2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

# Accelerating and Improving FPGA Design Reviews Using Analysis Tools

Anna Tseng, Product Engineer, Mentor, A Siemens Business, Fremont, USA
(anna_tseng@mentor.com)

Kurt Takara, Product Engineer, Mentor, A Siemens Business, Fremont, USA
(kurt_takara@mentor.com)

Abdelouahab Ayari, Application Engineer, Mentor, A Siemens Business, Munich, Germany
(abdelouahab_ayari@mentor.com)

***Abstract*— Design reviews against program requirements and prescribed methodologies must be conducted as part of program risk management designed to ensure execution to schedule and budget requirements, and prevent functional issues in system deployment. This paper presents a process improvement that yields significant review improvement in coverage, effort and duration.**

***Keywords—Design Reviews, FPGA, Lint, CDC, RDC, Risk Management, RTL***

## I. INTRODUCTION

Abstraction often costs precision. Designing in abstraction is necessary in order to accelerate development. Such an abstraction in digital design is a Hardware Description Language (HDL). Common today is a functional level of HDL, the Register Transfer Level (RTL), allowing the designer to specify the function of the design via storage elements (flip-flops, latches and registers) and the logic modifying and transferring values between them without specifying the final implementation details. However, using any higher-level abstraction such as RTL provides the opportunity for unexpected issues to arise later in development or in product deployment. This is because what is built is not the abstraction, but a precise implementation. The abstraction is typically tested more, but the implementation sees service. Programs must manage the risk introduced through such an abstraction in order to minimize schedule and budget overruns, or worse, issues in system deployment.

In addition, designers make mistakes. Verification is performed to ensure that a design works as intended, free from mistakes. This is done through functional verification means, as well as via other means such as Clock Domain Crossing (CDC) and Reset Domain Crossing (RDC) verification that are typically used to cover gaps in functional verification flows. However robust the verification is, however, there is always the risk that an issue will escape, resulting again in schedule or budget overruns, or issues in system deployment.

A reliable development process requires that designs be reviewed for mistakes and errors that can typically arise via abstractions such that the development deterministically results in high-quality robust designs. Removing human error from this review process is key to quality results.

## II. CURRENT SOLUTION: DESIGN REVIEWS

A time-tested tool in minimizing risk is a detailed design review. The effectiveness of the review is dictated by the experience of the reviewers, the amount of time available, familiarity with the design, and completeness of the material being reviewed.

As reliability, data integrity and functional safety requirements become more prevalent in FPGA-based systems, teams that have been ensuring quality via manual design reviews are in transition. These design teams have often been performing manual reviews of the written RTL code, looking for issues known to increase program risk. The review is typically under schedule and budget pressures. The likelihood that issues escape the design review process increase. Manual reviews against a known set of rules is an error-prone and time-consuming task.

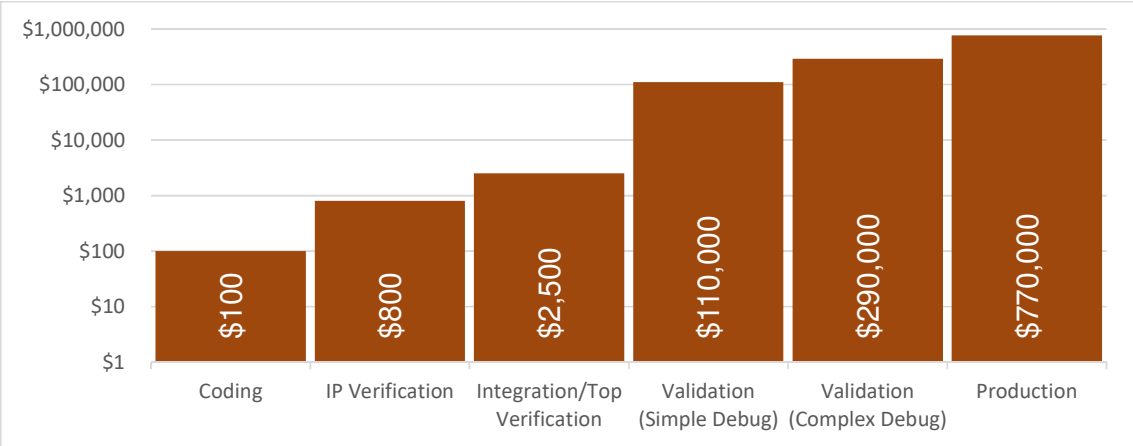


Figure 1 Cost estimates of finding and fixing a bug vs the development stage for FPGAs (log(y)) [1]

In addition, the review checklist of a manual review often works against a team working hard to improve release quality. A review checklist has two problematic properties. The first is that it is a sampling of issues that likely plagued previous products. It's less likely to be focusing on other issues that a new project may encounter that previous projects did not. The second is that checklists for manual reviews often focus more on whether or not something was done, to satisfy a requirement, and not on how it was done. Because of these two effects, manual reviews provide a false sense of security, ensuring that mistakes of the past *might* be caught, rather than whether or not those mistakes, and other mistakes likely to plague new designs *will* be caught.

