents define the quality requirements for the design: the specification defines the required functionality, and the verification plan defines how this functionality will be verified. Ambiguous specifications, including specifications for industry-standard interfaces and on-chip buses, continue to be a significant source of quality problems in IP.

Once a good set of requirements documents is developed, the IP needs to be designed according to best practices. In [3] the authors present a number of best design practices. It is worthwhile to highlight two of the most important ones.

The most important aspect of any design is how it is partitioned. A well-partitioned design results in a set of sub-units that have minimal interactions with each other. Minimizing these interactions is, in fact, one useful definition of an optimal partition. The resulting sub-units can then be designed, coded, and verified (largely) independently. When these sub-units are integrated, the minimal interaction constraint assures that there will be a minimum of subtle interaction bugs, which are often the most difficult to detect. The remaining bugs are most likely to be local to a single sub-unit. In practice, this often makes the bugs easier to find, but more importantly, fixing these bugs is much less likely to cause other bugs in other subunits – the kind of ripple effect that can make debugging and fixing a complex design a very difficult task indeed.

The second most important aspect of any design is its interfaces. Interfaces between modules are where the most challenging design flaws occur. A very common cause of bugs is when two designers design two interacting modules, but the designers have a subtly different understanding of how the interface between them works. One of the best ways to avoid this kind of misunderstanding is to have a standard set of interfaces, rather than creating a unique interface for each design situation. Standardizing on-chip buses is a key step in this direction. The emergence of AMBA and OCP as de facto industry standards has helped a great deal, but even at the sub-unit level, interface design should be viewed as a multiple choice, not essay, question. That is, an IP design team should have a small library of standard interfaces that are used, rather than invented, for each design situation.

### 3. Quality by review

In [2], Jones presents data showing that design reviews and code reviews are far more productive ways of finding bugs than testing. A few days spent carefully reviewing the design can find bugs that would take many days or weeks to find by testing.

Most design teams conduct design reviews, where the architecture, partitioning, and interfaces for a design are reviewed. This process is relatively mature in most teams.

Unfortunately, good code reviews are less common, partly because there is less of a consensus on how best to conduct them. Most software engineering experts agree that a line-by-line walk-through of the code is the best practice, and that this walk-through should be conducted by one to five engineers (in addition to the designer). Some of the best design teams use a buddy system – where the code review is done one-on-one, and the reviewing engineer ensures that they understand every line of code. Designers often find this review tedious and time consuming, but the data strongly indicates that it provides huge value in terms of improved quality.

In the case of RTL, there are automated design-rule checking tools that can check for a variety of coding guidelines and design rules. These are a valuable addition to (but not a substitute for) code reviews.

### 4. Quality by verification

Although good design and careful review are the most effective way of eliminating bugs from a design, the most effort in improving the quality of a design is spent in functional verification.

The technology for implementing functional verification has made great strides in the last few years. A few years ago, directed functional tests were the primary method of verification, and code coverage the key metric for verification completeness. Random testing was, if anything, an add-on late in the verification process.

Today, the state-of-the-art in verification is quite different. The verification environment is designed for constrained random testing, and directed tests are viewed as just a (severely restricted) subset of the random tests.

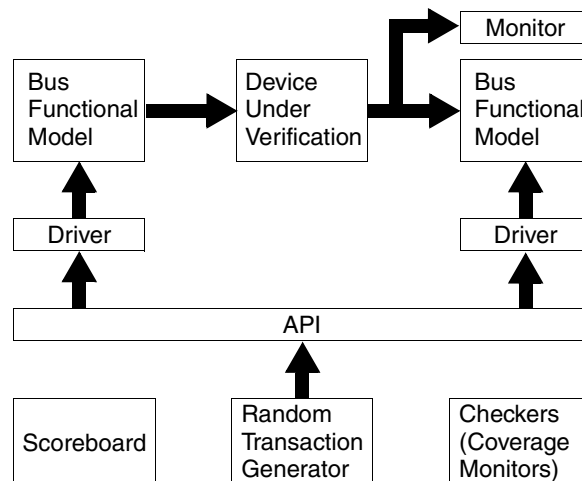


Figure 1. Constrained random test environment

Figure 1 shows a simplified version of a constrained random test environment for a very simple device such as a UART. A random transaction generator delivers a sequence of random stimulus to the drivers through a standard interface. The drivers convert these transactions into

commands for the bus functional models (BFMs). The BFMs convert them to pin-level stimulus to the device under verification.

As the transactions are generated, they are posted in the Scoreboard. As the transaction is completed by the device under verification and received by the appropriate BFM and driver, the transaction is marked as completed in the Scoreboard. Thus, the Scoreboard provides automatic checking of the functionality of the device at the transaction level. The monitor provides automatic checking of the functionality at the protocol level. Thus, this environment is a complete, random, self-checking test environment. For directed tests, the transaction generator is programmed to issue specific transactions, rather than random ones. In this sense, directed tests are just a subset of the random tests.

