On

The Cost of Bugs,

Or

Understanding the total costs of finding, and of not finding, hardware bugs

Bryan Dickman, Valytic Consulting Limited
Joe Convey, Acuerdo Limited

Abstract

Whether you are developing a hardware product or a software product or both, understanding bugs, what causes them, how to avoid them, the cost of finding them and the cost of not finding them, becomes one of the biggest drivers that shape how you develop products. Understanding all aspects of this will help you to reason about the balance between delivered product quality and ROI.

Figure 1: The ROI balance of finding bugs and not finding bugs

1 Introduction

We are writing this article from the perspective of IP hardware product development where bugs drive a large part of the development cost and getting it wrong can have a big impact. How do teams deliver on time, with functionality, performance and integrity and what are the costs involved in ensuring an absence of bugs and the delivery of high-quality products? Conversely, what are the impact costs when bugs are missed and discovered post release, incurring significant rework costs and other hidden costs such as lost opportunities or lost sales? These costs can also be passed down the customer/consumer food chain and eventually impact the end-users. Bugs can drive up development costs and subsequently drive down ROI and customer confidence.

Much has been written about the cost of bugs from a software perspective, but the true cost of hardware bugs is less well documented or understood.

Often the understanding of the impact scope is limited to the engineering rework costs of fixing bugs. This cost is highly dependent on when bugs are found. Finding and fixing bugs is part of the normal development flow for hardware designs, with bugs found later in the development cycle being more disruptive and more costly to fix. Bugs that are discovered after product release are even more costly, and their impact on software and deployed hardware products can be significantly more wide reaching. The true cost of bugs extends to other considerations such as the lost opportunity cost when

resources are diverted away from new product development activities, reputational damage and business impacts and the cost of really understanding the root causes and implementing appropriate improvement measures to prevent future similar bugs.

Everyone knows about Intel's infamous FDIV bug from the 1990's which at the time was probably the most expensive bug in history. In December 1994, Intel recalled the defective processors.

In January 1995, Intel announced "a pre-tax charge of $475 million against earnings, ostensibly the total cost associated with replacement of the flawed processors." [1]

And yet the root cause of this colossally expensive error was a simple engineering error that had been overlooked by the developers.

More recently the security vulnerabilities known as Spectre[2] and Meltdown[3] have emerged. These vulnerabilities were found and published by Google (the Google Project Zero[4] team) in an effort to demonstrate a new class of security vulnerabilities unique to advanced processors that implement speculative execution of out or order instruction streams. Many processors from multiple vendors were afflicted with these vulnerabilities and there has been much written in the media with much rhetoric about this being one of the worst bugs ever found.

So much so, that there is even well recognized branding and iconography for the 2 classes of problem, the Spectre ghost and the Meltdown shield!

Whether it is classified as a bug or just an unfortunate 'feature' of the design doesn't really matter. The fact is that the GPZ team identified a set of vulnerabilities caused by the hardware design, and exploitable by malicious code to compromise these assumed security barriers provided by the hardware. This is a huge deal.

Meltdown was dubbed by Daniel Gruss, one of the researchers that discovered the vulnerability, as "probably one of the worst CPU bugs ever found." [5]

These bugs really made global headline news[6] [7] [8] [9] and are still much talked about today. It is hard to estimate the full cost impact of these bugs and it is likely to have surpassed the Intel FDIV bug.

Product development teams need to be able to reason about the investments they make in workflows and methodologies. Businesses need to be able to reason about the cost of developing new products and the impact cost to the business of getting it wrong. Customers of IP afflicted with critical bugs may suffer even greater financial damages when products reach end users resulting in costly product updates or recalls.

Bugs can have serious financial impacts. Understanding these impacts is fundamental to investment decision making for avoidance and mitigation.

This article follows on from 'On the Origin of Bugs'[10] which discusses 'where bugs come from and strategies to avoid them'. Following on from this we explore the costs involved in finding bugs in terms of engineering investments, and then explore the costs of not finding bugs in terms of the impact costs.

1.1 Understanding Bugs

The Origin of Bugs paper discusses this topic at length from a hardware development perspective in terms of classification and characterization of bug types.

1.2 Living with Bugs

No design can ever be bug-free. Verification is an NP-Hard[11] problem, i.e. there is no perfect solution. No 'silver-bullet'!

All complex designs contain bugs, without exception.

How do you design a complex hardware IP such as a processor and fully verify it? We mean 100% verified,

[1] https://en.wikipedia.org/wiki/Pentium_FDIV_bug

[2] https://spectreattack.com/spectre.pdf

[3] https://meltdownattack.com

[4] https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html

[5] https://www.theguardian.com/technology/2018/jan/04/meltdown-spectre-worst-cpu-bugs-ever-found-affect-computers-intel-processors-security-flaw

[6] https://www.theguardian.com/technology/2018/jan/04/meltdown-spectre-worst-cpu-bugs-ever-found-affect-computers-intel-processors-security-flaw

[7] https://money.cnn.com/2018/01/03/technology/computer-chip-flaw-security/index.html

[8] https://www.ft.com/content/0052e072-f13e-11e7-ac08-07c3086a2625

[9] https://www.bbc.co.uk/news/technology-42561169

[10] https://www.valytic.co.uk/whitepapers

[11] https://en.wikipedia.org/wiki/NP-hardness

whatever that means! Hardware is getting more complex and bugs are getting correspondingly more complex.

Verification is a resource-limited 'quest' to find as many bugs as possible before shipping.