It should not be a surprise that one solution to the problems of human error, schedule pressures, focus on previous issues, would be to build a review process on top of design analysis tools, maintained by vendors to check for the latest issues, against the latest standards, with a high-performance, low-error, consistent review process. Such an automated approach should yield improvements.

Finally, regardless of the application, the cost of finding a design issue later in development, even if in a design review, increases the later it's found, as seen in Figure 1. One could argue that such data implies value in initiating design review processes early in the design, therefore, when it's being created, rather than waiting for the end. If reviews are time-consuming, expensive and error-prone, building design reviews onto automated analysis tools should enable early evaluations against review criteria.

However, for teams wanting to move to an automated review flow, often hardware availability is a limitation. Tools built to solve the problem in the ASIC market typically run on Linux machines. FPGA teams wanting to automate their review flows may only have Windows-based computers available.

## III. Proposed Solution: Semi-automated Reviews using Design Ananlysis Tools

A program interested in further minimizing risk beyond manual design reviews requires automated, faster analyses of designs built into a defined review process. These tools should be able to do what a traditional manual review will do – but faster and consistently. An example flow of such a review process is illustrated in Figure 2.

### A. *Linting*

The first tool in the automated review flow should be a linting tool. "Linting is the automated checking of your source code for programmatic and stylistic errors. This is done by using a lint tool (otherwise known as a linter). A lint tool is a basic static code analyzer" [2]. In fact, linters appear to have been in use since the mid-1970s in software development.

