I created the Verification Gap

Ram Narayan
Hardware Advanced Development
Oracle Labs
Austin, TX 78727
Email: ram.narayan@oracle.com

Tom Symons
Hardware Advanced Development
Oracle Labs
Austin, TX 78727
Email: tom.symons@oracle.com

***Abstract*—It is widely accepted that the productivity gap between design and verification exists and is widening. This paper explores the various factors that contribute to this Verification Gap. While touching on factors like growing design complexity that inherently cause the gap, the focus of the paper is on those factors that are self-induced and hence within our control. Experiences of individuals in the industry have been sought, digested and shared in this paper. The objective of this paper is to evoke a sense of individual responsibility in the reader through self inquiry.**

I. Introduction

If reading this far has caused you to pause and question if you might have created the Verification Gap, then the paper has already achieved its purpose.

A. Celebrating Strides in Design Verification

We have witnessed tremendous growth [1], [2] in the complexity of designs for a few decades now. The fact that we are able to verify such larger designs is a testament to all members of the design and verification community. During this period, we have seen verification solutions approach the problem from various angles such as simulation, formal property checking, hardware assisted verification/emulation and virtual prototyping [3]–[7]. Each of these techniques scale to varying degrees with complexity and abstraction. In addition to these verification engines, we have witnessed a convergence in verification languages and methodologies with SystemVerilog and UVM [8]. This includes means to specify design and verification intent with SystemVerilog assertions and coverage constructs. Verification planning and management techniques [9] have evolved to cope with the increased workloads. A successful verification project today employs more than one technique to achieve schedule and quality goals. A shift in the level of abstraction for design and verification components has presented further opportunity for accelerating design verification.

Since the days of verification being a task performed by the designer to verify the design he/she was creating, verification has evolved into a far more specialized and diverse discipline. It is fair to opine that a design verification job is viewed differently today than it was a couple of decades ago. It is certainly cooler to be a design verification engineer today!

B. The Verification Gap

Our ability to produce larger and larger designs has certainly gone up drastically over the last decade with design reuse. This is particularly true for System On Chip (SoC) designs. The number of cores and IPs being used in SoC is continuing to grow [2]. The complexity for IPs is also growing at a rapid pace. This comes from increasingly complex protocols like AMBA CHI and PCI Express among others. The demand for IPs that can be configured for multiple applications and usage modes drives the design complexity as well. Theoretically, verification complexity grows at an exponential rate compared to design complexity. With Moore's law, that makes it a double exponential rate of growth for verification complexity. Clearly, the trend in complexity can only be expected to cause the Verification Gap to widen.

At this point it would be beneficial to have a consistent definition for the Verification Gap. The most widely accepted definition is the difference between verification productivity and design productivity. In other words, it is the difference in our ability to design versus our ability to verify designs. There are a few ways we could define the Verification Gap.

- The ratio of verification engineers to design engineers.
- The ratio of actual time to complete a project to the time scheduled for it.
- The number of unplanned silicon respins.
- The ratio of the amount of time spent in verification to time spent in design.

Some of these measures are more subjective than others. Further, it is very difficult to obtain necessary proprietary data from organizations to more accurately evaluate the trends in the Verification Gap. Surveys [1], [2] have presented and analyzed these trends across the industry. From these trends, the Verification Gap as measured by the ratio of verification to design engineers is currently growing at a 2X rate. Trends in the number of silicon respins would suggest that the Verification Gap is not growing. Publications and tool vendor presentations have consistently been highlighting not just the existence of this gap, but also that the gap is increasing.

C. Motivation

The seed for this paper was sown during the panel discussion at DVCon 2014 [10]. While easy to point a finger at other contributors to the gap, this seemed like a perfect time for introspection within our community. The purpose of this paper is to sow the seed of self-inquiry in every member of our community - both individually and as an organization. Towards this purpose, we went down the path of interviewing a number of players in the industry. We talked to designers, verification engineers, methodology experts, design managers, verification managers, executives, tool vendors, IP vendors, verification consultants, standardization committee members, silicon validation engineers and software developers and managers. Our sample was not scientific at all, and hence we are not trying to make claims about trends across the industry. We approached each individual (and in some cases - teams) with a set of leading questions.

1) Is there a Verification Gap? If so, is it growing?
2) What is causing this Verification Gap?
3) What have you done to cause or reduce the Verification Gap?

The response from the people we approached was encouraging and enthusiastic. We started with the premise that every person or role in the extended design and verification ecosystem can contribute to the Verification Gap. The feedback certainly substantiated our premise. We are presenting our learning in a form that provides pointers that could benefit you in your self-inquiry process.

