A practical approach to IP quality inspection

Rich Pell - October 06, 2011

Not everyone likes surprises.

If you are a chip designer working with third-party IP, you have learned that surprises, not always of the good kind, are an inevitable part of the package. And you are not alone – the use and cost associated with third-party IP are on the rise.

So what can you do about it? Do you already have, or plan to have a systematic approach to inspect IP quality on delivery?

Clearly defining what you mean by “quality” can help both you and your supplier converge more quickly on a better flow. Furthermore, your definition of quality probably needs to expand beyond a bug-centric view. A robust process that can automatically assess quality at incoming inspection can have a large impact on your schedule and overall well-being. Instituting such a system may not prevent issues, but it will ensure that issues are trapped quickly, at the source, before they trigger fire-drills later in the design process.

If you are an IP supplier, I’m sure you are already familiar with the concept of “smoke tests” as a quick way to flush out problems in the inner loop of development. This kind of analysis can be used not only to validate correctness but also to give a quick, albeit coarse, assessment of design parameters, as I will explain below. When you are in what-if exploration, this can help you explore more options, more quickly than full implementation analyses would allow.

Why are quality problems unavoidable?

When you decide to use a particular IP, you probably run a comprehensive evaluation and/or you check with other users who have built production silicon around that IP. You may have personal experience and confidence in the suitability and quality of that IP as well. Coming out of this process, you feel pretty confident you have done sufficient due diligence to ensure the IP is solid. Then things start to go wrong. Why?

To explain, let me divide the issue of quality into three main components:

- **Objective quality** - a measure of issues in the deliverable IP according to purely objective criteria. For example, the RTL is lint-clean, the synthesis and timing constraints follow best practices, the IP performs functionally and area and performance are within the bounds of the evaluation reading of the specification. Evaluation should flush out most of these problems but new issues may emerge on the heels of bug fixes or specification changes, and you don’t have time to repeat the full evaluation cycle on each change.

- **Specification quality** - a measure of issues in the deliverable according to a disagreement in a reading of the specification or, more commonly, a measure of what a reasonable user might expect
versus what is actually supported. No matter how carefully you read the specification or conduct the evaluation, you will expect (and fail to test) some corner of behavior which the supplier may not have considered. This is particularly obvious with configurable IP. If an IP has 5 knobs, each with 3 possible settings, the supplier must test over 3,000 configurations over the full range of possible behavior for each configuration. Guess what. They don’t. It wouldn’t be possible.

The same consideration applies to behavioral usage. How far do you push corners of behavior for that hypothetical reasonable user? The supplier tests what they feel should be reasonable use-models. Over time, they evolve their testing as more users push on their definition of reasonable. If the supplier is careful (and lucky), most users can live within a relatively small subset of use-models. But don’t be surprised if you are the first user to push on an unexplored corner.

**Regression quality** – a measure of changes between releases which don’t change the specification but which may impact a specific design – for example if pins have been added or removed, or the IP contains new critical paths. Changes of this nature may be unavoidable but can still negatively impact design productivity. Since it is impossible to model all possible designs and design environments in a regression process, regression issues are common.

Who bears responsibility for a problem really doesn’t matter when a design is dead in the water. Because the customer comes first, you send me a fix. Therein lies the seed of one or more new problems, especially if the fix is triggered by a specification change. As the IP consumer, I need the fix urgently, so you do some basic testing and ship it off. But in the time you had to think about and make the fix, you didn’t consider some subtle consequences. Running through the full regression process is not guaranteed to catch new possible latent quality issues since the existing regression suite was never designed to consider interactions between the spec change and other features.

This is why it is not uncommon to see that a few releases are needed to stabilize support for a spec change. If you multiply this by several IP you are using in your design and more than one possible problem you may encounter per IP, it is easy to see how this could get out of control. You cannot re-run a full evaluation on each change, but neither can you afford to expose your design to incoming changes of unknown quality. You need a scripted system you can run across the entire IP library, checking for what changed and flagging only when an IP changes in some suspicious manner. The cost attributed to IP issues is clearly rising, as shown in Figure 1. A system such as this can help to tame this trend.
What should you check and how should you check it?

Ideally, you should run each incoming release through the full suite of tools that will be used in the design assembly, verification and implementation flow. Such tools include static quality checks, simulation, synthesis, timing analysis and ATPG. IP development organizations / companies do precisely that.

