Automated testbench generation from digital timing diagrams

Winand Seldeslachts

Supervisor(s): Prof. Luc Colman, Ir. Hendrik Eekhout, Ir. Lieven Lemiengre

***Abstract*: This report introduces a way to test VHDL designs in an easy and unambiguous way. A user describes the input signals and expected output signals for a specific design in a waveJSON [1] formatted file, the source file. WaveDrom [2] converts this file into so called documentation wave traces. These are the visual representation of what is described in the source file. At the same time the source file is converted into a VHDL test that will drive the design with the desired input signals and check the output signals. The test is simulated using the VUnit [7] framework. Simulating this test will result in so called simulation wave traces. These are compared to the documentation wave traces. The discrepancies are shown to the user in a clear way.**

***Keywords*: VHDL, WaveDrom, VUnit, automated verification, wave trace analysis**

I. Introduction

Verifying a design is an important step in designing a hardware system. Verification allows the designer to find and understand design flaws that compilers do not check. The traditional verification cycle for a hardware design is illustrated below.

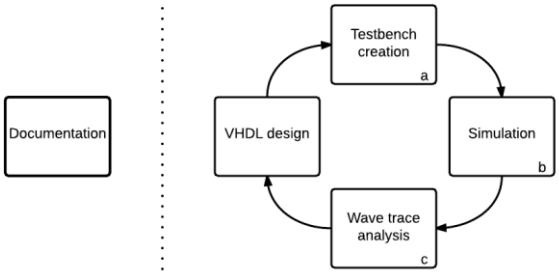

Figure 1: traditional verification cycle.

When a design has to be verified, **a)** A testbench containing one or more tests has to be created, **b)** These tests have to be simulated and **c)** the simulation results have to be analyzed. This analysis is done by comparing the resulting wave traces to the ones specified in the documentation.

This report introduces a way to streamline the verification of VHDL designs using a source file; a waveJSON formatted file. WaveJSON is an application of the JSON format. The purpose of WaveJSON is to provide a compact exchange format for digital timing diagrams. The software that has been built uses this file as a base for optimizing several steps in the verification process. The improved verification cycle is shown in figure 2.

W. Seldeslachts is an electronics engineering student, Ghent University (UGent), Gent, Belgium. E-mail: winand.seldeslachts@ugent.be

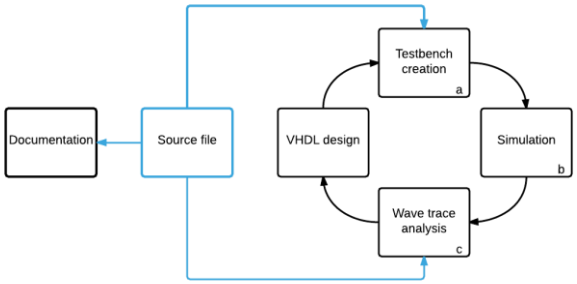

Figure 2: Improved verification cycle.

II. Problem Analysis

The main issue with the traditional verification cycle is that test (contained in a testbench) are often outdated and not self-checking. Self-checking tests are tests that will automatically fail if the design does not meet the required specifications. This is opposed to tests where input signals are driven and the output results shown, but any conclusion regarding passing or failing is left to the designer, who has to visually check for compliance with the documentation. Using these outdated testbenches often results in staring contest between the designer and a bunch of wave traces.

Even if tests are self-checking, there is often no way to easily see where the error fails.

These problems lead to the following research questions, which will be answered in this paper:

Can we design a system that will streamline the process of design validation?

Can we create self-checking testbenches with this system?

Can we optimize wave trace analysis with this system?

Does this system provide any benefits over existing verification methods?

III. Functional Analysis

In an attempt to answer these questions, a software tool was developed using the Python programming language. This tool has two major tasks: build self-checking testbenches and analyze the wave traces. These are two steps in the verification cycle. The third step, simulation is handled by an external simulation engine. To do this, the VUnit framework is used. The VUnit framework extends the functionality of VHDL and makes it possible to start a simulator with a Python command. The function of the tool is illustrated in figure 3.