D. Paper Organization

Section II attempts to dissect the Verification Gap into various subcomponents. It also classifies it so that we can separate the inherent components of the gap from the self-induced ones. Section III organizes the learning from the investigation process into the various gap components. Section IV summarizes how *I* created the Verification Gap by assuming various roles. Finally, this paper concludes in Section V by sharing some insights from our learning.

II. Dissecting the Verification Gap

In an attempt to understand the cause for the Verification Gap better, we partition the Verification Gap based on the factors that contribute towards it. This will enable us to strategically reduce the gap.

We are dicing the Verification Gap in two ways. We partition the factors that contribute to the gap as Verification Gap Contributors. We also fragment the gap by the amount of control we as individuals have towards its creation. We call this classification the Verification Gap Tractability.

A. Verification Gap Contributors

Based on our learning from the interviews, we arrived at this classification of the factors contributing to the gap.

The Complexity Gap:
This is the gap resulting from the increasing design complexity.

The Skills Gap:
This component relates to the gap between desired engineering skills and reality.

The Schedule Gap:
This component comes from the way projects are planned and scheduled.

The Quality Gap:
This component comes from the way quality goals are defined and executed.

The Metrics Gap:
This is the impact that the pursuit of metrics have on the Verification Gap.

The Reuse Gap:
This is the direct impact of increased design and verification reuse on the Verification Gap.

The Integrity Gap:
This is caused by the misalignment between intent/goal and actions of individuals across the organization.

We discuss these factors to greater depth in Section III along with various experiences we have uncovered through our investigation.

B. Verification Gap Tractability

Another way to classify the gap contributors is by the amount of control we have over the gap. This enables us to tackle the components of the gap that we can control.

Inherent Verification Gap:
: This class of the Verification Gap deals with factors that are just inherent due to the increase in design complexity. Increased complexity in the SoC requires a disproportionately higher investment in verification. The increasing software content in our system adds to the Inherent Gap. This portion of the gap cannot be easily addressed with existing solutions or behavioral changes. This often requires additional infrastructure and processes like software stacks for IPs and cores, standardization and reuse of design and verification intent at the system level. This solution space is out of the scope of this paper.

Transient Verification Gap:
: As the name suggests, this portion of the gap is caused by changes in the design and verification industry. A classic example would be the introduction of new verification methods and languages like SystemVerilog and UVM. While there is a longer term benefit with these methods, it requires a short term investment in developing skills and mastery. This investment involves learning by making "mistakes" , and that introduces a Transient Verification Gap. Methodologies like UVM are not the contributors to the gap. The gap is introduced by the processes in place to design, educate and learn these methodologies. We have a fair amount of control over these factors.

Self-Induced Verification Gap:
: This is the portion of the gap that we bring upon ourselves by not adopting practices that could make it easier to verify designs faster and better. This self-induced gap is entirely under our control. We do not have a measure for the impact of bridging this gap towards the entire Verification Gap. However, steps taken towards these factors will have desirable consequences beyond the Verification Gap.

We were unable to gather data from our interviews to quantify these components of the gap. That level of analysis requires a more in-depth scientific survey.

III. Verification Gap Contributors

Let us take a closer look at each contributing factor to the overall Verification Gap. This grouping of factors emerged from the various interviews we conducted.

A. The Complexity Gap

The impact of increasing design complexity on the Verification Gap is an evident one. This was very clear in our discussions with IP groups. The complexity of IPs is increasing due to more aggressive timing, power, performance and configurability goals. For SoC verification, the increased need for verifying the interplay between HW and SW contributes to the Verification Gap. As mentioned earlier, much of the Complexity Gap is an Inherent Verification Gap and is not of much interest in the context of this discussion.

B. The Skills Gap

Since the advent of RTL synthesis, and to a much lesser extent behavioral synthesis, we have not seen many, if any, advances in the IP design process. The verification world is an entirely different story. A verification engineer today needs to have many new skills, such as:

1) Deep understanding of software design, and how it can so easily go wrong
2) Solid understanding of Object Oriented Programming
3) Deep understanding of the RTL being verified
4) An understanding of how RTL design decisions can impact verification
5) Assertions and functional coverage
6) Random constraints and supporting testbench organization
7) SystemVerilog, but often also C and C++
8) Scripting environments like Perl, TCL, Python, Ruby etc.
9) Verification planning
10) New verification tools and strategies
    a) Formal model checking
    b) Static and dynamic power verification
    c) Graph based verification
    d) Verification class libraries (UVM)

1) Verification Skills: Many of our interviewees listed deficient engineering skills as a significant drag on productivity. Lack of adequate software skills was a common chorus for DV engineers. We also heard the opposite complaint, suggesting that DV methodology was becoming too complex for the average engineer, with the insistence that a simpler environment would be more productive. But that was very much the minority, with most people expressing a desire for more software skills on their DV teams. Some even said they would prefer to have a smaller team with more skilled engineers, as opposed to a larger team with a lower average level of skill. Others were less than polite, describing the field as being afflicted with *"failed hardware designers being asked do to software engineering."* While this is a simplistic and overly harsh statement, it does have a ring of truth to it as the majority of DV engineers today have not received much, if any, advanced or even basic software engineering training, with the vast majority of them holding EE degrees.

