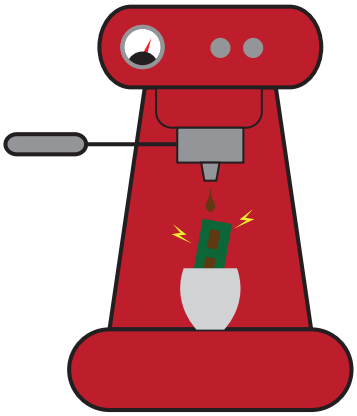

Verification of digital circuits using Java

02125 Bachelor Project

by Rasmus Wiuff **s163977**
s163977@dtu.dk

Supervisor: Martin Schoeberl
masca@dtu.dk

Draft

Abstract

Add abstract

TODO LIST

CONTENTS

1 Summary

This section summarises the sections of the report.
Section 2 summises UVM and some chosen chip verification frameworks, pivotting towards the motivation for, and introduction of, this project.
Section 3 breaks down the project into challenges and analyses solutions to these challenges.
Section 4 Section 5 Section 6 Section 7

Add summary of sections

2 Introduction

In the world of designing integrated systems on chips, it is crucial to describe these in some hardware description language in order to test designs before actual manufacturing. Part of testing these designs include writing tests in some framework supporting some abstraction and then simulate the chip using the applied framework, a process known as *verification*. One verifies the *Device Under Test* or DUT. One of the most common methodologies is *Universal Verification Methodology*, or simply UVM.

2.1. The UVM methodology

Using UVM, one have to built testbench components. These include e.g. drivers for converting tests into proper DUT stimulus, monitors for reading the state of the DUT and scoreboards for comparing expected behavior to actual behavior, etc. UVM has the great benefit, that once these components have been defined, they can be reused within the scope of some system of designs. This means a high overhead, with high reusability.

2.2. Current verification tools

In todays landscape a myriad of verification tools exists. Projects like Chisel[1] and it's forked project SpinalHDL[2] discards the traditional hardware description languages and testbenches altogether and implements their own, and pyuvm[3] takes UVM into python using cocotb[4] as a backend driver. These tools do however have some problems. Chisel and SpinalHDL requires developers to learn a new description language, which means the communities of developers are small, increasing slowly. pyuvm uses Python which itself is an intepreted language, meaning its runtime ressource requirements do not scale well on large projects.

2.3. The case against UVM and the mighty ABV

Chip verification is inherently done by hardware designers and engineers and UVM is inherently created by and for hardware designers. When software engineers verify their software, they use unit tests and assertions along with formal proofs. In recent years these approaches have been adopted by hardware designers. *Assertion Based Verification*, or ABV, along with formal verification is increasingly being applied to complex computing chip design verification and increases performance metrics over classic UVM[2]. Key impacts from this paper is summarised on Table 1.

Table 1: *Key impact of formal verification adoption over UVM[2]*

Verification cycles reduced by 25-30%
Pre-silicon bug detection rates improved by 20%
Security vulnerability detection increased by 40%

SystemVerilog Assertions, or SVA has been defined in IEEE 1800-2023[1, Chapter 16] meaning ABV is already a part of the SystemVerilog syntax.

2.4. The goal of this project

The goal of this project is a framework written in a strongly typed language, supporting SystemVerilog and core ideas from ABV, thus making it easy for designers to write their designs in SystemVerilog and use a well known language to implement assertion based tests.
Introducing *SteelBrew*, the chip verification framework written in Java.

[1] On Github: github.com/chipsalliance/chisel
[2] On Github: github.com/SpinalHDL/SpinalHDL
[3] On Github: github.com/pyuvm/pyuvm
[4] cocotb on the web: www.cocotb.org/

3 Problem specification and analysis

As mentioned in Section 2 this project aims to implement a testing framework for SystemVerilog designs, using core ideas of Assertion Based Verification, written in Java. This poses the following challenges:

- **Simulation driver:** The framework needs to communicate with some simulation driver.
- **Peek-poke-step:** Signal manipulation needs to be implemented to set up and carry out assertion of behavior.
- **Assertion-logic:** ABV logic has to be implemented in the framework.
- **Test translation:** The framework has to translate test logic from Java to SystemVerilog and create a testbench for test simulation.
- **Coroutines/concurrency:** As assertions can be time-invariant, it makes sense to implement concurrency or coroutines to efficiently execute the simulation.

3.1. Simulation driver

Driving the simulations is not part of the scope of this project. Instead the simulations are run by a third-party tool using Java to invoke the tool and listen for the results.

3.2. Peek-poke-step

Using the Java BigInt class, some logic for manipulating bits needs to be implemented in order to set up tests.

3.3. Assertion-logic

There are some assertion directives of interest:
1. Assert which raises an exception if some property does not hold.
2. Cover which moniters the coverage of some property.

These are the minimum that should be implemented. Some object for a test should be created and react if the driver raises an exception pertaining to the used directive. The logic in the framework needs to adhere to the logic stated in the SVA documentation.

3.4. Test translation

3.5. Coroutines/concurrency

4 Design

5 Implementation

Document implementation

6 Testing

Insert and describe tests on framework

7 Conclusion

Write up conclusion

References

[1] "IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language". In: *IEEE Std 1800-2023 (Revision of IEEE Std 1800-2017)* (Feb. 2024), pp. 1–1354. DOI: 10.1109/IEEESTD.2024.10458102. URL: https://ieeexplore.ieee.org/document/10458102.

[2] Kaushik Velapa Reddy. "Formal Verification with ABV : A Superior Alternative to UVM for Complex Computing Chips". In: *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* 10.6 (Nov. 5, 2024). Number: 6, pp. 90–98. ISSN: 2456-3307. DOI: 10.32628/CSEIT24106157. URL: https://ijsrcseit.com/index.php/home/article/view/CSEIT24106157.

List of Figures

List of Tables