Of course, this leaves the remaining bugs that will likely be present in the shipped products (the unknown-unknowns[12]), and we 'hope' that should they emerge, they are not impactful and/or can be satisfactorily worked around in the field.

1.3 Avoiding Bugs

How can design teams minimize the number of bugs that get coded into the design? As we have already asserted earlier, there is no such thing as a bug-free design, but there are classes of bugs that are extremely hard to find. We can't account for them in test planning, because we don't know what they are. We don't have coverage goals to reassure ourselves that these cases have been both stimulated and checked, because we don't know what they are. We 'hope' that comprehensive random verification environments will eventually flush them out. We can check to ensure that our random constraints do not over-constrain the stimulus and that sufficient[13] 'assurance-cycles' have been run that the code appears to be stable. We can review and re-review everything in the verification environment and brainstorm the question "what else can we do?". We can adopt an approach of continuous improvement. When a bug is found, no matter how it is found, we need to ask the question "why was this not found earlier?". We need to review the testing around this space to see if it can be enhanced to increase the probability of triggering this bug sooner and with higher frequency. We also want to check for the presence of any sibling bugs that may be lurking and use this hindsight to consider if other areas of verification can be improved.

It's not enough to endlessly improve the verification environment (but we are going to do that regardless), and we need to look at how the design can be codified in a way that minimizes bugs in the first place, on the premise that not all bugs can be found.

That is to say, it's as much a design methodology and design practices consideration as a verification concern. The 'Origin of Bugs' paper expands on bug avoidance strategies in more detail.

2 The Cost of finding Bugs

What does it cost to develop IP hardware products that meet the product quality objectives?

How much does it cost to verify a product and find all of the bugs before you release the product?

What investments are necessary and what are the operational costs?

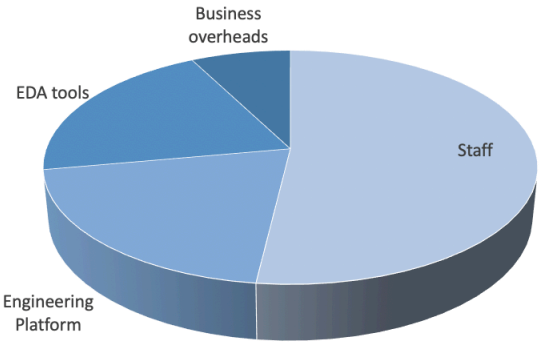

Figure 2: Cost of Business (fake data)[14]

2.1 People Investment

Let's start with the people investment. After all, the people investment is going to be the dominant cost, more than the Engineering Platform and the EDA tools required to perform product development work. And businesses do need to 'invest' in staff, because hiring is a costly activity, and at the end of the day, the success of the products comes down to the quality of the staff; how well trained they are; how experienced they are and how innovative they are; how well equipped they are with the 'best-in-class' tools and resources to do their jobs.

Engineers love to solve challenging problems, innovate, build 'cool' things, be experts and craftsmen, have access to the latest and best-in-class platforms and tools and work with teams of talented colleagues. They also like to get rewarded well, recognized for their achievements, achieve kudos, and engage with the wider industry to keep abreast of latest technologies and developments by participating in industry-wide and academic events from time to time. They have career aspirations and ambitions. They need to feel happy at work.

When these things are persistently not there, their engagement with the work and the business may falter and they may move on, because they are lucky to be in an industry where there is strong demand for their skills and

[12] https://en.wikipedia.org/wiki/There_are_known_knowns

[13] You have to analytically decide what sufficient means!

[14] Please excuse our 3D Pie Chart ;-) http://www.getnerdyhr.com/3d-pie-charts-are-evil/

their talent. That's why HR teams talk about 'talent acquisition' and 'talent development'. Spending too much engineering time dealing with critical bugs on legacy products can diminish engagement if it means missing out on development opportunities and getting stuck in a cycle of never-ending rework, which can be demotivating. That said, there are plenty of innovation opportunities when it comes to complex problem solving around design and verification methodologies. How to build better workflows and use smarter tools that will avoid future critical bugs, avoid costly rework, and enable engineering resources to be deployed towards making more 'cool' products sooner, that make the business even more successful.

It is critically important to hire and retain talent.

Talented engineers want and expect to have the best available tools, platforms and availability to support them in the quest to build great revenue earning products. An interesting challenge for any business developing complex products that consume costly tools and resources is how to educate engineering teams to be cost-conscious and use the available resources sparingly and effectively, and to understand and value the cost of providing those resources. Of course, engineering understand cost, but providing the teams with clear data showing the relationship between the cost of providing the engineering platform and the process of designing IP can create positive perspectives on how to create cost efficiencies;

"Wow, did we really spend that much? Surely we can do something about that by improving xyz"?

How do you encourage innovations in methodologies and deployment of tools, at a cost profile that fits the company's business plans and provides engineering best in class facilities?

Creating this commercial awareness in engineering teams is incredibly worthwhile as it makes partnership with other parts of the business responsible for funding and buying the resources much easier.

Having engineering as a willing partner also makes a difference to the success of tool evaluations and subsequent negotiations with vendors. Creating a culture of partnership that transcends internal barriers but also extends to the company's EDA and IT suppliers is really valuable, as it encourages vendors to become "partners" working with engineering in a joint effort to confront the technical challenges involved in a process like IP verification and debug.