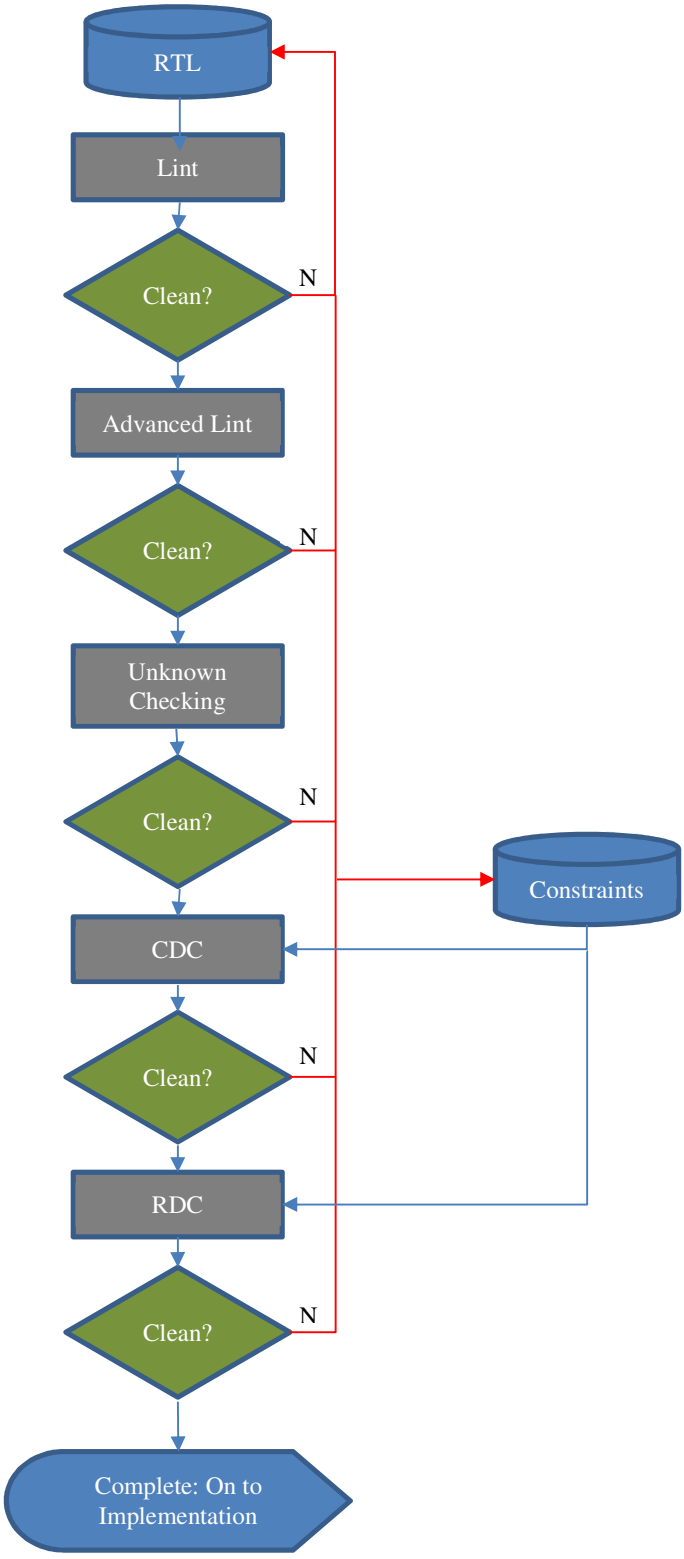


Figure 2 Proposed Semi-automated design review flow for FPGA designs

In the hardware development context, linting is a potential solution to this need for automation. The linting is primarily done on the RTL. In development, the RTL goes through transformations to map into smaller elements implemented in silicon. This implementation, known as logic synthesis, can also contain mapping algorithms to FPGA hardware resources, memories, cell designs and the like. As was mentioned earlier, that abstraction costs precision. This synthesis and mapping process can result in a function different from that specified in the RTL. Base linting focuses therefore on ensuring industry best practices in coding the RTL, as well as known problematic constructs such as ambiguous code, incomplete sensitivity lists, combinational loops, incomplete state machines, overflow conditions, and the like – things for which manual design reviews typically look.

However, linting can be expanded to do much more, such as checking for items that may be particularly sensitive to a specific development process, learned by observing failures late in the development process that could have been caught earlier, as well as stylistic elements or documentation standards required for project archiving. Linting can check for design elements known to be problematic in mapping to specific FPGA vendor products or families.

Finally, FPGAs can also enforce compliance and consistency required by some industry standards. For instance, if an industry standard requires that a specific set of checks be run on the design over the development of the project, and audits will look for these results for consistency, a lint tool can automate that entire process.

Lint tools typically examine the RTL at the syntax, semantic and structural levels of the language. Figure 3 highlights two different structural issues that a linting tool would typically find. The first is that there is ambiguity on the shifting function without the presence of ordering-enforcing parentheses. The designer might intend the result to be 4 but the tools will implement the result as 1. The second issue is a potential overflow problem in the assignment of ‘d’ as the result of an incrementer for ‘a’. These are things quite often found in manual code reviews. A linting tool in this flow will simply be more efficient, thorough, and less error-prone.

### *B. Advanced Linting*

Beyond linting lies a class of analysis tools known as advanced lint. Advanced lint is not typically run on RTL until the basic lint is complete and the RTL is clean. Advanced lint is looking for similar issues, but by leveraging stronger analysis engines beyond the structural analysis in lint, advanced linting can identify other issues, or determine that some issues lint identified are not real. This is typically done by using formal verification engines in a constrained way such that the formal technology is not exposed to the designer, but the exhaustive static analysis is available. A good example of this is again in Figure 3 with the incrementer. A structural analysis identifies that an overflow might occur on the assignment. However, a sequential advanced lint analysis will identify that ‘a’ will never be anything than either 4 or 1, and therefore no overflow on ‘d’ will ever exist. Advanced linting tools are also capable of deeper analysis of state machines, identifying deadlock and livelock conditions, conditional loops, as well as issues such as dead code, which can be a specific area of concern for some standards such as ISO-26262.