This theme was one of the most common we heard and was viewed by most as something that contributed very significantly to the Verification Gap. Across verification engineers of equal experience, we heard estimates of engineering productivity differences ranging from 2X to 10X, with the 10X estimate being the most cited. This is an incredible number. Yet we heard of only minimal efforts to understand this difference. People seemed to assume it was inherent to the individual and only one interviewee spoke of a modest mentoring effort to try to impart some of the skills of his top people to the rest of the team. Another flatly stated that companies simply do not know how to grow engineers into the 10X range, nor do they appear to spend much time attempting to do so. We know that this claim is not far-fetched, as there are many studies that have reported finding this same (10X) result [11]. And we have never heard of any other strategy that can be deployed that can yield anywhere near this kind of productivity improvement, yet it seems very little is being done to understand this significant factor.

2) Designer Skills: This theme was a source of frustration for several respondents. Their belief was that there are many tasks that designers could do that would shorten overall project schedules, but for multiple reasons, designers are unable or unwilling to perform them. Some example tasks that designers could do:

- Not-invented-here (NIH) syndrome. Use an existing, pre-verified component, rather than designing a custom one.
  - Equally significant, re-use an existing interface protocol, or sub-protocol, rather than designing a new one.
- Adequate documentation and code comments
- Adequate use of assertions
- Inclusion of white-box functional coverage
- Thoughtful consideration for verifiable design
- Using an auto-generated component, rather than hand-generated components, such as
  - Register files
- Use of design constructs that improve readability, debug-ability and connect-ability, such as
  - enums, structs, SV interfaces
  - 2001 port declarations
  - arrays instead of hard-coded names
  - functions
  - packages
  - global project constants

Respondents also cited these designer behaviors that slowed the verification process.

- Adding complexity for its own sake
- Adding complexity to save a few gates, a few wires or a clock here or there that has no material effect on cost or performance of the final product. Elegance is king, regardless of the verification cost.
- Resisting adding minor logic that would greatly assist verification

Several respondents indicated that if designers put in white-box coverage, the schedule benefit would be HUGE!. Only one respondent reported ever seeing any designer put in any white-box coverage. The others did not even see comments providing details of what should be covered so it could be implemented later by someone else.

Now, to be fair, more than one of our interviewees suggested that requiring designers do white-box coverage was simply too much to ask. Not only is it a time-consuming skill to master, but designers already have enough on their plates with increasing design complexity, speed, power requirements, and their own schedule pressures. One respondent suggested that maybe a good strategy to help close a Verification Gap would be to hire an extra designer to two, rather than only DV engineers. This way, the design team would have more bandwidth to perform some of the verification supporting tasks that ideally are best handled by designers themselves. Seems like an idea worth considering if we want to shrink this gap.

We also received several comments about design teams that considered any sort of verification work beneath their station. Some designers were even unwilling to launch a simulation or regression on their own. On the flip side, we did hear about a number of teams that worked well together and made many efforts to focus on the project's goals, rather than just their own. However, under schedule pressures, we have ourselves seen teams circle the wagons and blame the other side.

3) Skills Development: A few respondents spoke of a desire to see more focused efforts at skills development for both design and verification engineers. We heard of only limited efforts at any form of code reviews or any other methods to either identify or improve productivity and quality. There were many instances of new methodologies being investigated and adopted, new tools being deployed. However, there was not much effort to improve the engineering skills of individuals beyond basic training for new tools or languages. We heard of nothing targeting advanced verification or software skills. Given the reported productivity gap of up to 10X, it would seem that this would be a great opportunity for significant improvement.

One respondent also spoke about what he saw as very limited attempts at mentoring. We depend more on colleges to prepare employees and make only minimal mentoring, if that, available to them. Yet college preparation for verification is very limited, and virtually non-existent at many schools. So this leaves a huge hole that we are simply hoping the engineers will fill on their own.

