```
alias opcode tb.dut.cpu.opcode
alias state  tb.dut.cpu.state
DECODE = 2
# cover   NAME    CONDITION           WORK
  cover   opcode  (state == DECODE)   values(opcode)
  cover   timer   True                posedge(timer_expired)
```

my version of covergroups and stuff

Coverage and python driven verification.

Ilia Greenblat
freelance VLSI Designer

August 11, 2020

Everybody have something to say about Verification, not many people actually like doing it. Is it because industry standard tools suck? Not many folks realise there are alternatives. And many more are prevented from using them. Thats is why i try to spread a sensible approach - Python driven verification.

And why do i care how You work? I dont want to affect Your job security - (better tools == less you).

Two reasons: Approaching new client - if the word is out - less justifying myself.

Second: If my RTL is verified by far away teams, and as result i get obfuscated log files (UVM) , huge waves and zero insight. Even tiny request to add something to the test bench reporting or checking is met with fear. I fully commiserate with them, but it is an easier job to trace Covid19 patient zero in Bnei-Brak, than get to the bottom of UVM reported problem.

That is why i take every opportunity to present the Python way to anyone who agreees to listen. The opposing arguments are pretty standard. And, please, seriously, give

me “pandemic” grade resistance. Not your everyyear influenza.

In the post below i try to present the popular opposition points, so next time You could do better. #verification #vlsi #python

“constraint solver is important” is usually the star member of the set.The trigger is my statement “there is no need for elaborate constraint solver, random standard package is enough”. My answer is a challenge : “Show me a real case where full fledged solver has real advantage over humble random module” I am still waiting. I am sure there exists one, alas never saw this mythical creature.

Were we to switch to Python, tons of infrastructure need to be developed.”

This is not true, only education part is somewhat true. But education is important in any flow. My experience is that good verificator can work with any good tools, and for bad one - nothing is good. UVM/SV tend to produce tons of files, classes and what not. Python is easy.

Shallowness of the test bench is important for our mental health. One of the hallmarks of UVM/SV is deeper inheritances.By aficionados it is considered an advantage. To me, debugging such a contraption is harder than corona epidemiological tracing of whom to blame for the outbreak. When something is broken in the test bench, i want to dig into as few layers of complexity as possible to trace the problem. Using python generators and decorators may appear as sophistication, but to me it is an ammonium nitrate in disguise.

Lack of fork/join, virtual multithreading comes next. This is a nice idea, but makes it harder to debug the test bench. In Cocotb they use generators to mimic this. In my VPI connection, I just use classes with state to do that. Which is to say, it is just an agreement, not an infrastructure. Having said that, You are welcome to try real or virtual multithreading in Python 3.

What about the coverage? Actually this is a valid point. Coverage (functional brand) ensures we don't leave any corner unchecked. Till now my answer was - You are welcome to use both line coverage and functional coverage built-in into the simulators. However it works only for commercial tools. I want to measure and accumulate coverage in free tools too.

How hard is to create coverage setup in python driven? The answer is **"few hours"** - provided You know what You have to do. **rtlCoverage.py** is my new module and soon in GitHub.

For now, it runs on VCD trace of the simulation. Why? because during development phase it is much easier and faster. And because it is Python, Same code will run in simulation too.

It employs my "vcd_python" app, You provide VCD waves and coverage definitions - and results are added to coverage aggregation data base. Data base is a big word for "python ready" files created. All is humanly readable and easily comprehensible. Aggregating to XLS or HTML is another easy task.

Cover definitions are in a text file, something like this:

```
alias opcode tb.dut.cpu.opcode # to shorten susequent references

alias state tb.dut.cpu.state

DECODE = 2 #constants

#  cover NAME  CONDITION     WORK

cover opcode (state == DECODE) values(opcode)     # will check all values of the opcode

cover timer True       posedge(timer_expired) # will count the number of timer expired
```

This is a small example - in hard to edit Linky-dinky format.

Usefull for project i am currently engaged. In the full module, there are bins and ranges and other goodies. And dont tell me "but this is not verilog format!" - . Verilog is for synthesiable RTL only. And dont tell me, this humble effort doesnt cover page 475 in the SystemVerilog standard. If it is on that page - nobody needs it - it is just a ruse to suppress competiton.

Look Ma, No inheritance, no decorators - just simple job done.

Published by

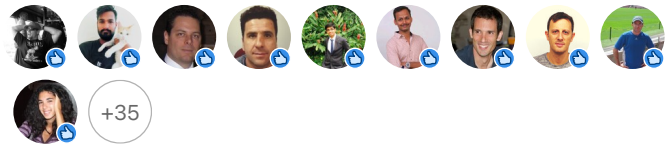
Ilia Greenblat 25 articles
freelance VLSI Designer
Published • 2y

Everybody have something to say about Verification, not many people actually like doing it. Is it because industry standard tools suck? Not many folks realise there are alternatives. And many more are prevented from using them. Thats is why i try to spread a sensible approach - Python driven verification. And why do i care how You work? I dont want to affect Your job security - (better tools == less you). Two reasons: Approaching new client - if the word is out - less justifying myself. Second: If my RTL is verified by far away teams, and as result i get obfuscated log files (UVM) , huge waves and zero insight. Even tiny request to add something to the test bench reporting or checking is met with fear. I fully commiserate with them, but it is an easier job to trace Covid19 patient zero in Bnei-Brak, than get to the bottom of UVM reported problem. That is why i take every opportunity to present the Python way to anyone who agreees to listen. The opposing arguments are pretty standard. And, please, seriously, give me “pandemic” grade resistance. Not your everyyear influenza. In the post below i try to present the popular opposition points, so next time You could do better.
#verification #vlsi #python

Like Comment Share

Martin Simlastik and 46 others 9 comments

Reactions



9 Comments

Most relevant

Add a comment…

Grigory Marderfeld • 1st 2y
ASiC/VLSI Digital Design Engineer

Mustapha Slimane-kadi

Like | Reply

Ohad Tsadik • 1st 2y
Senior Manager IC design (DV)

Hi ilia ,
I think It could be useful to integrate DPI-C(not just VPI) library to your

py libs .
It could potentially be useful for HW emulators based TBs , UVM-SV is not so popular for such TBs

Like | Reply · 3 Replies

Load previous replies

Ohad Tsadik • 1st
Senior Manager IC design (DV)
2y

Don't know if it worth your trouble . It depends on the customer.
For test beach acceleration on HW emulators , The DV env transactors use DPIc to communicate with the synthsizable BFMs . Typically (I think) people use C++ for such platforms .
So python might be a nicer alternative . Could also be interesting

Like | Reply

Ilia Greenblat • You
freelance VLSI Designer
2y

Ohad Tsadik in a nice company, moving verification to python , was being justified by exactly this argument : use almost the same code in Cadence, Verilator, Palladium, FPGA emulation and later in bring-up board. But there, i was not aware of DPI-C. let me check that with Verilator first.

Like | Reply

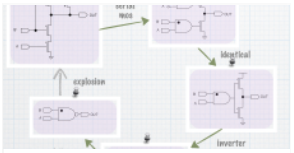
Load more comments

Ilia Greenblat
freelance VLSI Designer

More from Ilia Greenblat

Verification is easy, debug is hard.
Ilia Greenblat on LinkedIn

His post got a lot of comments, here are my re-comments.
Ilia Greenblat on LinkedIn



Recognize your spice?!
Ilia Greenblat on LinkedIn

See all 25 articles