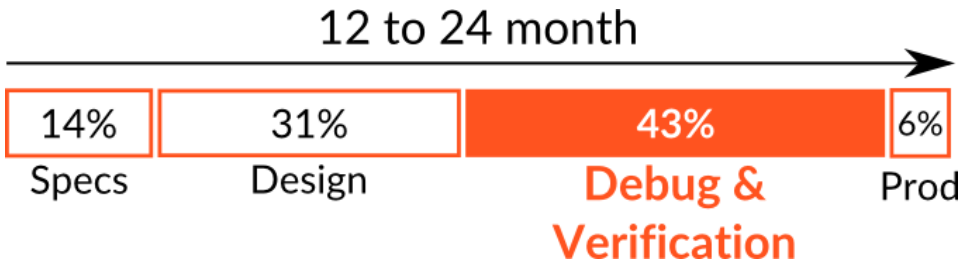
EXOSTIV LABS

White Paper

Upgrade tools to boost FPGA debug

The complexity of modern electronic systems is exploding. More gates per square millimeter, more computing power, larger memories, faster I/Os and busses... Yet, engineers worldwide have managed to keep up with the pace of technology and have been able to produce state-of-the-art electronic systems.

The secret? More spending on verification. More time, more resources and more money. As the systems have grown in size or in complexity, the work of verifying them has become the largest slice of the project time budget, **accounting for over 40% the total design time and more.** The efficiency of chasing bugs has become the determinant factor for reaching a product market window.



Typical FPGA project

Verification is a huge task. The complexity is not only to check each piece of IP separately. The challenge is verifying how these pieces collectively interact. There are a lot of interactions that cannot be defined really well from the functional behavior of the system's individual pieces.

Of course, all systems are not equal in this quest for bugs.

FPGA engineers are mostly unequipped for verification.

The systems based on FPGA (or programmable logic, or whatever they are called today...) hold a very special place here. FPGA might be one of the most advanced chip technology available and yet**, FPGA engineers are mostly unequipped for verification.**

Let's have a closer look, noting that FPGA have largely inherited portions of its design -and verification- flow from its ASIC and SoC[1] cousins.

[1] In this paper, 'ASIC' and 'SoC' refer to application specific integrated circuits and system-on-chip that are manufactured in a foundry, in a specific process technology and that require masks for production. We acknowledge that once programmed, a FPGA becomes an 'application-specific IC' and that some FPGA are really 'systems-on-chip'. At the start of 2015 the search for the right name is not over yet...

Using hardware is key to the verification process.

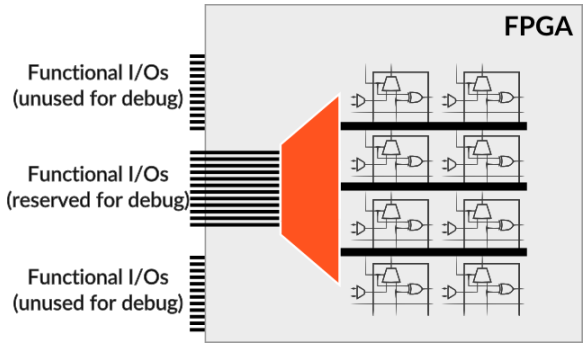
If speed of execution was not a problem, simulation would solve most of the ASIC and FPGA verification requirements. However, it can require days to simulate the first boot seconds of an integrated circuit with a conventional technique. Going for higher levels of abstraction is usually proposed to speed up simulation - but even in this case, the chance of finding bugs depends on the accuracy of the higher level behavioral models. So, simulation is an essential tool - and you'll use it if you are serious about designing an ASIC, SoC or FPGA - but you will not use it alone.

