The Vtool Friday Tech Club

Verification Convergence Phase
Principles and Practical Hints

Hagai Arbel
October 2020
Agenda

- What is verification convergence stage
- The 80–20 principle in verification
- How to converge faster
  - Code robustness
  - Controlling generation
  - Functional coverage holes hunting
  - Scripting and regressions
  - Servers and licenses resources
  - Profiling
  - Proper diagnostics using Cogita
- Summary
“I’m Almost Done”
A typical verification cycle over time

- Either functional coverage was defined in the beginning or we have other ways to measure convergence.
- 80% of the coverage is achieved in 20% of the time.
A typical verification cycle over time

- Phase 1 – Writing VE, integrating DUT, basic tests
- Phase 2 – Massive random, most RTL bugs found.
A typical verification cycle over time

- Phase 1 – Writing VE, integrating DUT, basic tests
- Phase 2 – Massive random, most RTL bugs found.
- Phase 3 – Problems!
What we want

- Predictable Phase 3
- Short Phase 3
- Happy Phase 3
Faster Convergence
Robust Code

- Bad code you write early on, will hunt you back at Phase 3.
- Do not hesitate to rewrite/improve the code at first signs of spaghetti, long debug or repeating bugs.
- Decide on a messaging system and embed it in the code from day 1.
Controlling Generation

- You must have a good way to steer random toward your desired scenario.
- Your sequence lib should be modular so that you are creating constraints quickly and apply them to your tests.
- Think of it as a control panel of a sound system – You have convenient ways to adjust all kind of attributes, not just the volume.
Coverage Holes Hunting

• When you try to fill this last 0.1% and these stubborn coverage holes:
  • Running more tests is hardly the right solution.
  • Instead:
    • Check the distribution of coverage hits – Do you have good chances to hit what is missing? If not, adjust weights
    • Write directed tests that will hit immediately.
    • Consider well whether this item / cross / range - is really needed.
Scripting and Regressions

- Make sure your regression script can run tests in parallel and that there are enough licenses and servers.
- Run regressions with no waves and no verbosity. Each of these can make your tests 10X slower.
- Do not compile the design in every run – the regression script should compile once and use the same executable.
- Are the tests efficient? Do you wait in each test for PLL lock? If so, force it in most tests.
- Separate between regressions that are for debug and regression that are for coverage. In debug regressions, run without coverage – 10X faster.
Servers

- Analyze servers performance to check why your regressions are slow.
- Use `top` or `htop` command (`htop` is clearer but sometimes not available)
htop servers performance hints

• Check the total available RAM and weather swap is in use. Using swap (use disk as RAM) means there is not enough RAM – there will be a slow down.

• Check that each simulation job is closed to 100% CPU – The load balancer should not use a core for more than one job.

• Shift-m will sort by memory, not by CPU

• For more information, take a look at this recording or ask Bukurecki

  • https://us02web.zoom.us/rec/play/ElUQJ22Ni2wa84PKzbdVibaWneAyA1kqvicdAeoqt3CtgBxAus2Tfr0eYEEHv2eCUQi2te3Q8JXnQWk.E5M-7_Ou1NiJYhyu?autoplay=true&startTime=1599227287000
Profiling

- Your code might be extremely inefficient. Common reasons are:
  - Much faster/slower clock in Verilog – check if we can run most simulations without it (i.e. PLL)
  - Over sampling of coverage
  - Huge Queues
  - Etc.
- Use the simulator profiling command and look for BIG things.
Faster Debug and Diagnostics
What are diagnostic

- In Phase 3, you need a fast way to triage failures and to understand what happened in a massive random test – BEFORE YOU START THE DEBUG.
  - How? Use Cogita
- You need a quick way to check the distribution of attributes in your tests, in order to merge faster into coverage holes hunting.
  - How? Use Cogita
- You need to avoid
Guys, I created a Player CFG in Cogita that screens Errors and Warnings by module. Among other things, it can help you get a quick understanding regarding your errors.

Now we will do reload every minute and see how the picture is changed.
Summary

• We want to converge faster of the “last 10%”
• For that, there are several ways and methodologies to be applied.
• Know them, use them.
Questions?
Thank you!