University of BRISTOL

29th March 2019

Open Source Hardware Verification

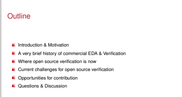
A survey and suggestions for future work

Ben Marshall

University of Bristol Computer Science Department

Outline

- Introduction & Motivation
- A very brief history of commercial EDA & Verification
- Where open source verification is now
- Current challenges for open source verification
- Opportunities for contribution
- Questions & Discussion


Outline

- Introduction & Motivation
- A very brief history of commercial EDA & Verification
- Where open source verification is now
- Current challenges for open source verification
- Opportunities for contribution
- Questions & Discussion

So, the aim of this talk is to give an overview of *functional* verification in the context of open source hardware design.
I'll start with what motivated me to look into this, which will hopefully give context to some later assertions I'd like to make.
There's an extremely brief history of EDA development (which is hopefully fairly familiar to everyone!), and a comparison between the OSDA community and the commercial alternatives.
This'll include an overview of what we *can* do with current OS tools (as opposed to what is currently done), and where there are opportunities for future contributions.
I'll try and finish a bit sooner, since this talk is very much a starting point for discussion rather than an end in itself.

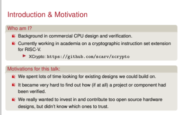
Introduction & Motivation

Who am I?

- Background in commercial CPU design and verification.
- Currently working in academia on a cryptographic instruction set extension for RISC-V.
    - XCrypto: `https://github.com/scarv/xcrypto`

Motivations for this talk:

- We spent lots of time looking for existing designs we could build on.
- It became very hard to find out how (if at all) a project or component had been verified.
- We really wanted to invest in and contribute too open source hardware designs, but didn't know which ones to trust.


Introduction & Motivation

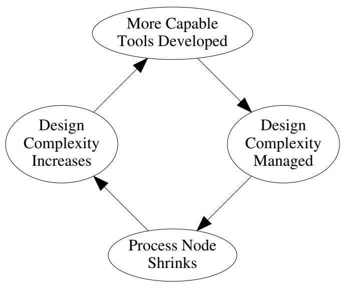
Who am I?
- Background in commercial CPU design and verification.
- Currently working in academia on a cryptographic instruction set extension for RISC-V.
    - XCrypto: https://github.com/scarv/xcrypto

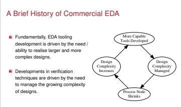
Motivations for this talk:
- We spent lots of time looking for existing designs we could build on.
- It became very hard to find out how (if at all) a project or component had been verified.
- We really wanted to invest in and contribute too open source hardware designs, but didn't know which ones to trust.

My background is not academia, I originally did CPU design and verification in industry. There I learnt about the dire state of some commercial EDA tools, and how to verify a CPU three ways: directed testing, constrained random/UVM and end-to-end formal; all using commercial tools. So it's this background which influences how I come to the open source community.
Now though, I'm working on an extension to RISC-V. It's a general purpose cryptography accelerator, designed to be a little more flexible than adding an "AES instruction" and not using the vector extension as a base. Please do come talk to me about it afterwards!
At the start of this project, we wanted to use existing RISC-V cores and SoC infrastructure as a starting point for the hardware prototype. Almost a year later, we are still really struggling to find designs we trust and that "just work". We've found a core, but were struck by how hard it was to evaluate different designs. The projects we surveyed had no *quantitative evidence* of verification effort, and it's this theme which I'll come back to later.

A Brief History of Commercial EDA

- Fundamentally, EDA tooling development is driven by the need / ability to realise larger and more complex designs.
- Developments in verification techniques are driven by the need to manage the growing complexity of designs.


More Capable Tools Developed
Design Complexity Managed
Process Node Shrinks
Design Complexity Increases



First though, I want to talk about the motivations behind commercial EDA tools. For our purposes, you can simplify it to a four step cycle: a smaller process node (or bigger FPGA) enables bigger designs, the design complexity increases, tools must be updated to cope, which makes designs more manageable in time for process nodes to shrink again. This, I think, is the painfully simplified history of EDA.

Now, enter open source EDA...

Where Open Source EDA Fits In

Design Description
Verilog / VHDL / Chisel / MyHDL / SpinalHDL / Clash / PyGears

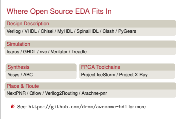
Simulation
Icarus / GHDL / nvc / Verilator / Treadle

Synthesis
Yosys / ABC

FPGA Toolchains
Project IceStorm / Project X-Ray

Place & Route
NextPNR / Qflow / Verilog2Routing / Arachne-pnr

- See: `https://github.com/drom/awesome-hdl` for more.