This test environment is more complex than the standard directed test environment of a few years ago. It is made practical by the emergence of the hardware verification languages that have many of the constructs for random transaction generation built into them. These tools and languages allow flexible and powerful control over the randomization of transactions, allowing tests to be constrained for various probabilities of transactions and sequences of transactions.

The high degree of partitioning in the test environment allows a high level of re-configurability and reuse. For instance, one of the BFMs can be replaced by the RTL for another IP. This may require some modification of the driver, but the rest of the test environment remains unchanged.

The biggest challenge in a test environment is verifying that the tests have completely tested the device under verification. The role of the checkers or coverage monitors is to verify functional coverage. These checkers monitor the transactions at various points in the test environment – at the drivers, at the interfaces to the device under verification, and perhaps even at internal points in the device under verification. A set of required tests and test sequences are defined, and coverage objects determine whether these are executed. The random testing is said to achieve 100% functional coverage when all the functional coverage points are hit.

The actual completeness of the tests depends on the completeness of the checkers, of the functional coverage points. When the specification for the device is ambiguous, developing the functional coverage objects is open to interpretation, and inevitably some errors will occur. However, standards committees are beginning to develop more formal specifications in the form of assertions, or properties, that will allow much more complete and accurate test benches. Improving the rigor and formality of standards specifications is one key area that could greatly help improve the quality of IP.

For configurable IP, there is the added complexity of verifying the IP in the various configurations that will be

used by customers. For many designs, it is not possible to test all possible configurations. In the past, a fixed set of configurations were tested, with the hope that any specific configuration would be similar enough to a tested configuration to be reasonably well verified. More sophisticated teams are now generating random configurations and applying constrained random tests to each configuration. This approach requires a very sophisticated test environment design, but pays off in a more complete verification of configurable cores.

Hardware validation of IP, either in an FPGA or in a test chip, is an important adjunct to simulation-based verification. Simulation is the methodology of choice for verifying functional correctness of IP, because of the ability to control the test environment and transaction sequences. Hardware validation is primarily of benefit in establishing interoperability. For instance, an FPGA prototype of a USB subsystem can be tested with real USB products to assure correct operation.

Finally, there is no substitute for use of IP in actual applications. As a particular piece of IP is used on a variety of SoC designs, the confidence in its functional correctness increases significantly. The additional verification by the SoC design team, as well as the actual usage of the final product in a production chip, gives the final indication of the quality of a design.

## 5. Results from this methodology

Developing a state-of-the-art constrained random test environment requires a substantial investment in engineering resources. We do not have systematic data to measure exactly the improvement in quality achieved by this methodology, but we do have a host of anecdotal evidence. The Synopsys IP team has used such a verification methodology on several designs that had been thoroughly tested by standard techniques, and which had not seen many actual applications in silicon. On each of these designs, the constrained random testing uncovered numerous serious bugs not exposed by previous tests.

For example, the Synopsys IP team re-verified an early (now obsolete) version of the PCIX core. This core had been verified using directed test techniques, had been taped out a number of times, and was considered reasonably mature. Random testing exposed about a dozen defects in the core, two or three of which were serious. One such bug was the following:

- 1) A target has 2 splits pending;
- 2) It goes on to complete one of them;
- 3) If this one is of a size that is  $128*n+1$  – that is, 1 byte beyond ADB,
- 4) and if the completion timed out just before delivering the +1 byte because of a large data transfer ,
- 5) and then the target completes the other split transaction;
- 6) THEN the +1 data (for the first completion) is delivered with the second split completion causing

data corruption.

It is unlikely that any amount of directed tests would have found this error.

Similarly, a number of recently released Synopsys IPs have been through the kind of methodology described above, and they have consistently shown bug rates lower than older IP titles that were not verified with this methodology.

As a result of this experience, all new Synopsys cores are being verified using the kind of constrained random testing described in this paper.

## 6. Future trends

Constrained random verification has added a certain element of rigor to verification. Defining functional coverage points as the metric for verification completeness, the methodology has moved beyond code coverage and toggle coverage, which were clearly inadequate metrics.

The effort to improve the rigor of verification continues. Assertion-based design, as described in [4], looks to define the design goals and verification metrics as an integral part of the design process. Foster and associates advocate developing assertions for macro-level behavior during the specification process, before design begins. Alternatively, for standards-based designs, the standards committee would develop a set of assertions that completely describe the standard (some standards committees are already working on this.)

An often-quoted, very simple example of an assertion is the requirement that a “pop” command is never issued to an empty FIFO, and that a “push” command is never issued to a full FIFO. Such an assertion can be written in Verilog, or in any one of a variety of assertion languages. The following is an example of the underflow assertion written in SystemVerilog (version 3.1):

```
always @ (push or pop or cnt or reset_n)
  if (reset_n)
    if ({push, pop}==2'b01)
      underflow_check: assert (cnt!=0) else
        $error("underflow error at %m");
```

More complex assertions can describe, for example, legal transactions at the input for a core, or certain required behavior of state machines.