We were cited multiple examples of how inadequate knowledge of tools caused significant delays. In many cases, the engineers did not spend adequate time asking the relevant questions. In many cases, the engineers' questions did not get the attention from the solution providers. The effectiveness of EDA tools today is limited not by their features, but rather by lack of knowledge about the very existence of those features, let alone knowledge of how to apply them.

Additionally, no one spoke about deficient management skills affecting productivity or schedules, let alone training plans to improve them. Yet, clearly, failure to more aggressively address the engineering skills of a team is an obvious management failure, given the gains that are perceived by so many of our survey respondents.

4) Innovation Management: Many new and different methodologies are employed to help address the Verification Gap. Often, the difficulties in deploying and adopting these methodologies can cause them to at least temporarily add to the gap. Our respondents reported these difficulties in adoption by both design and verification engineers, with some even suggesting that certain methodologies were overly complex and should be discarded in favor of a kinder, gentler alternative.

To crack the Verification Gap, non-trivial methodologies must be deployed, and how they are deployed and used are as important a part of the overall process as the methodologies themselves. While no one suggested it directly, it did seem as if well thought out deployment practices did not receive the attention they deserved. The targeted engineers generally received more than their fair share of the blame for the failure of the new methodologies to achieve the expected results.

A few respondents described how the deployment of new methodologies fell short of expectations because the primary methodology drivers underestimated the learning curve for the engineers in the organization. In some of these cases, the new methodology was unfairly deemed the cause for the failure. On the flip side, there were examples of successful deployments of new methodologies when the available skills were accounted for in the deployment plan. These efforts realized a significant return on investment even during the early stages of new technology adoption.

C. The Schedule Gap

Many respondents reported that aggressive schedules, particularly aggressive intermediate milestones, forced them to take shortcuts to have any hope of meeting these milestones. However, one underlying contributor of many themes was the lack of time to properly address and solve ongoing issues. The question raised was, *could overly aggressive schedules or milestones be themselves a cause of schedule slips?* It seems a number of our respondents have a strong feeling that there could be some truth to this.

Here are some examples cited by our survey respondents:

- No time for proper architecture definition
- No time for adequate documentation
- No time for micro architecture design documentation. Just start coding.
- No time for testbench architecture documentation Just start coding.

- No time to include good comments, enough comments or even any comments
- No time to insert assertions or white-box coverage up front in the design process
- No time for code reviews
- No time to improve known process flaws
- Too much hacking. Not enough time to do it right.

While some of the lack of time should really be attributed to inadequate skills, a bulk of it is inadequate planning and undisciplined practices. One result of many of these practices is that, without adequate up-front efforts, too much time is spent reverse engineering design details to solve problems as they came up. It is the age old saying, "*Pay me now or pay me later.*" Given the heavy schedule pressures faced by complex chip projects, most tend to the latter, where the cost is always higher.

We all know this. Yet, we still do it. Why? The answer to that is well beyond the scope of this paper. It does, however, seem as if this is a common cause of the Verification Gap.

One of the authors and his son used to race remote control (RC) cars. RC racers have a few great sayings "*fast is slow*" and "*slow is fast*" . We were always told to "*slow down and go faster.*" It was such obviously good advice, but it was hard to follow - there was just too much pressure to try to go faster, right now! It appears that chip design and verification have a similar problem.

It appears that some work is needed here to improve overall scheduling, and to avoid schedules that are self-defeating. One suggestion is for the industry to develop some standard scheduling metrics, from which more realistic schedule estimates could be created. The idea here is to gather real project data to develop reliable correlations between design metrics and design, development and debug time projections. This is similar to hourly estimates for lines-of-code, but something more appropriate for RTL and testbench design. Something concrete is needed to break the on-going conflict between unrealistic schedule expectations vs unnecessary schedule delays.

D. The Quality Gap

1) Architectural Maturity: Almost every single respondent cited ongoing architectural changes, incomplete definition, poor, late or non-existent macro or micro specifications as being a significant cause of extended verification (and software) schedules.

Respondents cited specs with too many features, unnecessary features, excessive complexity and features that were too hard to verify. All reportedly contributed significantly to schedule delays. Architectural changes added late in the design cycle caused even more difficulty, particularly when adequate schedule adjustments were not allowed to accommodate them. One user cited (for an extremely large, complex and high-speed chip) that the excessive architectural thrash and resultant verification thrash they experienced was responsible for adding 50% of time to the verification schedule. Another respondent suggested that we lack good assessment tools that can accurately estimate the real cost and increased verification effort of introducing a new feature late in the design cycle.