2.2 Engineering Platform Investment

The Engineering Platform refers to the entire stack of infrastructure, applications, environments, tools and automation layers that a business may provision and operate for their engineering teams. The platform is the factory that enables product development teams to do their work and develop products. It is one of the two biggest product development cost overheads (the first being people of course) and is a massive investment for developers of complex IP products such as Processors, GPUs, SoCs and other semiconductor componentry.

The Engineering Platform may comprise a huge investment in either on-premises (let's use the 'on-prem' abbreviation) or off-prem (cloud) compute and storage, specialist hardware acceleration technologies such as emulators or FPGA farms, and often a similarly huge investment in Electronic Design Automation (EDA) technologies that consume this compute capacity. We will characterize the Engineering Platform as a hierarchy or stack of capabilities that as a whole deliver the necessary production engineering capabilities that the product development teams consume.

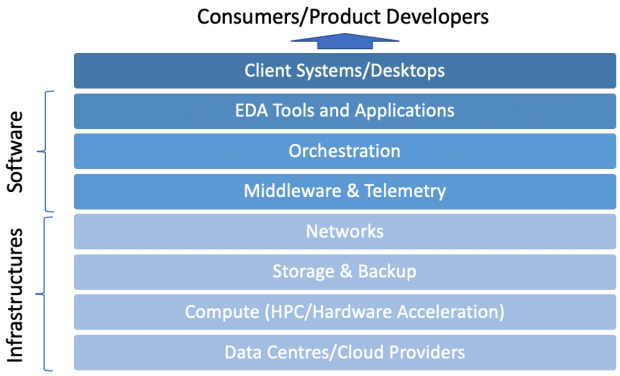

Figure 3: The Engineering Platform

This compute capability is required to crunch through vast volumes of product development work.

Typically, >80% of this compute resource is consumed by verification activities such as Verilog simulation/emulation.

Over the last two decades the industry has seen the compute requirement for Verilog simulation grow by several orders of magnitude and the compute infrastructures have scaled up accordingly. For complex IP products such as processors, most of this simulation consumption is taken up by constrained-random verification strategies where the ability to consume cycles is open-ended. Product teams struggle to define meaningful targets for constrained-random soak testing and will therefore tend to consume whatever compute resources and tool licenses are available to them. More cycles means more confidence, and this philosophy is reinforced every time a late bug is discovered deep into soak testing cycles. However, unfettered access to compute cycles can lead to complacency about the

efficiency and effectiveness of verification strategies. The established dilemma is:

"I want to run ONLY "good" verification cycles!"

So, when are verification cycles "good cycles"? Only if "Verification Progress" has been made. By that we mean that it has either found a new bug, or it has measurably increased the testing space i.e. demonstrated correctness. The former is easy to track (we can count bugs), but the latter is harder because we don't know precisely how many bugs are present. We wish we did!

At the end of the day, the cost of the Engineering Platform needs to be accounted for as part of the product ROI calculation.

2.2.1 Capacity Planning and Demand Forecasting

If the Engineering Platform is a shared resource, shared between multiple product development teams, then the available capacity has to be managed unless the capacity far exceeds the demand, which is not typically the case. To avoid a situation when demand outstrips capacity, governance and decision making are required to prioritize and schedule capacity fairly and with optimal overall benefit to the business. Good forecasting practices are required to plan for capacity scaling in a timely manner. Actually, good practices and also good behaviors are required. If teams are aware that they are in competition with other teams for shared resources, it's tempting to inflate the 'demand' a little, knowing that there is a negotiation process where a lower 'allocation' may be given. How do you encourage a more altruistic attitude to demand forecasting? Forecasting needs to be as data-driven as possible, and less opinion or judgement based. This depends on having good historical data (the 'utilization').

Capacity management is driven by 'Demand, 'Capacity', 'Allocation' and 'Utilization'.

As previously mentioned, constrained-random verification techniques have proven to be very effective at flushing out obscure bugs. We stated earlier that verification is a "resource-limited quest" and a consequence of this is the likelihood of consuming a lot of expensive compute resource for marginal gains. Verification teams want to be 'smarter' about verification, but there may be some considerable effectiveness challenges, leading to the consumption of many 'dumb' cycles (e.g. repeated cycles where there is no variance in the stimulus, adding little or no value to the testing). However, in the course of a product development, there may be times where extra compute capacity becomes available for some reason, and it makes sense to consume those additional 'available' cycles for the sake of keeping the platform running at close to maximum throughput[15], and opportunistically increasing verification assurance levels. When this happens, it is important to be able to recognize this opportunistic consumption because the utilization data will subsequently be used as the basis for future demand forecasting. Note that if you are using cloud-based compute, the model is more likely to be PAYG so there may be less opportunity to consume available capacities, or the decision will be more budget driven.

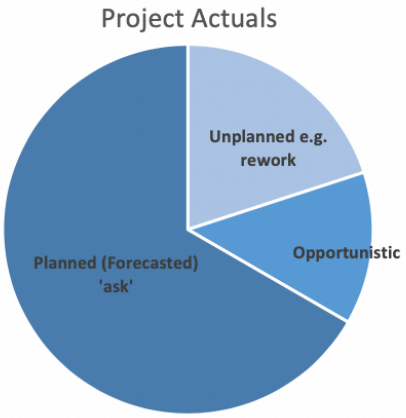

Figure 4: Platform Consumption (fake data)

Some businesses operate the Engineering Platform as a charge-back service. There is a cost model associated with the platform and project teams are cross-charged based on allocation and/or utilization. This charge-back model can lead to better demand forecasting and utilization behaviors and helps with product development ROI transparency. The cost model for the platform can be simple or sophisticated, it doesn't matter, but it does need to account for all aspects of the service operation including hardware infrastructures, licenses, staffing, energy, plant, maintenance, depreciation, insurance and business continuity costs. This model also lends itself to an easier transition to cloud services where cost models are the norm and allows teams to treat any on-prem platforms and cloud services in the same way. At the end of the day, development teams should not care if the platform is being provisioned internally/on-prem, or externally with cloud services. They should only care about the QoS, availability, and the cost.