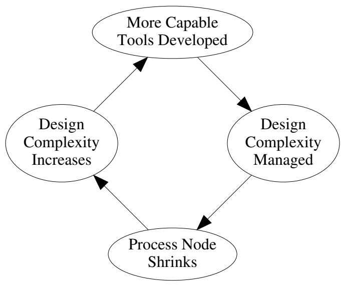


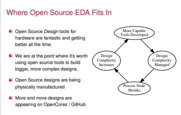
There's a lot going on! At pretty much every stage of the *design* process, there's an open source tool or framework we can use. One can argue about things like technology libraries for ASICs, but broadly, OSDA is providing some very compelling tools at all stages of the design process.
And, it's able to be a lot more innovative than it's commercial cousins. I think this is best demonstrated by the lengths people will goto to avoid writing Verilog or VHDL. "I don't care if I have to build 1000's of lines of framework and compile it to Verilog anyway, I don't want to write in Verilog!"

As an aside, I think these sorts of MetaHDL are really cool, but theres some things being missed which I'll come back to later.

Where Open Source EDA Fits In

- Open Source *Design* tools for hardware are fantastic and getting better all the time.
- We are at the point where it's worth using open source tools to build bigger, more complex designs.
- Open Source designs are being physically manufactured
- More and more designs are appearing on OpenCores / GitHub


More Capable Tools Developed
Design Complexity Managed
Process Node Shrinks
Design Complexity Increases



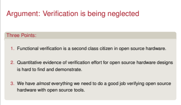
So here we are: OSDA is great, always getting better and a compelling choice for ever larger projects.
It's a joy to see so many new designs popping up on Github / GitLab / OpenCores or LibreCores, and to see so many new people getting involved because these tools don't cost the same as a family car.
But...

We are now at the point where the complexity of many open designs means they need substantial verification effort in order to be safe to re-use.

Argument: Verification is being neglected

Three Points:

1. Functional verification is a second class citizen in open source hardware.
2. Quantitative evidence of verification effort for open source hardware designs is hard to find and demonstrate.
3. We have *almost* everything we need to do a good job verifying open source hardware with open source tools.

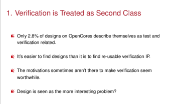

Argument: Verification is being neglected
Three Points:
1. Functional verification is a second class citizen in open source hardware.
2. Quantitative evidence of verification effort for open source hardware designs is hard to find and demonstrate.
3. We have *almost* everything we need to do a good job verifying open source hardware with open source tools.

This is the argument I want to make: I think that so far, functional verification has been somewhat neglected: both in terms of tool capability and actual engineering practice, and that we're now at an inflection point in OSDA which means that can no longer continue.
Breaking this down a bit:
Up to now, verification has been something of a second class citizen. There are plenty of reasons for this, some technical, some human.
Now point two: it may be that actually we all do perfect jobs of verification, but unless we have a way of communicating that to people browsing our designs, all that effort gets wasted.

And, most importantly, I want to convince you that we already have most of the tools we need to do a damn good job. That said, there are certainly some areas where we could make our lives easier with a bit more investment.

1. Verification is Treated as Second Class

- Only 2.8% of designs on OpenCores describe themselves as test and verification related.
- It's easier to find designs than it is to find re-usable verification IP.
- The motivations sometimes aren't there to make verification seem worthwhile.
- Design is seen as the more interesting problem?


1. Verification is Treated as Second Class
- Only 2.8% of designs on OpenCores describe themselves as test and verification related.
- It's easier to find designs than it is to find re-usable verification IP.
- The motivations sometimes aren't there to make verification seem worthwhile.
- Design is seen as the more interesting problem?

Now, I understand that this is open to interpretation. I absolutely do not mean to say people thing verification isn't important or necessary. I think we all know it's important, but that upto now in open source hardware, the focus has been on tooling needed for design: simulation, place and route and so on. After all, you can't do verification if you can't build your design.
We're now at the point where investment in verification is necessary. I don't think it's reasonable to build something as large as a CPU and show it off as a reusable open source component (with the expectation others will use it) without at least stating what kind of verification has been done and how much of the design is covered by it.
Of-course, this isn't applicable for every project, but if we want open source designs to proliferate, this is what I think is needed.

One thing I've found since falling into academia: there is almost *zero* incentive to do verification on hardware designs. You want just enough to get defensible results and publications, because that's what your funding depends on. Of-course some projects do put the effort in, but this is not the norm in my experience. This is important because big open academic projects around RISC-V are starting to set a benchmark for what open source hardware projects look like.