Consequently, in most cases an engineer is going to use hardware in order to verify a design in a timely manner. **There are 3 choices today: emulate, prototype or use an 'instrumented' version of the final hardware.**

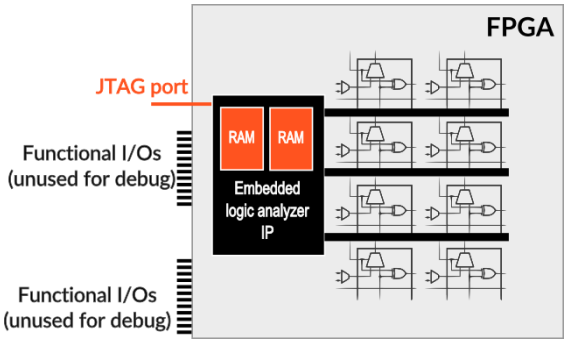
Verifying an ASIC requires emulation and/or prototyping, as it is generally unthinkable to spend the cost of manufacturing a custom integrated circuit - tenth or even hundreds millions of dollars - for iterating through verification, especially at the early stages of it. Only the designer of a target system based on FPGA has the additional option to use the 'real system' target hardware for verification, as changing the chip(s) functionality only requires re-programming them. Even the price of modifying the target board after verification is quite bearable - in the (ten) thousands to several hundred thousand dollars.

Using the target hardware for verification is seducing: no compromise on speed and no worries about properly modeling the environment or the system. In other words, you have the certitude to spend time on debugging the system, not on debugging the prototype. Unlike emulation or prototyping, the time spent on setting up the system for verification is not 'wasted', since the same procedures that will be used for the real system setup can be used and verified as well.

From 'logic analyzer' to 'embedded logic analyzer'.



Traditional logic analyzer approach.



Embedded logic analyzer approach

'Instrumented system debug' mostly relies on observing the system using the system resources or the system as its self-observer. Beyond observing the system through its interfaces with the 'external world', some of the system's resources - like a microcontroller executing a specific sequence of instructions - can be dedicated to verification. Temporary resources can be added as well. This is typically the case for a debug connector used to hook up a scope or a logic analyzer. This kind of resource may be removed from the board once it is validated and goes to production.
Specifically, FPGA engineers used to route internal nodes to the chip's I/Os wired to the debug connector and visualize the evolution of the logic nodes on the screen of a logic analyzer (figure 1).

As system speed and performance went up, it **has become increasingly hard to reserve a number of I/Os that was still relevant to really get an insight of the FPGA behavior.** In addition, the FPGA internal logic may run faster than what can be sampled at its I/Os. To use an analogy, keeping on using this technique would reduce to watching 1 image on 5 of a movie through a keyhole.

The 'logic analyzer-based' verification approach has evolved towards 'embedded logic analyzer'-based verification. In this kind of **in-system verification**, the FPGA is used as a **recorder of its own behavior**: the evolution of key internal nodes over time is recorded and stored into a memory located in the FPGA. Once the memory is full, it is read-back using the chip's JTAG interface. This clever approach saves the FPGA user I/Os and solves the 'watching speed' problem, as everything is recorded from within the FPGA, by the FPGA, at FPGA speed (figure 2).

End of the story?

Our company has been in touch with numerous FPGA engineers. **Using in-system verification resources in the form of a embedded logic analyzer is a *very* successful technique. We practically do not know any FPGA engineer who does not use it for verification.**

Does this mean that they are all very happy with it? Nope. It is a great tool because - with a careful iterative work - it greatly helps during verification. In addition, this is a rather cheap solution.

However, with the ability to collect a few hundred kilobytes of data at best, FPGA engineers complaint about not having sufficient visibility (or, we prefer: 'observability') during verification.

The problem is observability.

To understand why FPGA engineers are demanding more 'observability' - that is, the 'ability to observe the system'- let's examine what they exactly do when they verify hardware.

Verification on hardware is mostly used to check whether the various designed and pre-verified elements of a system properly work together in their future system environment[2].

Such verification sessions usually reveal the following weaknesses of system design:

[2] Of course, there can be session for checking specific pieces of the system as well, like verifying if the Ethernet connection works perfectly when hooked up on a real network.