Many users stated that this sort of architectural thrash was inevitable, and this was the best that could be done given the complexity of the chips and the difficulty in getting everything correct up front. But others felt that an extended architectural definition period, with significant design and verification feedback would more than make up for its cost in time by removing or at least significantly reducing thrash related schedule delays. This is one example of where excessive schedule pressure or over-eager milestones may themselves cause schedule delays.

2) Environment Quality: This theme was a very common complaint. Most respondents agreed sub-optimal environments needlessly consume significant hours for both design and verification teams.

One common problem cited was *broken models,* where the latest model checked into the central source control repository was broken in some manner, forcing subsequent users to deal with the problems. One interesting point here was that some users reported that overly aggressive solutions to this problem, while preventing broken models, required such significant checkin procedures that the costs of the solution exceeded its gains. This was a common sub-theme - where overall net costs of a solution were not carefully or correctly considered. Of course, this also highlights the common theme that many tasks undertaken here have no easy solutions (but we sometimes pursue them as if they do).

Several users also cited the lack of an efficient edit-compile-run loop, with some citing that a robust capability should be able to improve productivity here by 20%. And given that many DV engineers spend a significant portion of their time in the edit-compile-run loop, a 20% gain here can materially affect the overall schedule. To achieve a "robust capability" here, our respondents listed the following requirements:

1. An efficient "incremental compile and link" capability, where changing a single line of code would only require a small fraction of the compile/elaboration time of a fresh build.

2. An Integrated Development Environment (IDE) that provided rapid code navigation, on-the-fly linting, code completion, templates, quick documentation references, etc.
3. Accelerated reset/startup times to minimize long delays before getting to the relevant cycles.
4. Extensive use of unit-level environments, avoiding the long compile/run times of SoC environments wherever possible.
5. Adequate compute/network performance.

Another problem area discussed was regression failure triage and analysis. We spoke with some DV contractors that said few of their customers did much to accelerate this process, often with no automation whatsoever. Efficient processes here were reported to reduce the time spent finding which failure would be the most fruitful to debug and allowed users to spend more time doing actual debug.

In summary, it seems that many companies have not exploited all the best-known-methods that could improve productivity for engineers in their most common day-to-day work.

E. The Metrics Gap

Many respondents reported frustration with the growing use of metrics, arguing that they had a strong tendency to reward low-value behavior. One example was measurement of assertion counts added by block or by designer, which led to low-value or even negative-value assertions (simulation cost) getting added to the design. Another common complaint was overuse of functional coverage. Many users reported their projects had fallen into the trap of inserting large numbers of trivial coverpoints or especially crosspoints, which then consumed significant time to collect, merge, analyze, report and then finally either close or simply waive. We also heard complaints of limited tool capabilities to deal with large coverage spaces, making the coverage cost much higher than it needed to be. One user estimated that better coverage capabilities could reduce their (extensive) coverage costs by 80%.

With the increased emphasis on coverage driven verification and functional coverage, we have seen an exponential increase the amount of coverage that could be specified in a relatively short amount of time. Many respondents commented that the art of specifying coverage metrics that truly matters is yet to be understood by most, let alone mastered. Many have chosen to rather not pursue coverage metrics than add the unpredictability to the schedule for low probability returns.

F. The Reuse Gap

The broad use of design IP has been a commonly cited source of the Verification Gap. Designers can integrate third-party IP with relative ease, while the verification engineer has a much more daunting task to get it working and to verify that it is properly integrated and compatible with the surrounding system. Protocol standards allow a designer to be quite familiar with the interfaces that must be connected to integrate new IP. But there is no equivalent for integrating the functioning of the same IP into a design. IP knowledge gained during unit integration or unit development was cited as often lost further up the chain. Higher level verification, post-silicon validation and software development often must re-learn this same knowledge. Some form of standardization, in the form of some kind of executable design specification, was cited as sorely lacking by our respondents who deal with this issue. There are some efforts under way toward this end, but our interviewees suggested that they currently have not been able to contain the growth of the Verification Gap as it pertains to IP integration.

G. The Integrity Gap

This theme came from the suggestion by several respondents that verification was doing "too much." The observation was that some tasks were not necessary, but were done because "they might find a bug" , even though most everyone felt the probability of finding a bug was extremely low, often because the same tasks did not find any bugs on previous projects. Several respondents expressed frustration with this issue as it was suggested that politics rather than objective decision making forced this extra work.