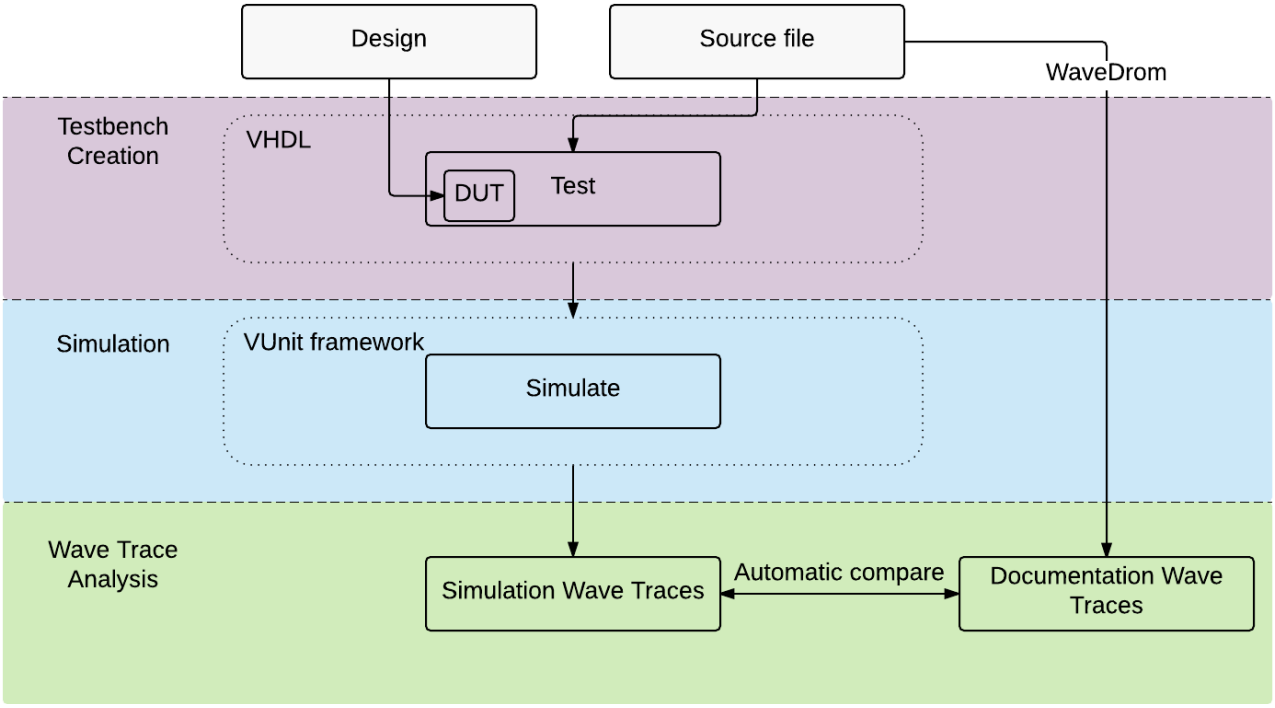

Figure 3: overview.

The testbenches are built according the VUnit requirements based on a template. This template defines the basic structure of the testbench. Using string manipulation the tool adds test specific code to this testbench based on the description in the source file. The result is a self-checking VHDL testbench for the targeted design.

At the same time the documentation wave traces are generated by WaveDrom. WaveDrom is an open source engine for generating wave traces from waveJSON files. It uses the description in the source file to visualize how the design should behave.

Then the test is simulated using the VUnit framework. This results in the simulation wave traces. These are compared to the documentation wave traces. If they are equal, the test has succeeded. If they are not, the difference is shown.

In short the software will convert a source file describing a feature of the design into a VHDL test, simulate it, and then output the comparison of simulation and documentation wave traces to the users as shown in figure 4.

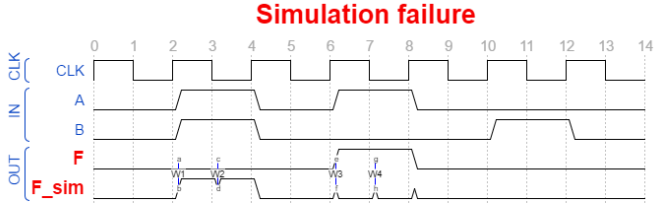

Figure 4: Output example.

This all is controlled by the graphical user interface (GUI) shown in figure 5.