The ultimate goal of this methodology is to develop a complete set of assertions for a given design. These assertions would then provide a formal definition for the required behavior of the design, independent of the actual implementation. These assertions can then be used to guide random verification and to provide a metric for completeness of verification.

Some of these assertions can also be formally proved using formal verification tools. For example, in many designs the FIFO assertions mentioned above can be formally verified; that is, the tools can prove that the state machines in the design are such that a specific FIFO will

never be “popped” when empty or “pushed” when full. These tools are still not powerful enough to prove all of the assertions that comprise a complete design, but they can still prove useful.

In addition to the macro-level assertions, Foster and associates recommend developing assertions at the micro level during the coding process. These assertions give the verification tools (and team) detailed and formal descriptions of the implementation goals of the design. These assertions have a similar function to comments in the code, but are much more useful, since they can be used as constraints and/or coverage points for the random verification.

Assertion-based design is a very promising methodology for adding rigor to verification and improving the quality of IP. Foster cites a number of examples of projects that used assertions to improve and accelerate verification.

The Synopsys IP team is currently evaluating how to use assertions to enhance IP verification. In one experiment, a mature, fully verified USB core was re-tested using assertion-based techniques. A set of assertions describing legal inputs and transactions was developed, along with a set of assertions describing required behavior. The Magellan verification tool was then used to generate random transactions, subject only to the constraints defined by the assertions for legal inputs. Other assertions, describing the behavior of the core as defined by the specification, were used to determine whether the core performed correctly. In addition, the formal verification engines in Magellan were used to attempt to prove the assertions about the behavior of the core – that is, to prove that the core behaved correctly and was bug-free.

In this experiment, the tools were able to prove only a few of the assertions, or properties, defining the correct behavior of the core, but the random simulation generated from the constraint assertions was very effective in finding defects in the design. Most of these defects found using assertion-based random verification had to do with the behavior of the core if subjected to inputs and input sequences that the designers never considered, including illegal inputs (thus testing error recovery capability of the core).

## 7. Summary

The progress in improving IP quality, and in improving verification methodology, has been rapid over the last few years. New technology – hardware verification languages, sophisticated bus functional models, constrained random verification, assertion based design – allows much more complete verification than ever before.

Unfortunately, the ultimate goal of defect-free IP is still as elusive as ever. Although verification is improving rapidly, there is no obvious path to intersecting the axis of zero defects. Instead, we seem to be riding an asymptote towards ever-increasing quality, but without ever hitting the axis itself. The state space of interesting designs is still

too large for complete coverage in a finite amount of time, and formal tools are a long way from being able to prove all the important properties of a complex design.

In the meantime, IP users continue to have to deal with the fact that there will be bugs in IP. These bugs are becoming fewer and more obscure as we improve our verification capabilities, but they will continue to exist.

Returning to the car analogy, even the best cars today occasionally break down; not often, but certainly not never. In spite of the massive investments car companies have made to improve the technology and design of cars, this still happens. One of the ways that car companies deal with this is to offer longer warranties, extended service contracts, and roadside assistance. Some of the high-end cars even have a button that the driver can push to summon help when the car does break down.

This also is a key component of the IP vendor-customer relationship. Customer support, immediate reactive support for when problems do occur, is an essential component in IP quality.

As long as engineering continues to be a human endeavor, designs will be imperfect, problems will occur, and fixes will need to be made. The design of complex chips is becoming a partnership between IP providers and SoC designers who use IP. Recognizing that it is a partnership may well be the key to IP quality in the long run. IP providers need to meet their obligations in the partnership by using the most advanced and effective verification methodology to provide the highest quality IP possible. They must go from the equivalent of constant maintenance to requiring tune-ups every 100,000 miles. The technology

is emerging to make this dramatic improvement in IP quality. The investment to apply it will be significant. This cost of quality may, in fact, substantially shape the IP industry in the near and mid-term, but it is essential that we make this investment and improve IP quality to the point that it substantially meets the expectations of the SoC design community.

Once that goal is achieved, the key to the partnership is communication. IP developers need to educate IP users about how a block of IP has been verified and its usage. IP users can then integrate it into their SoC design in a manner that is most likely to work without problems. As IP developers learn more about the intended applications of their IP, and the systems where they will be used, the more they can verify their designs for those specific environments. Finally, as IP developers and users work together over time to address the quality challenges in design, they can drive for more standardization in interfaces, usage models, and verification methodology at the block and SoC level. This cooperative approach is what, in the end, will produce the IP quality revolution.

## 8. References

- [1] Poulin, Jeffrey, "Measuring Software Reuse", Addison-Wesley, January 1997
- [2] Jones, Capers, "Applied Software Measurement: Assuring Productivity and Quality", McGraw Hill Text, August 1996
- [3] Keating, M., Bricaud, P., "Reuse Methodology Manual", Kluwer Academic Publishers, June 2002
- [4] Harry Foster et al., "Assertion Based Design", Kluwer Academic Publishers, May 2003