Why Verify Components?

The concept of TDD comes from the software world’s Agile approach, sometimes referred to as RED-GREEN-REFACTOR.

Why Refactor Code?

The focus of the first two steps of the TDD flow is creating a passing test case. These steps may produce code that is not readable, straight forward or even commented. Common examples of issues fixed in the refactoring stage include breaking down large classes, removing duplicate code, or splitting a method that is simply too long. Refactoring these sections will make the code easier to work with by making it not only easier to read, but also resolving hidden or dormant issues that may exist within the code.

What is a Unit Test?

A Unit is the smallest piece of your code to be tested. In design, it would be the modules that make up the design. In UVM, it would be individual verification components. So unit testing is just testing individual pieces of a design or verification component.

Using SVUnit - Overview

SVUnit is a simple verification framework is intended for design and verification engineers. It can be utilized to help create more complicated code with fewer bugs, making it possible to create cleaner code. TDD has been lacking in current approaches. It gives structure to the art of coding, making it possible to create high quality code that is bug free and easy to read.

Writing code is challenging and it’s not getting easier; not only from a technical standpoint but because it’s more of an art than engineers would care to admit. Engineers are thrown into the fire, asked to complete a verification task with little to no planning or documentation – forced to peek at the design code in order to determine what should be tested. Everyone wants to create high quality code that is bug free and easy to read. If that is the case, the tests should not just test that the design behaves as coded; instead tests should target desired behavior. Agile software techniques such as TDD and Unit Testing offer a different approach for developing more robust verification environments.

Benefits of TDD and Unit Testing

- Applying TDD helps to limit scope creep. By creating an upfront list of exactly what the RTL and corresponding verification components are supposed to do and creating a test for each feature, unplanned or unnecessary tests can be avoided.
- Having these little tests that all pass allows changes in the code to be made with assurances that a change doesn’t break any previous functionality.
- Due to the bottom up approach to testing, you know your building blocks are in great shape so integration testing becomes easier.
- As engineers, we traditionally balk at documentation. It’s sort of in our DNA. Unit tests that are created correspond to the critical parts of the components and therefore are a sort of living document that can be extracted and used to create more formal documentation later.

In order to verify UVM components, unit (uvm_test_start) must be called. This will kick off the running of the UVM phases. SVUnit has run and been tested on the big four simulators and comes with scripts and helper functions to help you get started.

Benefits of TDD and Unit Testing

- Applying TDD helps to limit scope creep. By creating an upfront list of exactly what the RTL and corresponding verification components are supposed to do and creating a test for each feature, unplanned or unnecessary tests can be avoided.
- Having these little tests that all pass allows changes in the code to be made with assurances that a change doesn’t break any previous functionality.
- Due to the bottom up approach to testing, you know your building blocks are in great shape so integration testing becomes easier.
- As engineers, we traditionally balk at documentation. It’s sort of in our DNA. Unit tests that are created correspond to the critical parts of the components and therefore are a sort of living document that can be extracted and used to create more formal documentation later.

These unit tests are small and therefore run faster than the traditional full random simulation.

With SVUnit test cases packaged with the VIP, you can make small changes in the agent and make sure they don’t break the overall functionality of the agent.

TDD is very systematic and structured. At first, the rigidity of it feels constraining and time consuming. This process makes not only cleaner code but a more productive coder.

Real World Results

In order to test SVUnit, an experiment was needed. What better way to test SVUnit than on the code most environments were created with? UVM-Utest is an open-source initiative that demonstrates the value of unit testing relative to an industry standard code library. In UVM-Utest, unit test suites were written for several core components of UVM. The intent was to rigorously verify the functionality of each component in isolation, an approach uncommon in hardware verification.

UVM-Utest core UVM components:

- uvm_object
- uvm_mib
- uvm_printer

2 engineers * 6 weeks = 500 UVM unit tests

500 UVM unit tests * ~14 seconds (in simulation) =

10 UVM Defects!

Conclusion

Writing code is challenging and it’s not getting easier; not only from a technical standpoint but because it’s more of an art than engineers would care to admit. Engineers are thrown into the fire, asked to complete a verification task with little to no planning or documentation – forced to peek at the design code in order to determine what should be tested. Everyone wants to create high quality code that is bug free and easy to read. If that is the case, the tests should not just test that the design behaves as coded; instead tests should target desired behavior. Agile software techniques such as TDD and Unit Testing offer a different approach for developing more robust verification environments.

To be fair, TDD and thereby SVUnit may not be the best solution for every project, but they both bring along a very systematic approach to developing verification environments that is sorely lacking in current approaches. It gives structure to the art of coding, making it possible to create cleaner code. TDD has been in use in the software world for some time now and it continues to be utilized to help create more complicated code with fewer bugs, and on tighter schedules.

Acknowledgements

We would like to thank both Superion Technology and XtremeEDA for giving us the opportunity to work on this. We’d like to thank all those of you using SVUnit.

Why Verify Your Verification Components?

What is TDD?

The concept of TDD comes from the software world’s Agile Development. Unlike in the waterfall process, with TDD flow, the verification stage is before the implementation stage. The idea is to first determine a feature of the code (in this case a verification component) and then write a test to test that feature. This test must fail. Only after the testcase fails, is the verification component feature written that the test was checking.

Using SVUnit - Overview

SVUnit is a simple verification framework for SystemVerilog and UVM. This simple verification framework is intended for design and verification engineers writing and running tests against Verilog modules, classes or interfaces, as well as against SystemVerilog components and objects. SVUnit uses a 3-level hierarchical structure. At the lowest level, SVUnit is built from a unit test template which contains a simple unit test along with the code. The individual pieces would go to Unit Test, grouping pieces to make a TestSuite would be like an agent and a whole protocol tester would be a testrunner. Let’s go back to our earlier idea of creating an agent. The components of an agent are the driver, the monitor, the sequence item and the interface. Each one of these would have their own unit test template. These would be grouped together in the next level of hierarchy where there is a test suite.