1) Fear Driven Verification: Others suggested that it was the duty of a verification engineer to be paranoid and to do whatever possible to mitigate that paranoia. Few projects can tapeout with all tasks completed or an absolute certainty that no bugs will be missed. There must be some deterministic attempt to weed out the low probability tasks to improve schedule and still not expose the project to excessive risk. Drawing this line is not easy, with many engineers being afraid to challenge even obvious overkill, on the fear that they would be criticized should they be wrong and let a bug escape to silicon. If they schedule the task, and get rewarded for completing the task, why should they stick out their neck to improve the overall schedule by suggesting that their time could be better spent doing something else? It seems there needs to be a different kind of reward system that allows engineers to take reasonable risks, and not be burned by it when a low-probability bug escapes. It is a problem akin to buying a warranty service for every item you buy. You know that some items will fail, and the cost to replace those few items will be less that the warranty service cost for all items. So it makes perfect financial sense to buy no warranties, and pay the cost for those few items that do fail. How can we apply this to engineering decisions to avoid tasks that have low overall value and low risk, but not burn the one engineer who identifies a low-value task? This sort of risk taking needs to be applied to the whole project, not separated out to each engineer who makes the individual proposals.

Our interviewees cited the following examples where they felt low value tasks were needlessly pursued:

- Writing core-based tests where BFMs, connectivity checks or other strategies were already done or were more easily done
- Porting a high percentage of tests from pre-silicon to post-silicon
- Porting unit level tests into SOC environments
- Porting unit level golden models into SOC environments

Note that we are not implying that these tasks are always of low value, only that they were considered as such in particular cases by our respondents. This highlights an interesting aspect of verification that came out of our discussions: blanket rules and strategies rarely apply in all cases, but we often try to apply them anyway.

Several respondents viewed this low-value work as a key problem in verification today. Several did not. Getting a consensus on non-trivial issues is pretty much impossible in this business, as with many complex human endeavors.

2) Miscommunication Driven Verification: One additional aspect came up in our survey regarding accurate time estimates. We heard of a number of instances of excessively low cost estimates coupled with overly high estimates of benefits. This seemed to be particularly true for non-standard verification tasks, such as scripting efforts or creating a new BFM or integrating a new tool or developing a new flow, etc. Some portion of this is possibly due to the desire of some individual or group to perform the task in question, thus inflating the net benefit of the project. But whatever the reason, it is important to carefully assign costs and benefits to these side projects as well as core verification tasks.

It was interesting to observe the impact of our opinions on our productivity. We interviewed some designers and some verification engineers in separate sessions. The discussion with the designers was highly productive and very introspective about what they could do to bridge the gap. The verification engineers were debating tradeoffs between verification methodologies and revision control solutions. The need for architecting testbenches *"better"* was visibly overshadowing the primary goals (schedule and quality) without any significant returns. Some interviewees cited how project management channeled the focus of multiple team members in areas of diminishing returns because of unfounded opinions about verification methods. While exposing a likely skills gap, this situation highlights a communication gap within the team which contributes significantly to the Verification Gap.

One final and somewhat surprising aspect we noticed during our survey was that a number of respondents did not have many suggestions about what was causing the Verification Gap or where they could improve on their own verification schedules. We would expect that very few organizations have perfectly mastered the art of verification such that they have no room for improvement. One possibility for this could be simply that they are too busy to step back and review how well they are doing.

IV. Summary

A. I created the Verification Gap

*1) **Verification Engineer:***

As a Verification Engineer, I have created the Verification Gap by

- doing too much - beyond any returns on design quality.
- not collaborating with the designer in the verification and design planning stage.
- not following recommended guidelines for the project and insisting on doing things my way.
- resisting newer methdologies and solutions.
- not making the effort to learn and apply tools and methodologies better.
- not helping my designer become more effective in working with newer testbench methodologies.
- being too opinionated about languages and methodologies instead of just doing my job.
- not communicating with management when asked to work on redundant tasks.

*2) **Design Engineer:***

As a Design Engineer, I have created the Verification Gap by

- resisting using assertions and/or white-box coverage.
- not considering design-for-verification requirements.
- not providing adequate design documentation early in the project.
- not including adequate comments in my code.
- not reusing existing components/interfaces when they were imperfect but good enough.
- not following recommended guidelines for the project and insisting on doing things my way.
- optimizing for speed or size when the benefit was negligible, but the verification costs were real

*3) **Design/Verification Manager/Architect:***

As a Verification Manager, I have created the Verification Gap by

- sacrificing the final schedule for overly-aggressive short-term milestones to please upper management.
- not accounting for the capabilities of each engineer in the project planning.
- not having a plan for continuous skill development.
- asking my engineers to do more beyond returns on quality.
- creating an US vs. THEM environment between our design and verification teams.
- not forging partnerships with solution providers to enhance verification processes and capabilities.
- forgetting that an open-mindset is the most important skill required while hiring engineers.