- A block of IP was verified against an incomplete or incorrect simulation model or even under poor ‘stress conditions'. Sometimes, third party IPs have not been used in a sufficient number of real products and keep 'youth diseases' that are revealed under higher or different 'stress' conditions. The 'reality' brings up unforeseen situations that are not well handled by the system 'as a whole'.

- The system was designed based on a well-specified standard, but the environment does not match it perfectly. This may happen when the system has to interface an existing equipment for which there isn't any alternative. Even if this equipment is the industry 'de facto standard’, cases occur that it imperfectly matches a specification. This usually leads engineers to make their own equipment 'bug-compatible' with this equipment.

- The system specification is prone to interpretation. This usually results in blocks of IP badly working together because the system that 'glues them' does not use them properly.

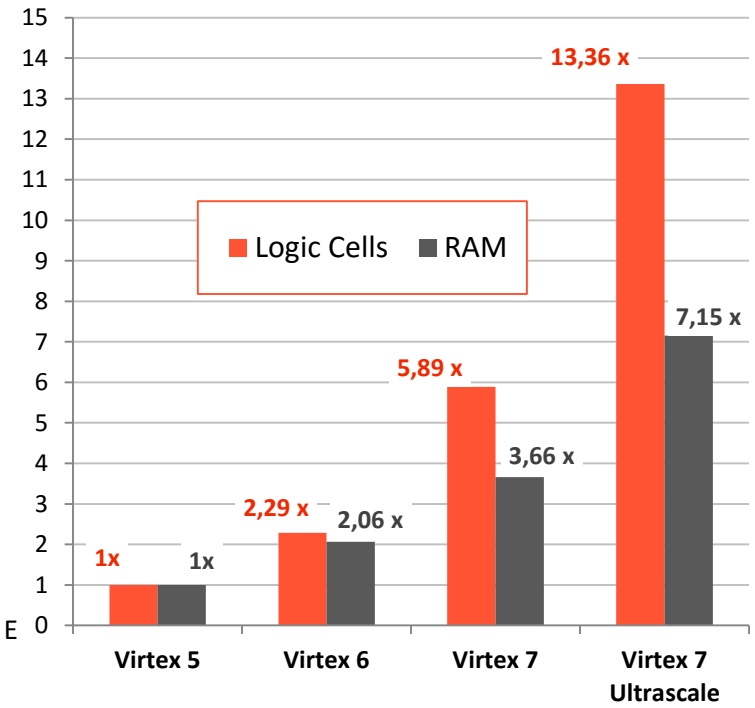
In other words, virtually anything can go wrong even if each separate piece is functionally ok.

Because 'anything' can go wrong, the verification scope of a system by using in-system verification technique virtually spans over the ***whole system.***

Being able to gather a few hundred kilobytes of data is not much when a problem may be located anywhere among hundreds of thousands FPGA 'nodes'. Verification is a tedious process that requires a lot of iterative work. The engineer's experience and system knowledge are crucial to define where to look for the potential cause of a problem.

And when a bug happens randomly once in a one-week period, verification may really reduce to finding a needle in a haystack...

Things are worsening.



This chart shows the relative evolution of the logic and memory resources for several generations of Xilinx 'Virtex' devices over time. The values for the Virtex-5 series are normalized to '1'. For the sake of simplicity, we always take the maximum of these values available for each of the FPGA families.
We can see that the quantity of logic resources is increasing faster than the amount of memory. For the latest Virtex-7 Ultrascale generation, logic resources reach more than 13

times what was available for Virtex-5, while the memory is 'just' (!) multiplied by a little more than 7.

So:

- FPGA have grown to be very complex mixes of computing resources - more than 4 million logic cells and 132 Mbit of memory as for the Virtex-7 Ultrascale family.

- If you wish to continue to use the FPGA memory to look into the FPGA logic, you are clearly losing this capability: the number of logic nodes to observe is increasing faster than the resources where you can store the debug information.