[15] Clearly you don't want expensive 'paid-for' capacity sitting idle and for every compute slot to be doing useful work for as much time as possible.

Capacity management is a highly data-driven enterprise, requiring good data engineering platforms, good data curation practices, good data visualization, and good data science practices to gain insights from the data and drive improvements in areas such as prediction-based forecasting.

Ultimately, Capacity Management is all about data.

2.2.2 Operational Analytics (Telemetry)

Extensive operational analytics are needed in order to operate a resilient and performant platform service delivering the appropriate QoS. These analytics will monitor system performance and capacities, track operational metrics over time, raise alerts when interventions are required, and may exploit machine learning prediction algorithms to alert to pending failures so that mitigations can be deployed ahead of a critical failure point. Such systems may become 'self-healing' as interventions can be applied before end-users notice any impact on QoS.

Designing, deploying and operating widescale platform telemetry systems is a major and important part service operation of the platforms requiring many software and analytical systems.

2.2.3 Scalability/Flex

Scaling the platform to meet current and forecasted demand is a significant challenge for most businesses. Historically there has been a continuous growth in demand for capacity driven by the demand for higher volumes of testing driven by ever increasing product complexity. This demand growth has not always been linear, and clearly un-checked growth may lead to products with negative ROI, so some limits have to be set on platform consumption and developers have to find smarter and more effective ways to wring out the bugs and deliver the required product quality. Note also that platform expansions are expensive and take time to execute. They need to account for lead times on physical equipment, environmental/facility constraints, cost constraints, and human resource constraints. Unplanned capacity expansions typically cannot be met overnight and robust demand forecasting is key to ensuring that the capacities are available when they are needed.

For an on-prem platform, the ability to 'flex' capacity is a logistical and costly challenge.

Note there is trend to using so called co-located premises (or Colo)[16] for computing, providing a faster route, specialist facilities, higher security and rapid implementation of new capacity.

For businesses that share the platform between many projects, overlapping and competing project schedules can lead to some peaks in the aggregated demand forecast. Capacity planners need to smooth out these peaks as far as possible, in order to make the total 'demand' fit within the available capacity. However, projects will still need to be able to meet some peak demands, especially as they work towards key milestones, or to account for unexpected events such as urgent rework. Some ability to 'flex' overall platform capacity, either up or down, is desirable. Cloud services are well suited to this, as they have vast resources available on-demand, and can spin up additional capacity almost instantaneously, for the right price point of course. Some combination of on-prem capacity to meet predicted demand, and flexible cloud capacity to manage peaks and troughs seems like a good compromise. The key enabler to such an on-prem+cloud strategy, however, is portable workflows and portable data. Workflow encapsulation enables activities such as simulation to be dynamically allocated to either on-prem or cloud with the underlying details of the platform environment abstracted from the user. Portable workloads are not only a challenge from the user viewpoint, but also for the EDA companies who have to have an established commercial model for cloud and relevant tooling to allow seamless job switching.

2.2.4 Efficiency

Platform operators must continuously strive to maximize the efficiency of the platform. This means looking at all aspects of the platform from the raw performance of the compute resources, to network, WAN and storage performance, efficiency of work scheduling algorithms (load balancing), and also the performance of the software applications that are consuming the compute resources. The delivered capability of the Engineering Platform is dependent on all of these things and not only the raw slot count capacity of the estate. For example, server and storage hardware refreshes are normally cyclic and hardware suppliers are constantly improving the raw performance and capability of their products in line with memory and processer technology advances, so refreshing old slots for new slots normally brings with it a welcome platform performance/capacity uplift. Similarly, EDA tool suppliers are constantly refining and optimizing their products, so that they can perform more work in less compute time and the tools become more efficient. Additionally, product development teams may refine verification methodologies to do more effective

[16] https://www.racksolutions.com/news/data-center-trends/what-is-a-colocation-data-center/

work, i.e. more "good cycles", are achieved with less platform/EDA tool consumption.

Platform efficiency improvements, EDA tool improvements and methodology effectiveness improvements need to be able to keep up with escalating product complexity trends in order to avoid runaway platform demand and costs.

2.3 EDA Tools Investment

2.3.1 Cost of tools for finding Bugs

Most companies use a variety of technologies to search for bugs in designs. It is expensive to deploy some of these technologies, so there are practicalities and compromises that have to be explored versus cost before decisions are made.

EDA tool licensing costs are a VERY significant part of the overall Engineering Platform cost.

Businesses need to decide if they are going to align with a preferred vendor and drive standardization within the engineering teams, or, make multiple-vendor options available so that teams can choose according to their preferences. Although the latter option sounds like the most expensive, it fosters healthy competition between vendors both in terms of cost and capability. Engineering teams need to regularly benchmark tools in terms of performance and capability to ensure that the EDA industry continually improves its' offerings so that product teams can drive down development costs. Some of this burden of benchmarking can be achieved by sharing representative designs and payloads to the respective vendors, under appropriate IP access agreements of course.