Advanced linting tools such as Mentor’s Questa AutoCheck extend the capabilities of the review process beyond what is typically performed in a manual team review. This is because this level of analysis can require a thorough understanding of the state space of the design. For instance, consider the overflow condition previously described. In this case, the signal ‘a’ is effectively a constant. However, if ‘a’ were dynamic, defined in a different module, and depending on many conditions from throughout the design, a deeper analysis is required. Simply put, linting automates most of the checks that are part of a typical manual code scan. Advanced linting digs in deeper where most reviewers don’t have the time to go. Teams assume these issues will get caught in the lab or in simulation – if true, they’re found later when more expensive to fix. If false, the review process let an error escape.

```
wire [3:0]       a, b, c, d;

assign b = 4'h2;
assign c = 4'h1;

assign a = 4'h8 >> b >> c;

assign d = a + 1'b1;
```

Figure 3 Example Verilog code containing ambiguous code that will be reported by linting tools

### C. *Unknown Checking*

Designers creating RTL typically focus on implementing the function required. They do not focus on initialization of the design, often to their detriment. Of course some FPGA designs initialize at power-up but more advanced FPGA-based designs contain many resets and independent logic functions. RTL languages contain the concept of an unknown signal level, typically referred to as 'X'. X's are generated in incomplete initializations, certainly but there are coding styles that can be both X-pessimistic and X-optimistic[3]. X-optimistic and X-pessimistic scenarios are well-documented in papers such as that referenced above and are out of the context of this paper. However, the impact on downstream tools and product schedules should not be ignored. X-hunting can often be a time-consuming task in manual reviews. These require deep knowledge of some of the more obscure elements of the language, such as Verilog's 'casex' and 'casez' constructs.

While linting tools can catch some of these types of simple issues (for instance an uninitialized register) there are tools available such as Mentor's Questa X-Check that accelerate the process of ensuring that the RTL is impervious to X issues. These tools rely on the same type of exhaustive formal engines that advanced linting uses, this time focused on determining whether or not designs will improperly generate or consume unknowns. As with advanced linting, these types of analyses fall less into the category of what's caught in manual design reviews and more as issues teams hope to find in simulation or the lab, with similar results.

### D. *Domain Crossing Checks*

Another tool intended to automate and improve design review processes is focused on finding areas of the design susceptible to random behavior due to asynchronous events. These types of issues are generally understood at a high level, but the number of asynchronous clocks or resets, combined with the addition of large portions of designs being performed elsewhere, make exhaustive and successful manual reviews challenging. These are performed by clock domain crossing (CDC) or reset domain crossing (RDC) tools such as Mentor's Questa CDC and Questa RDC.

Asynchronous relationships between the data or reset, and clock inputs of a storage device such as a flip-flop, result in metastable behavior, which causes the device to retain an indeterminate state for a period of time. Well-known means to design circuits to accommodate these crossings exist and are out of the scope of this paper. However, today's design constraints complicate these crossings and increase the number of them. Beyond traditional crossings, re-convergent data paths or design implementation errors can cause further issues normally missed or impossible to find with manual reviews.

Teams implementing best practices have disciplined clock crossing or reset crossing methodologies, a library of synchronizers of which use is exclusively allowed, and limit the number of signals crossing so that reviews are simple. However, as FPGA designs get more complex, the number of crossings, despite these best practices, quickly exceed the limit of what can realistically be reviewed through manual processes. For any design, a Domain Boundary Crossing (DBC) be it a Clock Domain Crossing or a Reset Domain Crossing must be reviewed. For a design with $n$ domains, the maximum number of DBC ($DBC_{max}$) is defined as in (1)

$$DBC_{max} = (n!)/(n-2)! \quad (1)$$

By this math, a design that has just 4 domains has 16 crossings to verify. When considering issues such as reconvergence on top of the large numbers of DBCs to verify, the opportunity for a manual review to miss a crossing issue is significant.