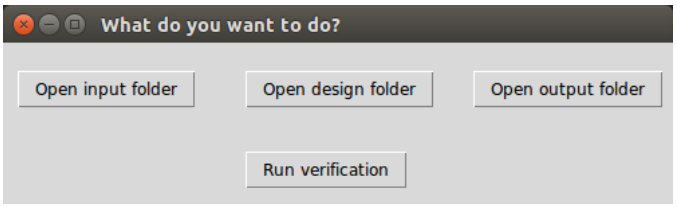

Figure 5: GUI controlling the tool.

The user can open the source folder using the first button to add all source files he wants to use, then he can do the same for every design that has to be tested by clicking the second button. Clicking the “run verification” button runs the tool and places all comparison files in the result folder. These comparison files are also waveJSON files that can be opened using WaveDrom.

IV. Technical Analysis

A. *Testbench creation*

1) *Testbench specification*

This section corresponds to the purple part of figure 3. To be able to create testbenches, we first have to define how they have to look. This depends on which framework is used in the simulation step (blue part of figure 3). In our case this is the VUnit framework, because it is one of the most advanced frameworks available. It’s also an open source framework, which makes it easy to work with. Other open source frameworks include: CoCoTB [3], SVunit [4], OSVVM [5] and UVVM [6].

VUnit testbenches are very similar to traditional testbenches, which means the learning curve is not very steep.

2) Source file

The waveJSON format was chosen because it was also the source format for WaveDrom, which is used to visualize wave traces and because it is an easily processable format. However, a standard WaveDrom file did not hold enough information to directly create a testbench from it. To solve this problem, some extra elements were added to it. This did not interfere with WaveDrom's ability to generate wave traces from the source file. The elements added to the standard waveJSON format include:

- The exact name of the unit under test
- The VHDL signal type for every signal
- The direction of the signal (input or output)

3) Conversion script

Now that both the source file and the testbench format are specified, the next step is to build a script that will convert one format into another. This is done using Python, because it is a flexible and easy to use programming language. It also has a large and active community, which means many examples are available online.

B. *Wave trace analysis*

This section corresponds to the green part of figure 3. After simulation a test will either pass or fail. This functionality is integrated in VUnit. The errors that are logged during the simulation are used to generate a new waveJSON output file that will show all the errors. The WaveDrom generated image from one of these files is shown in figure 4.

V. Example

In this section an example test is shown. An AND-gate design is tested for compliance with the specification

$$A \,\&\, B = F \tag{1}$$

The source file is shown below.

```
{"name": "andGate_timed", "test" : "andgate_full",
 "description": "test all possible inputs for an AND-gate",
 "signal": [
  ["CLK",
   {"name": "CLK", "wave": "p......", "type":"std_logic"}],
  ["IN",
   {"name": "A", "wave": "01010..", "type": "std_logic"},
   {"name": "B", "wave": "0..1010", "type": "std_logic"}],
  ["OUT",
   {"name": "F", "wave": "0..10..", "type": "std_logic"}]]
}
```

Figure 6: Source file for a full AND-gate test.

From this source file the documentation wave traces can be generated using WaveDrom. They are shown in figure 7.

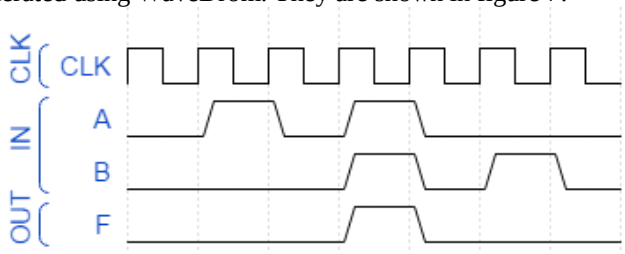

Figure 7: Documentation wave traces generated from the code in figure 6.

Then, a testbench is created from this same file. After simulation the logged errors are checked. In this case no errors were logged and the simulation succeeded. The AND-gate design complies with the specifications described in the source file. The result file will be equal to the source file, because the simulation wave traces are equal to the documentation wave traces. This file can be converted into an image by WaveDrom. This results in image 8.

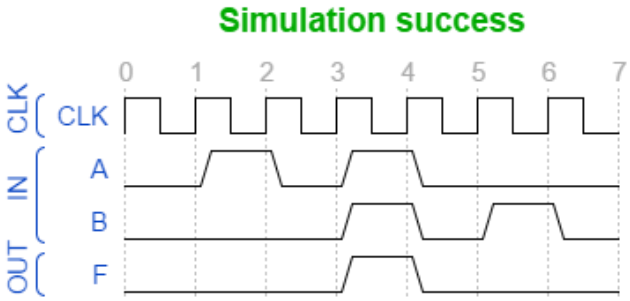

Figure 8: AND-gate test result.