2. Evidence of Verification

Evidenced Verification Means:

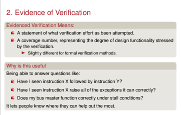
- A statement of what verification effort as been attempted.
- A coverage number, representing the degree of design functionality stressed by the verification.
  - Slightly different for formal verification methods.

Why is this useful

Being able to answer questions like:

- Have I seen instruction X followed by instruction Y?
- Have I seen instruction X raise all of the exceptions it can correctly?
- Does my bus master function correctly under stall conditions?

It lets people know where they can help out the most.


2. Evidence of Verification
Evidenced Verification Means:
- A statement of what verification effort as been attempted.
- A coverage number, representing the degree of design functionality stressed by the verification.
  - Slightly different for formal verification methods.
Why is this useful
Being able to answer questions like:
- Have I seen instruction X followed by instruction Y?
- Have I seen instruction X raise all of the exceptions it can correctly?
- Does my bus master function correctly under stall conditions?
It lets people know where they can help out the most.

So, this is the single most helpful thing new and existing projects can do to make themselves attractive to users. It's also somewhere that open source hardware can do much better than commercial counterparts.
In industry, IP gets delivered, and you take it on faith that because you paid for it, it's been well verified. The selling company will (hopefully!) have things like coverage metrics for the design to say how much of the functionality is touched by the verification (and hence tested), but this is not typically made available to customers of the IP.

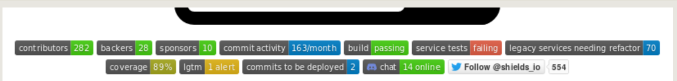
Open source hardware projects can actually show off about how well verified they are. Even an honest description of what the `testbench.v` file does is useful.

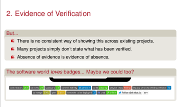
2. Evidence of Verification

But...

- There is no consistent way of showing this across existing projects.
- Many projects simply don't state what has been verified.
- Absence of evidence is evidence of absence.

The software world *loves* badges... Maybe we could too?


contributors 282 | backers 28 | sponsors 10 | commit activity 163/month | build passing | service tests failing | legacy services needing refactor 70
coverage 89% | lgtm 1 alert | commits to be deployed 2 | chat 14 online | Follow @shields_io 554


2. Evidence of Verification
But...
There is no consistent way of showing this across existing projects.
Many projects simply don't state what has been verified.
Absence of evidence is evidence of absence.
The software world loves badges... Maybe we could too?

Unfortunately at the moment, it's really hard to show this consistently across projects. Generating coverage numbers as a badge like software projects love to do would be fantastic. But, continuous integration for hardware is hard (keep an eye on LibreCores CI though) and only a few tools like Verilator support any kind of coverage collection.
Being able to quantify verification effort, and show it, is something which I think open source hardware needs and will really benefit from. It will need some tooling investment, but if it means we can look at a design and say "this looks well verified, I can use this" or "this bit isn't covered, I can contribute here", then that's a good outcome.

To borrow a phrase, "trust, but verify". We should be able to trust open source hardware designs, and that trust should be based on verifiable verification efforts.

3. We have (almost) everything we need!

Coverage Collection

- Verilator supports line and toggle coverage.
- Meta-HDLs can get line coverage for free from their parent language.
- Functional coverage is a bit more work, but possible.

Verification Libraries

- UVVM / OSVVM / VUnit / Cocotb

Formal Tools

- SymbiYosys / Z3 and friends

`github.com/ben-marshall/awesome-open-hardware-verification`

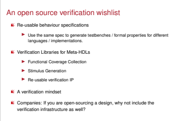
3. We have (almost) everything we need!
Coverage Collection
Verilator supports line and toggle coverage.
Meta-HDLs can get line coverage for free from their parent language.
Functional coverage is a bit more work, but possible.
Verification Libraries
UVVM / OSVVM / VUnit / Cocotb
Formal Tools
SymbiYosys / Z3 and friends
github.com/ben-marshall/awesome-open-hardware-verification

Perhaps the best thing I can say now is that actually, there are already some fantastic projects out there to make verification easier for open source projects.
If you are using Verilator then it already supports line and toggle coverage, you just have to turn it on. For MetaHDLs, you can exploit their host language tools to get things like line and branch coverage. Functional coverage it still an open problem in my opinion. If you look at what's possible in something like SystemVerilog, there's nothing comparable in open source tools at the moment.
In terms of verification libraries, theres a bunch of great stuff out there for VHDL, and Cocotb is an abstracted framework which works with several HDLs.