The challenge for a consumer of IP is that setting up a similar kind of checking is hard – very hard. It takes a long time to run (up to a week) and is arguably redundant. (Shouldn’t the IP supplier have done this?) The following line of reasoning inevitably emerges:

- I have to do some level of checking, otherwise I am completely exposed.
- However, no matter how hard I try, some problems will only be detected in design assembly/verification/implementation.
- Therefore I need to balance how hard I try against the delay in getting to use a new release. (This is ultimately a risk assessment - how much delay am I prepared to tolerate versus the potential hit of a problem I might have detected on inspection?)
- I will probably opt for the best quality assessment I can get quickly, especially if the method can be tuned (with experience) to minimize significant escapes.

What is the best quality assessment I can get quickly? Running production tools to screen quality, logical though it may seem, is generally not part of the answer. On the other hand, static quality assessments – lint, domain crossing analysis, power constraint validation, testability metrics and SDC quality checks – can all be run with minimal setup and in short run-time per IP.
Quality is also affected by release-to-release variances - the regression quality component. At the most basic level, if the supplier changes a pin-name, an IP - which may be within spec in all other respects - will still break my design if I started with an earlier version. More subtly, if the implementation gate-count, or latency to external pins, or internal critical paths change significantly from one release to the next, a design schedule that seemed substantially on-track can be sent into a tail-spin. Indicators to this kind of problem can also be provided by static analysis. Leakage and dynamic power can be estimated from RTL to within 15% of silicon measurements for a representative stimulus.

Pre-synthesis (GTECH) gate count can be estimated very quickly, as can logic path depth from I/Os to registers and internal long logic path depths. These are coarse estimates but generally good enough to flag potential major variances. I really don’t care how closely estimated gate count corresponds to implemented area, or if estimated gate count changed by 10%. I do want to know quickly if that gate count changed by 50%, because that will almost certainly translate to a size increase in the design.

What about functionality? Can I really check quickly that the function of an IP wasn’t disrupted in some subtle way? Formal verification may provide some help on small blocks or specific checks, but the setup cost (properties, assertions, etc.), run-time and size limitations may reduce the appeal of this approach in inspection flows. Running the supplier’s testbench seems redundant – presumably they have done this already. Unfortunately, regressing functionality is an area for which no one seems to have found a quick, low-cost solution. Like it or not, you are probably stuck with finding these problems through traditional IP and design verification techniques.

Setting up a practical flow

It is probably no big surprise that this needs to be a push-button batch flow. You don’t want to have to do anything interactive unless a problem is flagged. If you have a lot of IP (or internal blocks) being upgraded asynchronously, you may want to consider using a make command to further optimize inspection runs. Set up correctly, make can also help you manage differing directory/file organizations for IP from different sources, ensure the correct IP release is being checked, and more.

Here’s a list of quick, but fairly comprehensive checks you should include under make or a shell script. First the objective checks:

- **An RTL lint check.** Be careful here. Some well-intentioned lint checks produce a lot of noise – violations will overwhelm the output with noise and hide potentially serious problems. You should work with a solution which has been specifically tuned to report only must-have requirements. It may also be interesting to include some internal functional checks (e.g., deadlock state checks in state-machines). While you shouldn’t necessarily be peering inside the RTL, these can provide ammunition to challenge the supplier’s quality process.
- **An SDC lint check** (both for synthesis and for timing, if supplied) versus the RTL.
- **A clock/reset analysis** (predominantly focused on domain crossings). This will require a little setup in the form of a constraints file.
- **A power intent lint check.** If the IP is supplied with UPF or CPF intent, check that these are consistent with the RTL.
- **A testability analysis check.** Does the IP violate any major design for testability guidelines? Is predicted test coverage within acceptable limits?

Even on a large sub-system, these checks can easily be completed within a few hours.
Now the regression checks. It is important to remember in regression comparisons that I do not necessarily want to compare the latest release with the immediately prior release. I may have last checked out an IP at version 1.3 and now want to upgrade to version 1.5, and the latest version is 1.7. A usable regression script must allow me to specify which versions of which IPs I am currently using and to which versions I want to upgrade. It should be possible to automate building the current list, using native scripting in your source code management system.