There is innovation happening in how some vendors deliver these technologies; – rather than purchasing certain tools, you can pay for what you need, when you need it, with so called pay-as-you-go (PAYG). By offering an alternative, this potentially brings CAPEX intensive technologies such as FPGA and emulation into the reach of more design teams. It is a developing area, with new players entering the fray alongside traditional EDA Vendors, so is worth investigating fully before procurement decisions are made.

	Simulation	Emulation	FPGA	Formal
Speed	<1KHz	<1MHz	<50MHz	n/a (static)
Debug visibility	H	M	L	H
Cost to purchase	L	H	H	M
Cost per cycle	H	M	L	n/a
Set-up effort	L	M	H	M
Route to new commercial models	H	M	M	unknown
Up-time	H	M	H	H

Figure 5: Verification Methodologies Compared

As a design progresses towards final release, fewer and fewer changes need to be made. As the code becomes more stable it can be run on faster technologies to reduce overall simulation times. FPGA simulations run much faster than simulators running on cluster but are more difficult and time-consuming to set up. There is a trade-off; engineering teams can accept longer deployment times to reap the benefit of greater performance to test for bugs more widely and earlier in the design process, in both hardware and software.

2.3.2 Supplier Relationships and what to look for in EDA vendors

Ecosystem support

Many companies using EDA tools operate multi-vendor workflows, and this is certainly true for verification and debug. It is not unusual to find a simulator from one vendor being used alongside a debug solution from another, for example. In most cases there are standards agreed among vendors that govern the formats that can be used to transfer data from one tool to another, but this is not universally true.

Partnership vs transactional sales

EDA tools are complex and difficult to develop and maintain. There is constant pressure to optimize tools for new design challenges that come along and this translates into fierce competition among vendors. This means tools are expensive as vendors naturally try to pass R&D costs on to customers and remain profitable.

So how do you improve your leverage when it comes to negotiations?

Due to the competitive nature of the market for EDA tools, vendors often look for opportunities to get early adoption of new tools and features and want to promote this to the rest of the semiconductor world. That is the time to seek evaluations, especially for leading edge tools you might adopt. See if there is some way to work with

the vendor in a more partnership-like way, rather than just being another sales cycle. It might help with the commercial outcome you get, as well as making for a better technical relationship as you seek to deploy the tools into production.

2.3.3 New trends

Machine learning

Machine learning applied to EDA tools became marketing reality in PR in early 2018 and has progressed from then. When it comes to better targeting for large simulation jobs, it certainly has a role alongside in-house efforts to use "big data" to do the same. Progress can be seen in functional simulation and physical simulation areas, where improved use of data can help provide a faster route to results, or more accurate outcomes. Extensive evaluation of ML capabilities is essential to assess real gains in relation to your specific verification workloads. Some ML capabilities may need a degree of "personalization", which the vendor will provide to in the form of payable services.

Simulation in Cloud and Cloud SaaS

As you would expect, the big three EDA vendors all claim to offer cloud capabilities, but they are not alone as various disruptors vie for a position in this growing sector.

Why should you care if you already have on-prem capacity? There are various use cases that might lend themselves to a cloud-based approach, be it using your own licenses, or a specific cloud SaaS offering;

- **BCM**[17] – most self-managed on-prem compute facilities don't necessarily offer the most robust compute platforms versus what a service running on one of the big 3 cloud vendors can achieve.
- **Spikey workloads** – capacity management is hard, as already discussed, so a viable bank of last resort option is to transfer spike demand to cloud options.
- **Throughput** – cloud offers ubiquitous compute, so an on-prem capacity crunch can have undesirable effects on throughput of jobs. This can cause delays in product release plans. Cloud can get around these problems. In the case of testbenches that run slowly, just throw more licenses at it and the run the testbench in the cloud, making up for the slow individual performance shortfall.
- **Performance** – EDA companies are starting to offer significant increases in turnaround time as they can throw (in theory) infinite numbers of licenses at simulation jobs. However, this does not guarantee you are running good cycles, so the question of better targeting remains a significant issue.
- **Targeting** – big EDA have tried to improve bugs found by introducing intelligence into the cloud. Machine Learning has been shown to improve targeting in the cloud, therefore offers both performance and quality gains. Quite often these engagements require a somewhat bespoke services agreement in addition to the license and cloud costs.
- **Specialist Disruptors** – New vendors now exist that offer simulation specific cloud services.

Cloud verification strategies – Make vs Buy…

If you decide to leap into cloud-based compute, especially for intensive processes like verification, there are some basic decisions to take regarding which route you want to follow. From the engineering side the overwhelming view will be that…

"…it can't be that difficult to put sim jobs in the cloud, please just get on with it!".[18]

So, there may be considerable pressure to adopt a home-grown approach. The alternative is to let someone else do most of the worrying and buy a service from a third-party provider.

Gotchas with "make" your own cloud-based simulation system

If you intend to use your own EDA licenses, cloud-based implementations require significant development of existing orchestration capabilities in order to offer users seamless transition of jobs from on-prem to cloud.

Your current on-prem workflow probably offers easy transitions from simulator to interactive debug tools, without delays or complications. Decisions have to made about how interactive simulation and debug licenses could be run in the cloud in a performant way.

Cloud cycles are not necessarily cheaper than on-prem (often the reverse), so one major concern is to ensure that engineers' access to cloud is managed. This involves throttling applications, zombie-job deletion etc. in order to prevent massive bills at the end of the month from your cloud supplier.