Of-course, we're very fortunate to have the formal tools and flow in the form of SymbiYosys. The fact you can end-to-end verify a RISC-V CPU using only open source tools is something I find absolutely amazing.

An open source verification wishlist

- Re-usable behaviour specifications
  - Use the same spec to generate testbenches / formal properties for different languages / implementations.
- Verification Libraries for Meta-HDLs
  - Functional Coverage Collection
  - Stimulus Generation
  - Re-usable verification IP
- A verification mindset
- Companies: If you are open-sourcing a design, why not include the verification infrastructure as well?


An open source verification wishlist
Re-usable behaviour specifications
Use the same spec to generate testbenches / formal properties for different languages / implementations.
Verification Libraries for Meta-HDLs
Functional Coverage Collection
Stimulus Generation
Re-usable verification IP
A verification mindset
Companies: If you are open-sourcing a design, why not include the verification infrastructure as well?

Now, having all of that is great, but I think we can do better. This is my wishlist as an open source verification engineer:
Re-usable verification IP and design specifications are probably the best way to contribute at the moment. If I can just download an AXI bus agent to use in my Verilator / GHDL design, then as a designer, that saves me a huge amount of time. Similarly, easy stimulus generation for different interfaces would be fantastic. Even better, you can imagine a package manager like FuseSoC being used to just grab all the verification IP you need! These re-usable libraries are somewhere that MetaHDLs can really show their worth. Every one I've seen so far makes the claim that "X is written in a high level language, which also makes it perfect for verification". Which is great! But I've never seen one with a corresponding verification library, it's always been about the design.

Also, It's great that companies are starting to open source their hardware designs, or parts of their frameworks. What would be fantastic is if they did the same with their verification infrastructure. Of-course it helps if that IP is actually usable by current open source tools. Google for example released a RISC-V program generator, but it's all in SystemVerilog and hence no good to the open source community.

Learning from Industry

- **Remember:** Industry has been through all of this before.
- Use the lessons from industry to decide where to invest in tools.
  - Methodologies, best practices etc.
- *Wilson Research Group Functional Verification Studies: 2012-2018*
  - Extremely good resource for looking at how commercial verification has changed over time.
  - Verification is not a solved problem: **As of 2018, 84% of surveyed FPGA design projects suffered a non-trivial bug escape into production**

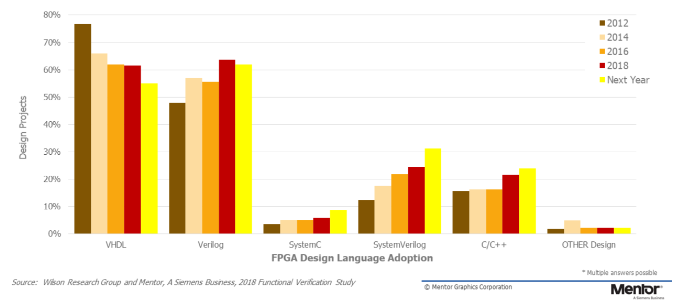
Learning from Industry
Remember: Industry has been through all of this before.
Use the lessons from industry to decide where to invest in tools.
Methodologies, best practices etc.
Wilson Research Group Functional Verification Studies: 2012-2018
Extremely good resource for looking at how commercial verification has changed over time.
Verification is not a solved problem: As of 2018, 84% of surveyed FPGA design projects suffered a non-trivial bug escape into production

The other thing that the open source community has going for it is that we essentially ”know what’s coming” in terms of many technical challenges. Industry has been there before, and there are hundreds of excellent articles, training materials and surveys on the topic.
This won’t be much of a surprise to many given how much overlap there is, even in this room, between people being in industry *and* pushing for open source tooling.
I did want to briefly look at one survey in particular, because it nicely underlines the needs and opportunities there are in building open source design verification tools and frameworks. The Wilson Research group functional verification studdies, run by Harry Wilson at Mentor, are a great resource for looking at trends and so on.

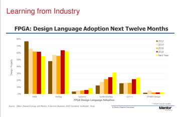
My favourite sentence from the most recent edition is there on the slide.