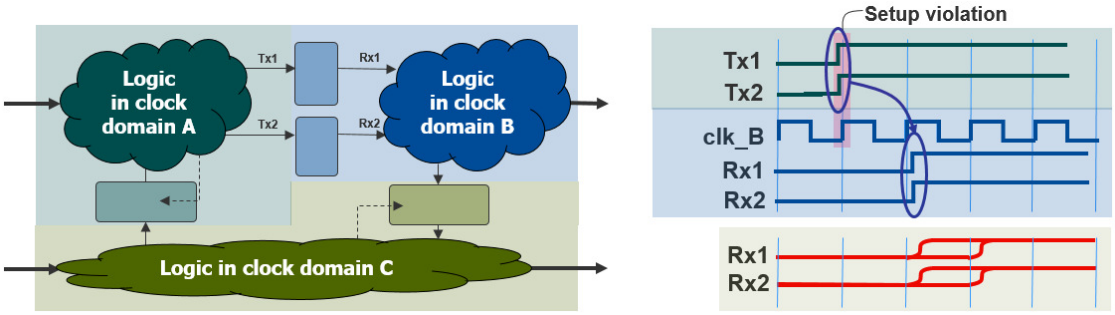


Figure 4 Complex reconvergence

Reconvergence in a domain crossing sense refers to the interaction of logic signals in a new domain that have a defined relationship. If each signal is synchronized, the synchronization implies an uncertainty that results in a race condition in that new domain. Without sufficient safeguards in the clock domain crossing boundary, the relationship may no longer hold resulting in functional errors. A complex reconvergence scenario is outlined in Figure 4 which illustrates that even as the signals may retain their relationship in one receiving domain 'B', the addition of an additional domain 'C' and potential synchronization across all possible boundaries results in a complex analysis. Modern CDC and RDC tools exhaustively verify items such as reconvergence despite the number of DBCs, taking the review process beyond what a manual reviewer can do, despite their expertise, in the time typically allowed.

Reset Domain Crossings deserve special mention in an FPGA flow. Not all FPGAs are susceptible to RDC issues. FPGAs that are entirely initialized at power up and do not implement asynchronous resets during functional operation could be immune from RDC issues. However, with the increase in complexity in FPGA-based silicon, including dedicated IP macros, many FPGA designs today resemble large ASIC-based SoCs. In these scenarios, RDC verification is an important addition to a semi-automated tool-based review flow.

While a full discussion of RDC analysis is out of the scope of this paper, an example of a simple RDC issue is shown in Figure 5. The assertion of an upstream asynchronous reset in a synchronous flip-flop pair across a reset DBC could result in metastable behavior downstream. This type of issue, should it exist in the design, is also subject to the same math as the reconvergence example above. While the circuit in Figure 5 is synchronous and will not show up in any CDC analysis, it is nonetheless an issue that deserves attention.

Finally, referring back to Figure 2, constraints enter the analysis for the first time with CDC and RDC. Unlike with previous analyses looking only at the RTL, domain crossing analysis requires additional constraints to be specified to help the analysis understand the clock and reset relationships in the design. Unlike with the other analyses, a corrective action coming out of the review might be to modify the constraints rather than the RTL. However, there is now additional burden on the review, in that the constraints themselves must be reviewed for accuracy. The analysis is only as good as the constraints. State-of-the-art tools support this review automatically as well and should be used to ensure a complete review.

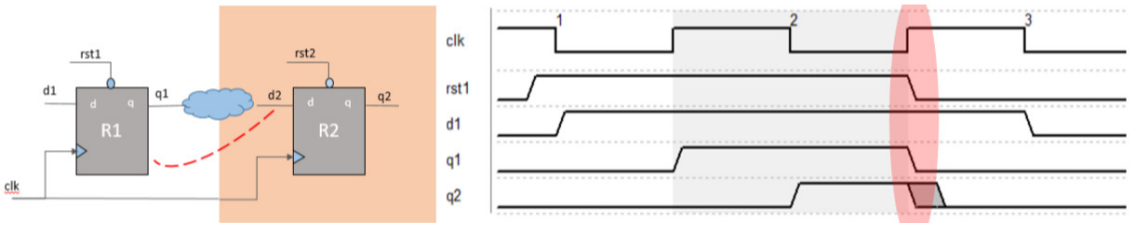


Figure 5 Reset Domain Crossing issue

### *E. Characteristics of a Successful Semi-Automated Review Flow*

In order for the proposed semi-automated tool-based review process seen in Figure 2 to succeed, it (and therefore the tools used, the compute infrastructure, and the like) must exhibit the following characteristics:

- Available: The tools must be available to the designers on their systems of choice or necessity. As was previously mentioned FPGA teams often do not have access to Linux systems yet their demands are similar to the ASIC flows in many cases.

- Accessible: The tools must be usable by the design team during the creation process to ensure conformity to the review requirements prior to the review. As was mentioned earlier, the fastest path through a review, automated or not, is to ensure that the code is being subjected to a review as it's created.

- Configurable: Teams must be able to create and enforce their own rules and checks such that the process provides a custom review per team requirements. Teams specify their own review criteria, and in many cases, their target markets specify standards that must be complied with. This level of configurability is key to an automated flow.

- Identifiable: The flagged areas for review must be able to be identified and easily debugged with suggestions for appropriate resolution. Simple reports of potential problem areas do little toward a semi-automatic review process.

- Repeatable: The process must be able to produce the same results with the same configuration and the same reviewed design. Several standards of interest in the FPGA design community require demonstrable repeatability.