VI. Benefits and Shortcomings

The tool has many benefits over the traditional validation system. For example, User input has been reduced to a single file: instead of the user needing to create a testbench, analyze the simulation wave traces and document the design, all this is now done automatically.

It does however also have some shortcomings. Some of the main problems are discussed here. First, because it has many dependencies and there is no installer available, it is not very easy to install. Installation is also only possible on a Linux device. The software was developed on Ubuntu 14.04 LTS. Secondly, the system is not suited for large tests, because either the source files or the resulting wave traces would become too large to clearly display. Then again, this is a problem other system would also have to deal with. Large tests are tests where the amount of input and output ports of a design exceeds the maximum showable amount or where the amount of clock cycles in the wave traces exceeds the maximum showable amount.

VII. Conclusion

This section will try to provide an answer to the questions in chapter II.

A. *Self-checking testbenches and documentation wave traces*

As the source file is a waveJSON formatted file suited for use with WaveDrom, it is possible to directly generate wave traces from it. As the desired functionality is fully described by the source file, these wave traces can be used to document the design. This provides half of the functionality required to answer the first sub question. The other half was made possible by adding some extra fields to the original WaveDrom file. By adding all the required information for creating self-checking testbenches. Using a conversion Python script and a template file it is possible to build fully fledged self-checking testbenches.

The resulting system does indeed fit the requirements. This means that it is indeed possible to design a system that can

create self-checking testbenches and documentation wave traces based on the same source.

B. Optimizing wave trace analysis

As the testbenches generated by the tool are self-checking, analyzing the tool becomes a lot easier. If a test succeeds, that means the design complies with the specification described in the source file. If a test fails however, it does not comply. In this case wave trace analysis comes down to easily comparing the simulation wave traces to the documentation wave traces. To do this, the tool creates a new waveJSON file where the differences between the two are described. This file can be converted by WaveDrom to visually show them.

C. Streamlining the process of validation

The tool facilitates the process of testbench creation and wave trace analysis, while at the same time minimizing user input and creating documentation wave traces to help document the design. Considering the limitations, we can say this tool indeed streamlines the process of validation up to a certain point. The software designed in this project can act as a proof of concept and lays the foundation for further improvement.

D. Benefits over existing verification frameworks

As the tool is unique in its kind, it does in its way provide benefits over existing frameworks. It extends VUnit and adds extra functionality. It does this by concentrating user input into one file and simplifying wave trace analysis.

It is important however to consider that this is only the first version of this system, which still has many flaws. While it extends VUnit, it also takes away a lot of the functionality the VUnit framework offers. By better supporting this framework an even better system could be made. Also, it could be an improvement to support other frameworks, like CoCoTB, as well.

REFERENCES

[1] A. Chapyzhenka, "WaveJSON · drom/wavedrom Wiki," WaveDrom, 5 May 2014. [Online]. Available: https://github.com/drom/wavedrom/wiki/WaveJSON. [Accessed 13 Oktober 2015].

[2] A. Chapyzhenka, "WaveDrom - Digital timing diagram everywhere," WaveDrom, 2011. [Online]. Available: http://wavedrom.com/. [Accessed 30 September 2015].

[3] PotentialVentures, "Welcome to Cocotb's documentation!," PotentialVentures, 2014. [Online]. Available: http://cocotb.readthedocs.io/. [Accessed 14 04 2016].

[4] agilesoc, "SVUnit," agilesoc, 5 June 2015. [Online]. Available: http://www.agilesoc.com/open-source-projects/svunit/. [Accessed 17 May 2016].

[5] OSVVM organisation, "Open Source VHDL Verification Methodology," OSVVM oranisation, 2013. [Online]. Available: http://osvvm.org/. [Accessed 2016].

[6] Bitvis, "UVVM - Overview," Bitvis, 2013. [Online]. Available: http://bitvis.no/products/uvvm/. [Accessed 2016].

[7] L. Asplund, "VUnit documentation," 2014. [Online]. Available: vunit.github.io/documentation. [Accessed 2016].