Learning from Industry

FPGA: Design Language Adoption Next Twelve Months



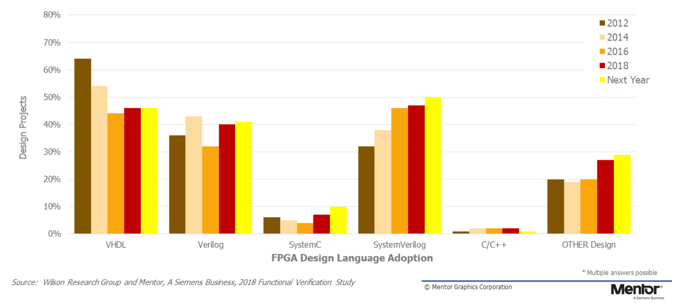
Source: Wilson Research Group and Mentor, A Siemens Business, 2018 Functional Verification Study



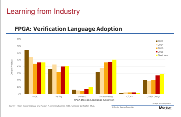
My cherry-picked insight from the most recent survey is to compare language adoption for FPGA development.
This graph shows the percentage of surveyed designs using each of the main HDLs, and crucially, which ones they plan to use *next* year.
It’s pretty clear here, VHDL and Verilog are still king, with SystemVerilog and C/C++/SystemC gradually gaining popularity. The *other* category is more of a rounding error for design.
However...

Learning from Industry

FPGA: Verification Language Adoption



Source: Wilson Research Group and Mentor, A Siemens Business, 2018 Functional Verification Study



It’s a very different story for verification.
That *other* category has a much larger share, and that share is increasing. In the report, it mentions Python specifically, but they note that all sorts of languages are being used.
In terms of which *libraries/methods* are being used for FPGAs specifically, UVM is the one to watch, it’s the only one which has consistently gained share in industry since 2014. Of-course, that’s not so useful to us in open source, so having alternatives like OSVM and UVVM is really important.
The point in picking out these graphs is to underline that the big shift in industry is toward standardised verification methodologies like UVM and SVA, even if the actual language used is ”other”.

Making sure we can match these capabilities in OSDA, for whichever design language you prefer, is going to be essential.

Miscellaneous Talking Points

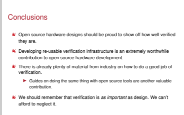
- Verifying security claims
- Human factors
  - More software oriented people moving into hardware development.
  - Agile development practices more suited to verification than to design?
  - Different perspectives on *product* development and testing.
- Commercial hardware development team structures look very different to open source governance / development structures.

Miscellaneous Talking Points
- Verifying security claims
- Human factors
  - More software oriented people moving into hardware development.
  - Agile development practices more suited to verification than to design?
  - Different perspectives on *product* development and testing.
- Commercial hardware development team structures look very different to open source governance / development structures.

I wanted to say much more than would reasonably fit in this talk, so in the spirit of provoking discussion, I figured I'd touch on just a few.
Open Source Hardware gets touted as a solution to hardware security flaws. I can get behind this to an extent, but security is predicated on functional correctness. If you can't show the latter, you don't have the former. Even then, security flaws can happen in perfectly functional designs. The problems arise because of system level interactions. If systems consist of open source components, we need to understand their interactions, not just components in isolation. This is an open problem.
There's a lot to be said for "agility" in hardware design. I actually think it is much better suited to hardware verification. What is a testbench if not a large piece of software? Here is where rapid iteration helps most, not in the RTL.
Finally, in industry you tend to have a separation of design and verification engineers to stop one repeating the other's mistakes. Open source communities don't organise that way, so it'll be interesting to see how big collaborative hardware design projects manage this.

Conclusions

- Open source hardware designs should be proud to show off how well verified they are.
- Developing re-usable verification infrastructure is an extremely worthwhile contribution to open source hardware development.
- There is already plenty of material from industry on how to do a good job of verification.
  - Guides on doing the same thing with open source tools are another valuable contribution.
- We should remember that verification is *as important* as design. We can't afford to neglect it.


Conclusions
- Open source hardware designs should be proud to show off how well verified they are.
- Developing re-usable verification infrastructure is an extremely worthwhile contribution to open source hardware development.
- There is already plenty of material from industry on how to do a good job of verification.
  - Guides on doing the same thing with open source tools are another valuable contribution.
- We should remember that verification is *as important* as design. We can't afford to neglect it.

So in conclusion:
OSDA is brilliant, and theres a-lot of momentum and enthusiastic people involved in making open source hardware designs a reality.
And, to keep this going, we need to pay more attention to either doing the verification, or making sure we describe and show off what verification has been done. Doing this will make open source designs more dependable, and make it easier for people to contribute.
We have lots of tools to help do this already, and there are good opportunities to contribute things like stimulus generation and functional coverage collection tools. Industry has had to do all of this before, and we can learn a lot about which approaches work best.
Thankyou for listening. I'm sure some of what I said needs more explanation or qualification, so please do ask questions now or come and talk to me afterwards.