As mentioned earlier, one of the key ingredients needed to make sure you are running good-cycles is excellent analytics, especially around cost. Cloud-based models should make this easier, but this measurement capability still needs to be developed if you intend to "make" your own cloud capability.

License contract provisions are another area to thoroughly check before embarking on a grow-your-own strategy. Standard EDA contracts will not usually offer the option to use your licenses in the cloud. It's more likely to be

[17] Business Continuity Management

[18] Read in a strong Glaswegian accent.

specifically prohibited. Getting permission will involve some contract re-negotiation to get the rights and expect the cost to go up.

Storage costs in the cloud can appear small for small amounts of data, but with complex simulations, vast amounts of data can soon grow costs. Careful analysis needs to be applied to your specific workflows to make sure the ROI of cloud vs on-prem works.

The "buy" complete integrated service option

Cloud based "Platforms" offering a complete service are now available, which can avoid many of the issues associated with the "make" option discussed above. They claim to provide all aspects of the workflow in one service with web-based verification manager, coverage reporting and debug tools, along with connection into existing infrastructure (e.g. storage). Effectively offering a complete SaaS-like solution, or "Verification-as-a-Service". Some combine these technical capabilities with a new PAYG commercial model which principally avoids new capital expenditure. This is very attractive when it comes to accurate cost and ROI analysis. Big EDA are starting to make similar offerings.

Although attractive, the cost per cycle, or minute (depending on the commercial model) needs to be carefully compared to on-site controlled cloud initiatives. These cost models are necessarily complex as they have to reduce all costs of on-prem, make-cloud options, or SaaS offerings to a common denominator, such as cost-per-cycle. Cloud often appears very expensive but beware of false comparisons which do not take all your existing infrastructure costs (including IT staff) into consideration. Not every organization has the cost modelling capabilities to do this in-house, so may turn to others for advice.

2.3.4 How do you know you are getting best-in-class?

Engineers presented with complex technical problems and tight deadlines can be trusted to do a good deal of due diligence, looking at leading-edge tools and capabilities from big EDA and Tier 2 vendors that will improve Quality of Results (QoR), accuracy or performance. Often the Tier 2 vendors are the ones tackling some of the really nasty challenges, using VC money. Once they have cracked it and a couple of notable customers start using the tools on real projects, big EDA usually end up buying them. Be prepared to work with the small guys in a collaborative way to help get the tools they are developing to a stage you can use them. Expect the big guys to provide longer term financial viability; think BCM by buying them.

Let's talk about evaluations and the role they play in this process. First of all, let's assume you have a clear EDA and IT strategy that details what you are trying to deliver to your engineering team. It's a big assumption, as many do not have a clear handle on strategy, so it worth taking the time to develop one. One of the key questions is;

What is the engineering capability end-state you are gunning for?

If that is known, then it is much easier to define your strategy and then decide what evaluations are in-scope and need to be carried out.

At this point, define clear evaluation criteria. Work with EDA but be robust. Work with them to define exactly how they will get you to your desired end-state. Set timescales and never forget to define the commercial end-state so there are no nasty surprises.

In as far as your organizational structure allows you to, consider setting up a governance structure to overlook the evaluation process. It should have representation from all interested parties including engineering, buyers, senior management budget holders, finance, facilities and engineering IT. Most new tool introductions will require agreement across all these areas, especially if you are talking about extra hardware like emulation and FPGA, requiring space and facilities.

Make sure you have an approach to evaluations that avoids anarchy by adding some structure (engineers try and buy at will, costing a fortune), but that doesn't stifle innovation. Recognize engineering, understand their requirements and get them involved in the technical *and commercial* process.

2.4 Methodology Investment

Since we are focused upon Bugs as the theme for this discussion, we shall limit our analysis to verification. Also, as mentioned previously, we assert that verification is the dominant consumer of platform resources for complex hardware IP development, the other consumers being Verilog design and implementation workflows.

Given the aforementioned scale of the verification consumption, there is a lot of opportunity to make radical cost savings through methodology effectiveness improvements. If platform availability is infinite and costs are irrelevant, we might not worry about effectiveness too much and instead focus our efforts on managing the execution of the verification payloads to maximize the throughput, regardless of how effective the payloads are at finding the bugs. We may be wasting valuable compute resources by: -

- Repetitively re-running the same tests so that there is no net gain in verification progress.
- Not smartly detecting and terminating runaway jobs quickly, possibly running huge regressions where the results are invalid and will be thrown away.
- Running verification payloads too early, when the RTL model is simply not stable enough. If we are generating more test failures than we can humanly

debug, then debug time is the limiting factor, not compute time.
- Not running our jobs at the most optimized performance because we forgot to enable the correct optimization switches, or we have unnecessarily encumbered the runs with poor testbench design or poor choice of tools.

Of course, in reality the platform is not free, and availability is constrained, so there is a constant need to wring the best possible ROI out of those consumed resources and deliver the product to the highest quality achievable in the shortest time.

Scarcity of platform resources/capacity can help to drive innovation for verification methodology developers.

Which are the smartest methods? How do I reduce wasted testing cycles? Can I measure effectiveness to facilitate continuous improvement? Verification becomes a very analytical discipline. It's all about having good data and good strategies to visualize and exploit that data in order to drive improvements. This is a rich opportunity space for data science where machine learning methods can be applied to large verification datasets and real effectiveness gains can be achieved. "The Origin of Bugs"[19] discusses how data and analytics can help to understand and improve bug search methodologies.