- Auditable: The results of the review must be sufficiently documented in an automated fashion that the results can be archived and audited. The same applications that look for repeatability also look for a time-based record of that repeatability.

## IV. RESULTS

Manual design reviews served the FPGA development community well. However, data in Figure 6 illustrates that 2018 was an inflection point. Logic or Functional Issues and Clocking Issues have consistently been the #1 and #2 cause of FPGA flaws contributing to a production issue since 2012.[4] However, in 2018 a significant drop was seen in Logic or Functional Issues compared to previous years of fairly consistent increases. Designs aren't getting any easier to implement and the FPGAs are getting more complex enabling significant new capabilities – yet functional issues declined. Of course adoption of verification methodologies plays a role here, but increased rigor in design styles, reviews and analysis – the easier aspects of a semi-automatic tool-based review flow – surely played a role as well.

In the same time period, Clocking Issues have been less consistent as a cause, and are unfortunately currently on the rise. This calls for more adoption of the complete review flow proposed in this paper. Doing so should result in a decline across both categories moving forward.

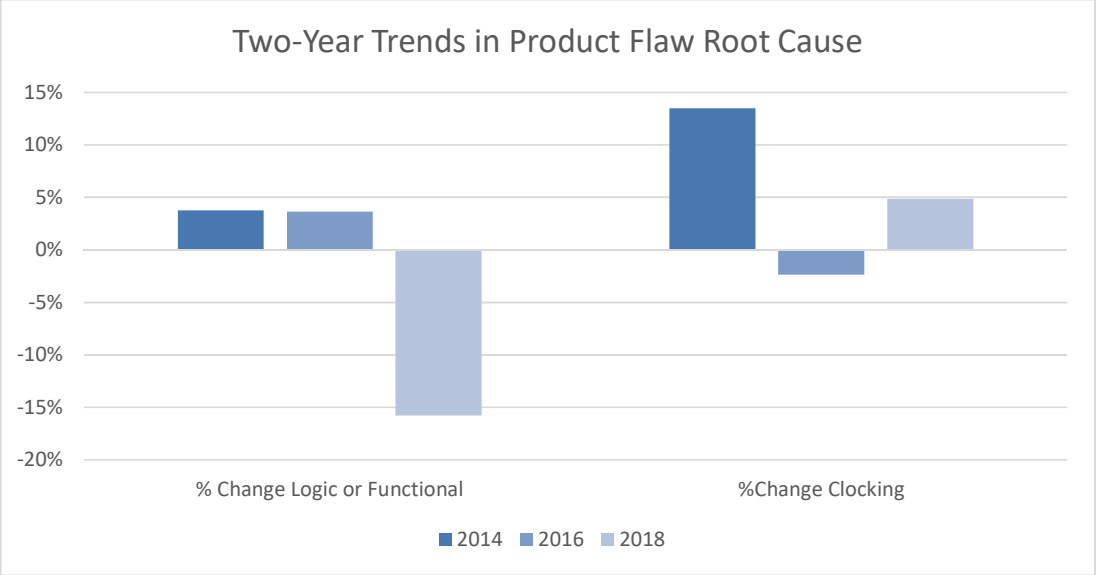


Figure 6 Two-year trends in percent change of types of flaws contributing to a FPGA production issue [4]

## V. SUMMARY

Design reviews are critical to risk mitigation in programs. Effective design reviews must rely more on process and automation and less on the more error-prone human moving forward. In addition, design abstraction, necessary for program schedules, introduces the requirement that industry best-practices and well-known problematic structures be avoided. Finally a good automation flow allows for quick and easy checking for program-wide style and readability as well as audit and archive requirement adherence.

Of course, implementation of these review flows with automation and tools will require additional expense. However, the cost of inaction is high, as designs grow in complexity. The limitation of platform availability for tools appropriate to this task should no longer be a concern.

It bears noting that the issues describe in this paper are not unique to FPGA but apply to ASIC design teams as well.

## REFERENCES

[1] Mentor, a Siemens Business research.

[2] R. Bellairs, "Why is Linting Important? And How To Use Lint Tools", Perforce Website https://www.perforce.com/blog/qac/why-linting-important-and-how-use-lint-tools , March 19, 2019

[3] S. Sutherland, "I'm Still in Love With My X! (but, do I want my X to be an optimist, a pessimist, or eliminated?)", DVCon-2013

[4] Wilson Research Group and Mentor, a Siemens Business, 2018 Functional Verification Study