And... an embedded logic analyzer will just allow you to use a tiny fraction of the FPGA memory to observe a system that includes hundreds of thousands - to millions of nodes.

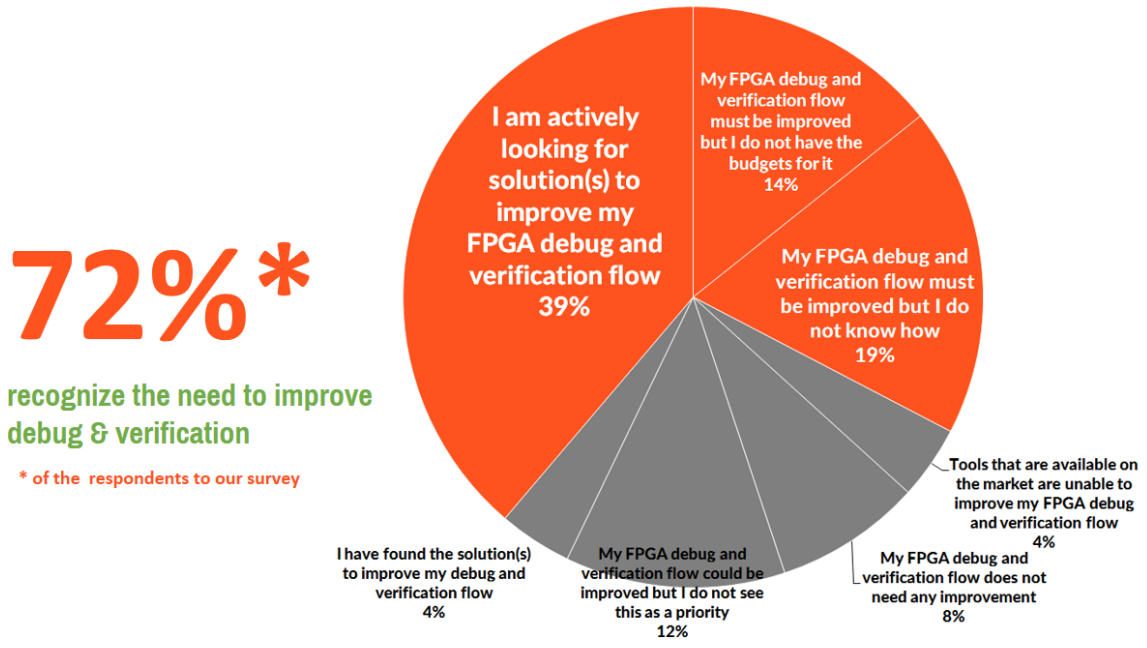
Although the FPGA engineers would like to get more observability from real hardware for verification, the tools they are using increasingly fail at providing this added observability.

