cucumber-verilog: Behavior Driven Development for Circuit Design and Verification

Melanie Diepenbeck[1] Mathias Soeken[1,2] Ulrich Kühne[1] Rolf Drechsler[1,2]
[1]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
{diepenbeck, msoeken, ulrichk, drechsle}@informatik.uni-bremen.de

***Abstract*—When designing hardware one usually applies a top-down approach in which starting from a natural language specification a design is implemented and afterwards tested and verified for correctness. In contrast, software development is pushed towards agile techniques such as Test Driven Development (TDD, [1]), where tests play a central role in driving the implementation. Behavior Driven Development (BDD, [2]) extends TDD by using natural language style scenarios to describe the tests. Essentially, in both techniques testing and implementation is interleaved: first, test cases are written, and second, the implementation is extended to satisfy them.**

Since nowadays 70% of the the effort to design hardware systems is spent on test and verification, these phases should receive more attention and be applied in early design phases. We present a BDD tool tailored for the Verilog hardware description language which enables a new flow for hardware design, test, and verification. BDD acceptence tests are readily given by means of the natural language specification. Assigning test code to their sentences yields a testbench which serves as a starting point for the implementation. In the same time, the natural language scenarios form a test documentation that is easily accessible also to non-experts. Furthermore, our tool allows for the generalization of test cases to properties suitable for formal verification. As properties are typically more difficult to formalize than test cases, our approach facilitates the access to formal verification.

In our demonstration, we will show how to implement hardware designs using our BDD tool and how properties are generalized from test cases which can then be verified by a model checker automatically.

Our tool implements a new flow for circuit design and verification. The flow consists of two main ideas, presented in [3], (1) a customized BDD flow that is suitable for circuit design and (2) generated properties that can be used for formal verification which are generalized from written test code. The main stages of our flow are depicted in Fig. 1.

Stage 1: Acceptance Tests

In Stage 1, the features of the hardware (or hardware components) are described by *acceptance tests* in natural language using the *Given-When-Then* sentence structure. Each *Given-When-Then* sentence is called a *step*. Although this stage does not differ from the conventional BDD flow, it is the most obvious difference between a conventional circuit design flow and our proposed flow. These acceptance tests can now be used to create a circuit design using the hardware description language Verilog.

Stage 2: BDD for Circuit Design

The actual implementation of the system takes place in Stage 2. In an agile manner, the step definitions, the testbench, and the implementation are developed iteratively. A step definition contains fragments of test code that describe the behavior of a single step or operation in a scenario. A testbench that drives the design under test is generated automatically and is filled with the essential test code fragments that belong to a specific acceptance test. During the BDD process each addition or modification immediately causes the test cases to be executed with the objective to make the acceptance

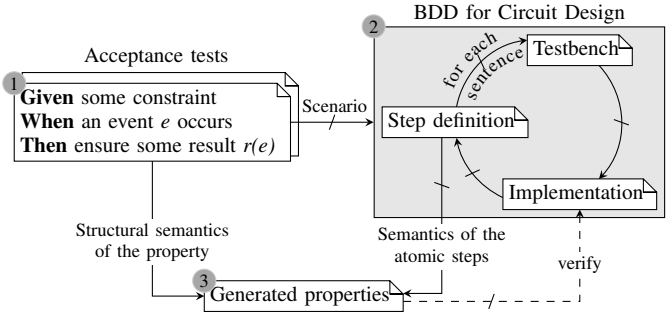

Fig. 1. Proposed Flow

tests pass. If the execution fails, code is either missing or faulty and additional iterations are necessary. In the beginning all tests will fail, and the designer needs to add first implementation details such as interfaces and input/output definitions. In the entailing interactive design process, more and more behavior is added to the implementation, driven by the acceptance tests, until eventualy all tests are passing.

Stage 3: Generalization of Test Code

Once all acceptance tests have passed, an implementation is available which fulfills all requirements that have initially been specified. However, due to their simulative nature the acceptance tests do not guarantee a bug-free design, since they cannot exhaustively cover all possible inputs and states of a larger circuit. As a result, only a subset of all possible test patterns is applied and bugs may be missed.

In order to improve the design quality, acceptance tests are *generalized* by automatically generating formal properties in the Property Specification Language (PSL, [4]). The given scenarios can then be verified thoroughly using existing state-of-the art algorithms for formal verification.

To obtain the PSL properties, the verification intent of a scenario, given by the *Given-When-Then* sentence structure is mapped to an implication property. The statements of the antecedent and consequent are created using the test code of the step definitions and the symbolic relations (italic arguments in Step 1 of Fig. 1) of the acceptance tests.

References

[1] K. Beck, *Test Driven Development. By Example*. Amsterdam: Addison-Wesley Longman, Nov. 2003.
[2] M. Wynne and A. Hellesøy, *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. The Pragmatic Bookshelf, Jan. 2012.
[3] M. Diepenbeck, M. Soeken, D. Grosse, and R. Drechsler, "Behavior driven development for circuit design and verification," in *Int'l Workshop on High Level Design Validation and Test Workshop (HLDVT)*, Nov 2012.
[4] *Accellera Property Specification Language Reference Manual, version 1.1*, http://www.pslsugar.org, 2005.