Building a better VHDL testing environment

Joren Guillaume
Supervisors: prof. ir. Luc Colman, dr. ir. Hendrik Eeckhaut, ir. ing. Lieven Lemiengre
Ghent University

Abstract– **The VHSIC Hardware Description Language (VHDL) is a language used industry-wide for developing digital electronics. This paper explores the possibility of using software development practices in hardware development that uses VHDL. Borrowing elements from test-driven development, specifically unit testing, a test bench is split into parts that are executed sequentially. To achieve this, a script is written in Python that uses the ModelSim compiler/simulator. This script is then automatically called by a continuous integration server that captures and processes the results.**

Index terms–**VHDL, TDD, unit testing, continuous integration, test benches**

I. Introduction

Developing digital hardware is and will remain a global industry of undeniable importance. VHDL is one of the important hardware design languages, having been used in this industry for many decades. Although improvements have been made to the language itself, the development practices have been lagging behind the software world for several years. As with all production, any improvements that can be done to the development process with appropriate cost should be a welcome sight. However, the industry is slow to adapt or convert to more up-to-date processes. This paper aims to investigate a number of software development practices and build a working framework in which they can be used for hardware design [1–3].

A. Unit testing

In a testing environment, a unit is the most basic behaviour of a piece of code that can be tested. It is called a unit because there is nothing smaller. Unit testing is the practice of taking these small parts and writing tests for them and only them, so that all tests are performed on units. A test only performs on a single unit of code; if the test uses parts of code outside of the unit, it is not a unit test. One of the biggest advantages of this testing method is the precise debugging information that is given should a test fail. After the pinpointing, only a small part of the code needs to be examined, making it very efficient [4, 5].

B. Continuous integration

Continuous Integration (CI) is the practice of automatically and very rapidly integrating code updates into a build. It encompasses a number of practices such as:

- Revision control
- Automated building
- Automated testing

CI aims to prevent the trouble in development that arises when developers wait too long to integrate their edits into the main build. It promotes early and rapid integration by automating these steps and downloading all the latest code from a repository. This is assuming that the repository contains the newest code and is updated on a frequent basis. After integrating and building the code, it should run the tests on this code and read the results. If all of these steps are automated and triggered on a timed basis, developers should have a steady stream of progress reports from the test results available [6, 7].

II. Methods

Transferring the concepts from software to hardware required a number of concessions that would otherwise make implementation impossible.

A. Unit testing in VHDL

A unit test was defined as a test or group of tests that could be run at earliest convenience. This sometimes needed inputs to be set and the simulation to be run up until the required point. All test groups were identified manually with markers in the original test bench code.

B. Python script

A script was written in Python to combine all of the aspects investigated. This script is called from the command-line or a shell and takes a number of arguments to determine method of separation, source test benches, output location and VHDL version. It read the test benches and separated them into new test benches. These test benches were then compiled and simulated using ModelSim and condensed reports of the output were made in the JUnit XML specification.

C. Bitvis utility library

A utility library made by Bitvis was used to speed up development. Functions in this library made it possible to quickly produce tests with uniform output. The library's default output was slightly modified to better integrate with the Python script's report generation.

D. Hudson-CI

Continuous integration was implemented using the Hudson-CI program. A job was made to download the code from an on-line Git repository. The job then executed the Python script and captured its reports. Hudson provided built-in support for JUnit XML report reading and automatically provided statistics on failure and success.

III. Results

Separating the tests at earliest convenience provided the benefit of not halting tests when a critical error occurred. Instead, only that single group of tests would return as failed to run. The CI server gave clear indications which tests had passed and failed, but was not able to correctly read the nano- and picosecond timing a hardware simulation requires. Furthermore, the script was limited when it came to the complexity of the test benches it could read, putting boundaries on which projects could be tested.

IV. Conclusion

It was proven that software techniques could be modified to work with a hardware development environment, but that more detailed and thorough work is required before it reaches full potential.

References

[1] Roger Lipsett, Erich Marschner, and Moe Shahdad. "VHDL - The Language". In: *IEEE Design & Test of Computers* 3.2 (Apr. 1986), pp. 28–41. DOI: 10.1109/mdt.1986.294900.

[2] R.D. Acosta, S.P. Smith, and J. Larson. "Mixed-mode simulation of compiled VHDL programs". In: *1989 IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*. IEEE Comput. Soc. Press, 1989. DOI: 10.1109/iccad.1989.76930.

[3] T.E. Dillinger et al. "A logic synthesis system for VHDL design descriptions". In: *1989 IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*. IEEE Comput. Soc. Press, 1989. DOI: 10.1109/iccad.1989.76906.

[4] Don Wells. *Code the Unit Test First*. 2000. URL: http://www.extremeprogramming.org/rules/testfirst.html (visited on 03/20/2014).

[5] Martin Fowler. *UnitTest*. 2014. URL: http://martinfowler.com/bliki/UnitTest.html (visited on 06/30/2014).

[6] Fazreil Amreen Abdul and Mensely Cheah Siow Fhang. "Implementing Continuous Integration towards rapid application development". In: *2012 International Conference on Innovation Management and Technology Research*. IEEE, May 2012. DOI: 10.1109/icimtr.2012.6236372.

[7] Sean Stolberg. "Enabling Agile Testing through Continuous Integration". In: *2009 Agile Conference*. IEEE, Aug. 2009. DOI: 10.1109/agile.2009.16.