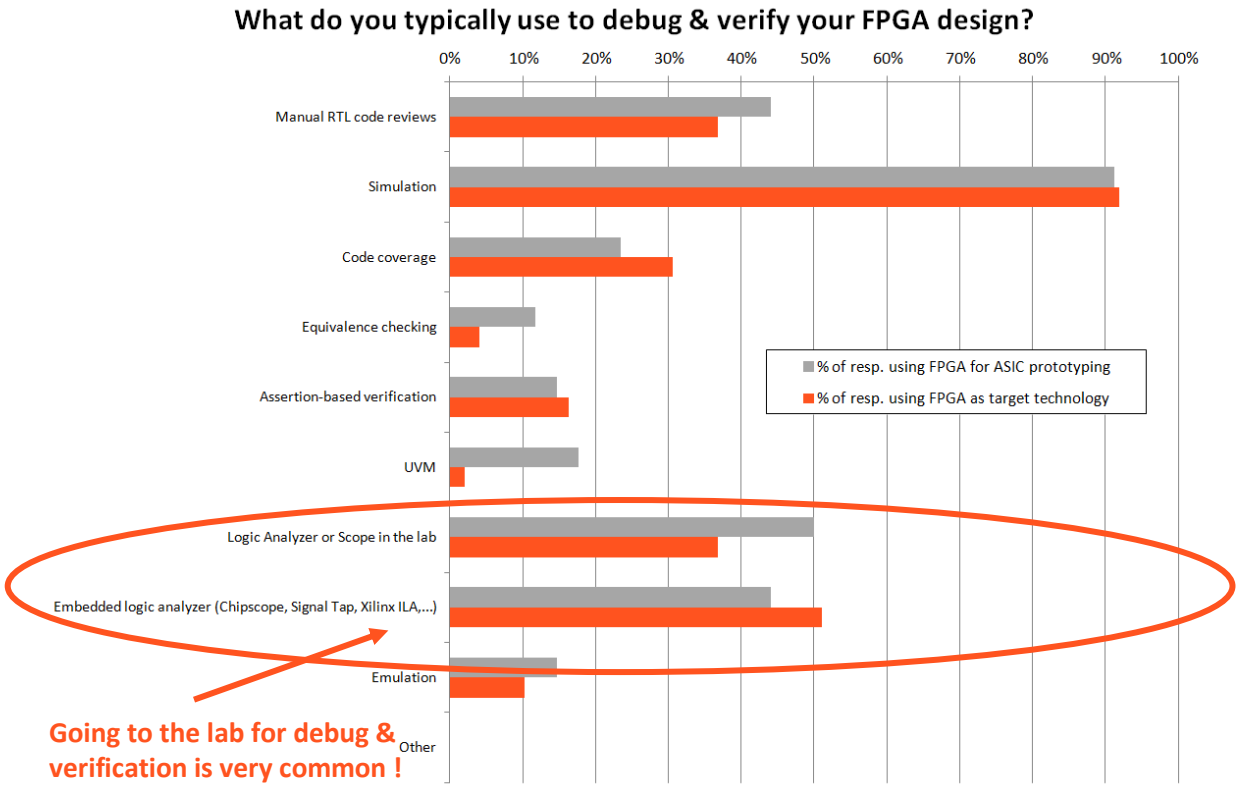
FPGA verification tools need an upgrade

Some results from the field

We have recently surveyed FPGA engineers about debug and verification. See the results below:

1) a majority of the respondents recognize the **need for an improvement** of the flow – and:
2) in the world of FPGA designers, **going to the lab** (= using a real hardware) is very common for debug & verification tasks. This spans across virtually all industries.





Emulation as solution for FPGA verification?

Emulators are quite complex pieces of hardware based on application-specific processors and FPGA that are able to emulate the target system at a speed of up to a few megacycles per second. In a world where systems run at multiples of hundreds megahertz or even gigahertz, this may not seem quite a bargain. However, this is by far many orders of magnitude faster than simulation and the big advantage of emulator is to reach the same level of visibility as simulation, even on very large designs.

This seems almost too easy: if traditional in-system debugging is not sufficient, why don't FPGA engineers[3] use emulation techniques?

It is a matter of economics.

It is virtually impossible for an FPGA engineer to justify a 6 to 7 figures price tag for a state-of-the-art emulator - even if this price is spread on multiple designs. The FPGA engineer does not have the argument of not risking a firm multi-million-dollar ASIC cost as a reason for spending a few hundred thousand dollars on verification tools and software.

[3] By ' FPGA engineers', we really mean those using FPGA as a technology for their final products

Isn't it hard to ask money to the management because your work contains errors?

What if it saves months of verification?

Let's take a simple example here: on an average design, 5 engineers spend 40% of the 1-year design time budget on verification. Let's assume that their salary is 120k$ yearly. The cost of 40% x 12 month= 4,8-month verification work would be: 4,8 x 120k/12 x 5 = **240 k$.**

Even if emulation reduces the verification to 1 month, it would save **190 k$** in working hours. Would this justify investing perhaps 500 k$ - or more - in emulation? Well, yes, only if you are *certain* that the emulator will work, and if you can reuse the same investment for multiple designs. Perhaps the gain will be smaller. **But this is absolutely not a no-brainer case[4].**

How about prototyping?