3 The Cost of not finding Bugs

In the introduction we already talked about some of the more public examples of the impact cost of critical bugs.

When bugs are discovered post release or post deployment, the total impact cost can avalanche as the impact cascades down the supply chain to the final end user.

Rework costs for the IP developer might be bad enough, but when that IP is deployed at scale into critical systems, or millions/billions of consumer products, the potential rework costs could be astronomical. Additionally, those rework costs are likely to divert IP developer resources, human and infrastructure, and impact the delivery of the business roadmaps. In turn, this will impact revenues across the business, not only for the product being reworked. To build a picture of the total impact all of these areas must be considered and measured where possible.

Further, to build a full cost impact perspective, the cost modelling needs to be applied to the multiple contexts of the

1. IP Product Developers
2. IP Product Consumers (e.g. SoC Developers)
3. Consumer Product Developers (e.g. device/equipment developers)
4. Third party software developers (Applications, OSs, Tools, ecosystems)
5. End Users (the device consumers)

The IP Developer is typically not able to assess this complete picture, but you get a sense of how a simple bug (which might be root caused to something as simple as a missing term in a line of Verilog, or a typo that has gone unnoticed) can be responsible for potentially vast and far reaching cost impacts that get passed down the development chain.

In general, the cost impacts fall into the three major categories of: -

- **People** costs – the engineering and management time consumed.
- **Platform** costs – the cost of the platform resources consumed (includes all elements of the platform including EDA tools).
- **Sales** costs – the cost impact to sales and revenues for the business.

It's worth developing this total cost-impact model in more detail.

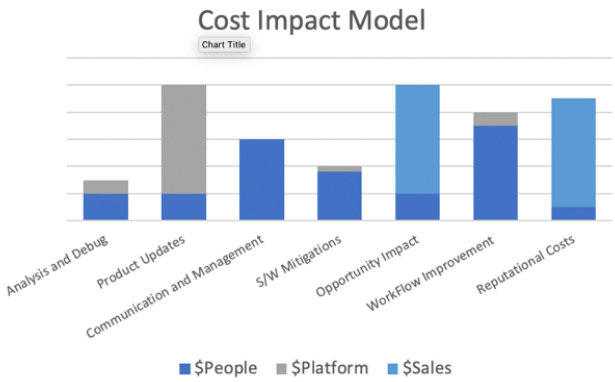

Figure 6: Example Cost Impact Model (fake data)

3.1 Analysis and Debug

If a bug has escaped the product development verification effort, it must have been found post release by some means or other. It may have been found internally through ongoing verification efforts, coincidentally by a related product team, by the customer during their own product development, or in the case of something like Spectre and Meltdown as cited earlier, by independent software

[19] https://www.valytic.co.uk/whitepapers

developers or security penetration testing teams. Either way, the full scope and impact of the bug has to be fully analyzed and documented by the IP developer. This can entail considerable time and effort depending on the complexity of the bug. The mechanisms of the bug and the implications of the bug have to be fully understood and an appropriate category assigned to the bug. The severity of the bug will dictate what happens next.

As a follow on to the analysis phase, the IP developer should conduct a formal or informal 'Root Cause Analysis' (RCA) process. This process needs to identify the root causes for the bug being in the design and also the root causes for the bug being evaded by verification efforts. This analysis may lead to a number of corrective actions e.g. extensions to the verification environment to address the testing shortfall, a review of other related areas that might be affected by similar sibling issues, or an analysis of related IP products that may also be impacted by the bug.

3.2 Product Updates

If a hardware mitigation is the only viable solution, this will trigger a product update with all of the associated rework that this generates, and re-delivery of the product. Note that a product update may incur substantial rework costs. Even a simple RTL change might affect nearly all of the product deliverables including documentation and reference workflows.

The hardware mitigation may also incur a range of costs for the customer depending on where they are in their own development cycle.

3.3 Communication and Management

How the IP developers manage the communications around bugs is as important as how they fix the bugs. Communications need to be timely and accurate. Customers may wish for a 'heads-up' whenever an impactful bug is being triaged and diagnosed, especially if an early intercept could avoid incurring large rework costs. However, these communications need to be carefully managed, as falsely raising the alarm too early can be just as costly. IP developers need to react quickly and be able to divert the appropriate engineering resources onto the analysis, impact assessment and documentation phase to ensure that accurate and complete information is available at the earliest opportunity. This often entails a diversion of resources with knock-on impacts to other critical work.

3.4 Software Mitigations

Once understood, the next priority is to develop and deploy any possible mitigations, especially if the IP is already deployed into customer products and is in the field. In the best case, a simple software mitigation can be deployed that will avoid the bug but will not be detrimental to the performance of the product. Clearly there is a burden of verification for any software mitigation that is released into the ecosystem. If the software mitigation is deemed to be impactful, it may be considered as an interim patch, until a hardware mitigation can be made available. Software mitigations can be costly if the impact is far reaching into the software ecosystem.

3.5 Workflow Improvement Costs

The RCA process should also attempt to identify practical long-term improvement measures as part of a philosophy of continuous improvement. These preventative measures should be implemented in a way that codifies them into engineering workflows in a way that ensures the same error cannot occur again for the affected product or any other product. This might be as small as adding an incremental improvement to an existing workflow, or it could entail the development and introduction of a new methodology or workflow with a much higher but necessary cost impact.