- A quick directory difference between releases (use eg diff -q -r). Don’t bother to do a comprehensive bill-of-materials check. Any reasonable IP vendor has already implemented a more comprehensive check than you can. However a quick comparison may be a useful reference if something else looks odd.
- Compare pin-out - pin-names, directions, widths, types (clock, reset, other), parity for clocks and resets, whether immediately registered, whether internally synchronized.
- Compare estimated gate count and rough pin-to-register logic depths.
- Compare internal maximum register-to-register logic depths.
- Compare the number of synchronized domain crossings and the number of unsynchronized domain crossings. If the latter changed, perhaps more configuration signals were added or perhaps someone forgot to synchronize.
- Compare power estimates (leakage and switching) across defined stimulus files.
- Compare test coverage.
- Compare timing constraint coverage - are all nodes constrained in at least one mode?

An easy way to setup these comparisons is to gather all of the above data in a datasheet format, then compare datasheets. Of course you want to run the comparison in such a way that only differences are highlighted, but that should not be particularly challenging for a good scripter. For many of these comparisons, you also want to give some thought to what constitutes a significant difference. For I/O pins, any change is significant. For domain crossings, an increase in the number of unsynchronized crossings would be significant. For area and logic path depths, an increase by perhaps more than 15% would be significant. For test coverage and timing constraint coverage, a decrease would be significant. Figure 2 is an example of a machine-generated datasheet.
Setup correctly, both objective and regression analysis together, even on a complex processor-scale IP, should take no more than a few hours. Under make, checks need only be performed on IP which has been updated. As a result, incoming inspection checks for all the IP on which a given design depends can be reduced to an overnight run – significantly more practical than a week-long analysis based on production tools.

A real example
Theory is great, but what kind of results does this strategy deliver in practice? One large semiconductor company that I know provides a good example. They build complex SoCs, including smartphones and multimedia devices. They use IP from a rich variety of sources – from the standard external suppliers (ARM, Synopsys, Imagination and others), from an internal central IP group (who supply bus fabric, interrupt and security IP) and from other divisions (who internally share successful differentiated subsystems, including video and audio processors, among other IP).

Given such a complex IP environment, this company recognized many years ago that a disciplined process for incoming IP inspection was necessary and already instituted a methodology based on a flow using production tools. This worked well, but as the complexity of the IP and the size of the library increased, they found, even with make, a typical library validation run would take one work week. This became a major problem for design schedules given that IPs were being upgraded frequently.

Recently, this company instituted a process similar to the one I have described. Using Atrenta’s SpyGlass RTL analysis product, qualification time is now around one day, a 5:1 cycle time improvement substantially reducing impact on design schedules. Some IP problems do escape past the qualification process and are ultimately detected in verification or implementation. But here’s the important point: the impact of escapes in the new flow is not noticeably different from the impact of escapes they had when using qualification based on the production tools. In addition, the new flow is substantially easier to maintain since the test fixturing for each class of test is significantly simpler to setup than it was for the production tools. Figure 3 illustrates a high-level
One central IP group has deployed a similar “new flow” process to great advantage. They use the regression part of this flow to make trade-offs and see the results before going through their full qualification flow. An IP lead said that he used to just make educated guesses and run with them instead of taking the time to try a few things and see which works out best. He felt that the estimations from the regression flow were quick enough and close enough that he could achieve better results than with his educated guesses. He still runs the full qualification flow when he has completed his experiments - there’s no getting around that - but it no longer slows down the discovery process.

**Time to rethink your IP quality process?** There are tools in the industry (Atrenta supplies many of them) on which you can build this kind of flow. You are very probably already using these tools. But are you extracting as much value as you can from what you already have? Hopefully, I have shown you how to set up a process which can give you solid incoming IP quality inspection - a process that quite likely finds potential problems you may not have checked before, all with a minimum of overhead both in maintenance and in cycle time.

Or you can continue to grapple with those unpleasant “surprises”, while trying to stick to that design schedule.

For myself, I prefer no surprises.

---

**About the author**

Bernard Murphy PhD is the Chief Technology Officer of Atrenta Inc. Dr. Murphy has over 25 years’ experience in design, sales, marketing, and business development.

Dr. Murphy previously held senior positions with Cadence Design Systems, National Semiconductor.
and Fairchild. He received both a Bachelor of Arts and D.Phil. in Physics from Oxford University.