*4) **Project Director/Executive:***

As an Executive, I have created the Verification Gap by

- not fostering a culture that encourages innovation and risk taking.
- physically separating the design and verification functions across the globe.
- not making my verification leads an integral part of project planning.
- not recognizing the role of software in SoC verification.
- not making necessary tools and resources available to my engineers.
- creating misalignment between goals and actions with ambivalent or ambiguous direction.
- creating a work environment where constructive feedback is not encouraged from engineers.

*5) **EDA Tool Vendor:***

As an EDA solution provider, I have created the Verification Gap by

- procrastinating the standardization process.
- not taking necessary steps to ensure that my customer is using my tools effectively.
- not practicing in my IP design organization what my company is preaching to our customers.

*6) **IP Vendor:***

As an IP vendor, I have created the Verification Gap by

- supplying IP that does not meet the quality expectations of my customer.
- not providing necessary verification IP that will help my customer verify my IP in their context.

B. Gap Control

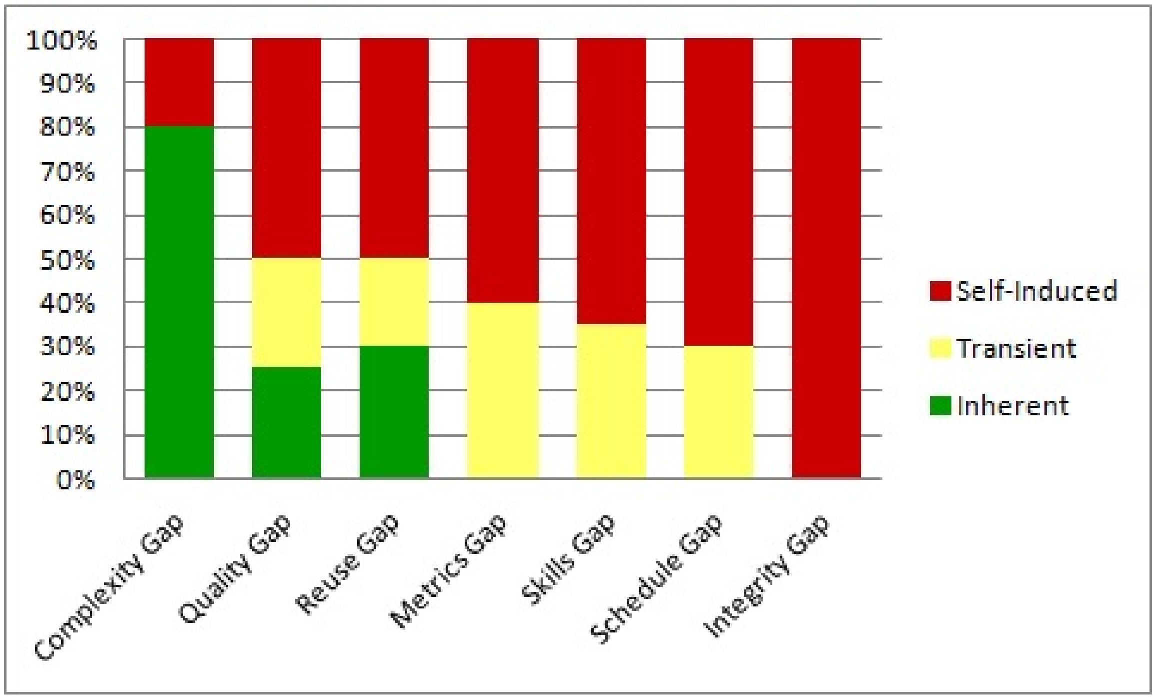

Fig. 1. Verification Gap Tractability estimates by Contributors

Figure 1 shows us the percentage of each gap contributor by tractability. Note that these are just rough estimates based on our survey responses - we do not have any hard data on which this graph is based. The graph is really a pictorial summary of the paper, showing that a large portion of the gap is self-induced or transient, and thus controllable and reducible.

V. Conclusion

A. The Verification Gap : A Perception

There has always has been and always will be some dominant gap in our industry. In the beginning, it was the Design Productivity Gap where our manufacturing capabilities were exceeding our ability to design. The prominent gap now is the Design Verification Gap where our ability to design exceeds our ability to verify. Tomorrow, we might be challenged by a Silicon Reliability Gap [12], [13] with feature sizes getting closer to the wavelength of light. It is already harder to close on design goals like power, performance, timing, functionality and schedule with a traditional serialized approach. Feedback and analysis on all goals are not available until late in the design cycle.