3.6 Opportunity Costs

As discussed above, late bugs, discovered after product release, can be very impactful in terms of resources, human and non-human. Oftentimes you need to call upon some of your most talented people to drop what they are doing and prioritize the bug analysis and subsequent rework. And those talented people may have moved onto the next product development, so there is an impact to these other important projects also. Opportunities to deliver new products against the business's roadmaps can be impacted with the knock-on impact to staffing, or engineering platform availability. That in turn may lead to a delay in revenues as those roadmap products come to market later, or even loss of business as market windows are missed due to unforeseen development delays. It's important to keep an eye on these knock-on effects when finite resources are drained and diverted onto unplanned and critical rework due to bugs.

There are also opportunity costs at a more personal level for the critical staff members impacted by the bugs. Aspiring developers do not want to be 'stuck' working on legacy products and want to move onto the next exciting project where they can innovate, develop and implement exciting fresh product ideas. Bug rework may not be 'exciting' from a personal development point of view, in fact it might be 'frustrating' and demotivating work, even though it is critical work. Glory can be found from leadership, technical or non-technical, in times of crisis management, and the problem solving required may be extremely challenging and therefore potentially rewarding. But at the same time, it is human nature to want to move onto the next great thing, a new product development, and leave historical projects behind. Of

course, the business may choose to staff up product maintenance and rework in a different way, protecting the new product development teams from distractions, and interruptions as far as possible. However, even with the best documented and cleanest codebase, those closest to the product development have the best understanding of their code, may have undocumented insights into the workings of it, and can solve the problems in the shortest time. The cost here is harder to quantify in currency terms, but staff engagement is an important aspect and we know that staff hiring, development and retention costs can be high and the impact of losing key staff affects the business's ability to innovate and generate revenue.

3.7 Reputational Costs

Vulnerabilities such as Spectre, Meltdown, and the infamous Pentium FDIV bug were headline news in the semiconductor world. They did make it into mainstream media and hit public consciousness to a level. Silicon design happens so far up the design chain that it is usually obscured from the public eye by powerful OEM marketing machines. Within the semiconductor world this was big news, however, and major IP suppliers had to move very quickly to provide mitigation and excellent communication about workarounds and mitigation to their customers help them avoid serious consequences in their product designs. By providing this level of service, the IP vendors were able to maintain vital levels of trust among their licensees. Part of that messaging will also have been about what future actions they were going to be making to avoid the same thing happening again, in the certain knowledge that the customer base would not be so tolerant next time around. The importance of maintaining that level of trust also extends to the investment community, who want to know that their stock holdings are not subject to negative impacts driven by major post-release bug discovery. Some vendor Annual Reports go to some lengths to explain the issues, the levels of risk and investment attached to them and plans to mitigate the impact on sales and financial provisions for recompense to unhappy customers (source Intel Annual Report 2018[20]).

4 Conclusion

So, what is the real question here? How can you start using this information to challenge your existing understanding and ensure you are getting the best ROI within acceptable risk levels?

As we said at the start, it boils down to your understanding of the total cost of bugs expressed as the costs burned in finding bugs (probably the most significant part of your total IP Product development cost) versus the cost of not finding bugs, i.e. what is the total impact cost when bugs are missed?

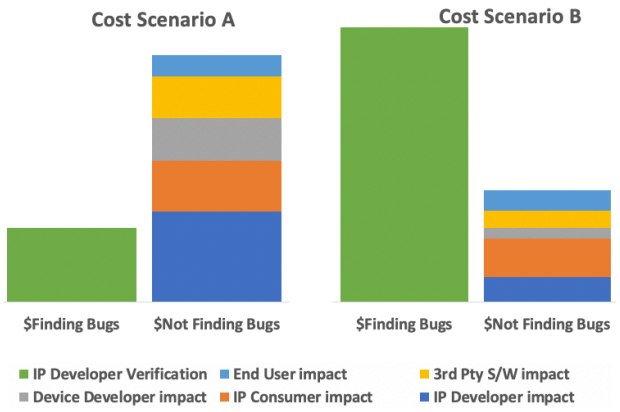

Figure 7: Which Scenario are you currently in? (fake data)

So, the question is, which of the above 2 scenarios are you in? Are you cost-to-find *low*, but high risk/impact regretting not having spent enough on verification, or are you cost-to-find *high* with impact risk/cost low wondering if you are spending too much?

Do you have enough data to be able to reason about this and decide which scenario applies to your operations? If not, then what actions should you be taking to find out where you are and what action to take in each situation?

If your analysis shows that you are in scenario A, it looks like you need to address product quality urgently. You're not really investing sufficiently in IP Product Verification and the impact costs are significantly reducing your products' ROI. Scenario B certainly feels more comfortable, but you might have nagging doubts about the efficiency of your operations.

It has long been said that verification costs overshadow design costs, and this does not look set to change any time soon. However, looking at your verification capability in terms of cost vs impact throws important light onto the risks you are running.

There is no verification "silver bullet" and there never will be. However, engineering teams can exploit the power of data to understand these costs, risks and impacts and seek to make verification as efficient and effective as possible, always striving to run "good" cycles and eliminate wastage and inefficiencies wherever possible. An analytical approach to the problem can enlighten teams to make good choices about the best methodologies and best tools and how to keep platform costs to a minimum. Data Science is coming to the rescue!

[20] https://s21.q4cdn.com/600692695/files/doc_financials/2018/Annual/Intel-2018-Annual-Report_INTC.pdf