There are many prototyping hardware systems available on the market today: they range from industry-oriented prototyping boards (e.g.: FPGA boards for video applications) to general-purpose set of boards that can be assembled like a Lego game to match the needs of the target system.

The fitness of a prototyping system depends on the application and at which stage you wish to use the prototyping system. Unlike ASIC or SoC design, prototyping an entire FPGA-based system is not mainly a matter of getting a sufficient number of gates to map the design onto.

If and when a system properly works on a prototype, it still must be verified with the real hardware. Back-and-forth session from the instrumented system and the prototyping system can be useful if a bug is detected.

However, beyond the price of a prototyping system, engineers report being reluctant of risking to 'waste' precious hours on debugging the prototype, rather than going directly to an -even augmented- version of the final system. This may be a problem of perception towards prototyping... or not.

Conclusions: reload FPGA debug

FPGA system verification needs a new generation of tools that builds up on the success of using instrumented version of the target system with more advanced in-system verification features.

This new generation of tools should provide an increased observability that is in line with the fantastic complexities reached by FPGA today.

Finally, 'reloading FPGA debug' cannot just be inspired by what exists in ASIC and SoC design flows, as designing a FPGA system has its own constraints and its own economics.

---

[4] There are other aspects to consider: are the engineers properly trained to use the emulator? What is the cost for setting it up?

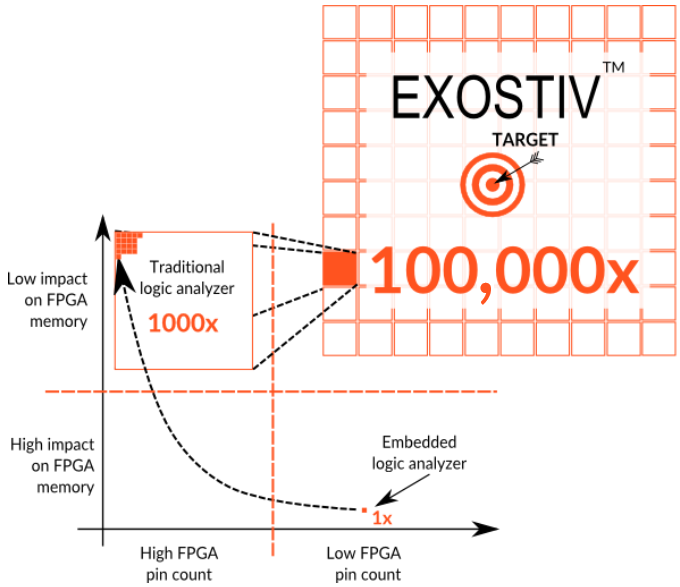
Introducing Exostiv Labs solutions

Exostiv Labs is a division of Byte Paradigm and a trade name for the company's solutions focusing on FPGA debug.

'The ability to see inside the chips is the way to build better design.'

Exostiv Labs is investing daily in understanding the constraints and requirements of FPGA system verification. **'Get more visibility.** This is the first demand that we get.

Visibility (observability) has its price. All engineers worldwide know that long gone are the days when people where asking for large debug traces without any impact on the FPGA resources, design flow or verification methodology.



EXOSTIV™ is the first FPGA in-system debug solution that uses the FPGA multi-gigabit transceivers (MGT) to flow the debug data out of the FPGA into an external memory. This approach increases the observability by a factor of 200,000 as compared to current embedded logic analyzers. It preserves the FPGA standard I/Os and memory for an observability increased by many orders of magnitude.

We believe that the first step towards more observability is creating the infrastructure that allows collecting large (gigabytes, terabytes, …) FPGA debug data from an instrumented system.
This requires scaling both the hardware the software tools used for FPGA verification.

About the author:
Frederic Leens is the CEO of Exostiv Labs
He can be reached at frederic.leens@exostivlabs.com
www.exostivlabs.com