In reality, the gap is merely a perception. It is the difference between the expectation of how long a process should take relative to how long it actually takes. It does take form in publications and vendor presentations. These discussions about the gap are instrumental and necessary in driving innovation and new solutions to continue to meet the challenges of the future. Since chips cannot go from concept to silicon in zero time, there arguably always will be the perception of a gap. Each organization will determine what measures are warranted to mitigate the consequences of the gap. Some organizations are more tolerant to quality gaps, while others are more tolerant to schedule slips. There is no universally applicable solution.

B. Choice and Responsibility

It is very easy for new tools and methodologies to become a distraction or a deterrent rather than a productivity enhancer. It becomes our responsibility to chose wisely and act with integrity - staying true to our goals. Much of the gap is self inflicted due to choices we have made. The most useful question we can ask ourselves often is - *What problem are we trying to solve?*. Answering that question with integrity will eliminate the conflict the new choices bring with them.

As we adopt new solutions, it is important to not think it necessary to relinquish existing solutions. Abandoning techniques we are good at for methods we are not yet good at causes a widening of the gap. With the advent of new solutions, verification and verification management has become more of an art than a science. We are in an exciting phase in design verification, and with that excitement comes responsibility.

C. Next Step

It would be interesting to see some scientifically conducted surveys that would quantify some of the estimates we have provided here for the various gap factors and their tractability. It certainly seems very impractical to gather the necessary detailed data from across the industry. Our recommended next step to you is to shift your attention inwards - as individuals, as teams and as organizations. It is important for you to ask yourself - *Have I created the Verification Gap?* This paper has provided a few pointers to guide your self assessment. As a team and organization you can outline your own gap factors. Some gap factors are more within your control than others. Our recommendation is - Don't Mind the Inherent Gap! It is what it is. Partner with solution providers to improve tools and methodologies and reduce that component in the future. Focus on addressing those factors that are within your control. Individual and organizational integrity, alignment, an open mindset and responsibility will go a long way to ensuring success. Organizations' ability to do this well will give them a competitive edge.

We may have taken *Only the Paranoid Survive* [14] too literally for too long in the verification world. We believe a Strategic Inflection Point has been hit. Verification is not for the paranoid. Verification requires integrity, diligence and an open mindset. Paranoia only causes the Verification Gap.

An immediate next step for the authors is to desensitize ourselves to behaviors in others that we may see as contributing to the gap. We see some of those judgmental tendencies rising within ourselves over the course of working on this paper. Each of us needs to focus on how *I have created the Verification Gap.*

Yes, I have *created* the Verification Gap. But, that is not the point! The important thing is to recognize when I *am creating* the Verification Gap and through that awareness make the necessary adjustment to prevent a widening of the gap.

Acknowledgment

We thank all the people who responded and contributed to this paper by openly sharing their experiences and opinions. This paper would just not have been possible without their inputs. We appreciate our management at Oracle Labs for the encouragement to pursue this interest. Many thanks to Walter Esling for periodically performing PDF to DOC conversions for us, which enabled us to continue to use LaTeX to create this document. Thank you for taking the time to read this and we wish you the best in your endeavors to bridge your Verification Gap.

References

[1] H. Foster, *2012 Wilson Research Group Functional Verification Study*, Verification Horizons Blog, http://go.mentor.com/2viir

[2] H. Foster, *2014 Wilson Research Group Functional Verification Study*, Verification Horizons Blog, http://go.mentor.com/41axi

[3] H. Foster, *Evolving Verification Capabilities*, Verification Academy.

[4] H. Foster, L. Loh, B. Rabii, V. Singhal, *Guidelines for creating a formal verification testplan*, DVCon, 2006.

[5] R. Narayan, *The future of formal model checking is NOW!*, DVCon, 2014.

[6] L. Rizzatti, *Hardware Emulation: A Weapon of Mass Verification*, Electronic Design, October 19, 2014.

[7] C. Daniels, *Use Case of Virtual Prototype Using Vista*, Design Automation Conference, 2014.

[8] M. Glasser, *Open verification methodology cookbook*, Springer, 2009.

[9] P. James, *Verification Planning and Management*, Verification Academy.

[10] *Industry Leaders Panel: Did We Create the Verification Gap*, DVCon, 2014

[11] S. McConnell, *Origins of 10X - How Valid is the Underlying Research?*, 10X Software Development, January 2011.

[12] M. Shafique, S. Garg, J. Henkel, D. Marculescu *The EDA challenges in the dark silicon era: Temperature, Reliability, and Variability Perspectives*, Design Automation Conference, June 2014.

[13] A. Shanmugavel, *Reliability Challenges In 16nm FinFET Design*, Semiconductor Engineering, October 2013.

[14] A. Grove, *Only the Paranoid Survive*, Currency Doubleday, 1996.