Towards Beneficial Hardware Acceleration in HAVEN: Evaluation of Testbed Architectures*

Marcela Šimková and Ondřej Lengál

Faculty of Information Technology, Brno University of Technology, Czech Republic
{isimkova,ilengal}@fit.vutbr.cz

Abstract. Functional verification is a widespread technique to check whether a hardware system satisfies a given correctness specification. As the complexity of modern hardware systems rises rapidly, it is a challenging task to find appropriate techniques for acceleration of this process. In our previous work, we developed HAVEN, an open verification framework that enables hardware acceleration of functional verification runs by moving the design under test (DUT) into a verification environment in a field-programmable gate array (FPGA). In the original version of HAVEN, the generator of input stimuli, the scoreboard and the transfer function still resided in a software simulator, and the peak acceleration ratio achieved was over 1,000. In the currently presented paper, we further extend HAVEN with hardware acceleration of the remaining parts of the verification environment. This enables the user to choose from several different testbed architectures which are evaluated and compared. We show that each architecture provides a different trade-off between the comfort of verification and the degree of acceleration. Using the highest degree of acceleration, we were able to achieve the speed-up in the order of hundreds of thousands while still being able to employ assertion and coverage analysis.

1 Introduction

Functional verification is a simulation-based technique which is typically used in the pre-silicon phase of the development cycle to verify not only functional aspects but also reliability and safety properties of hardware systems. Due to its ability to uncover the vast majority of design errors in a reasonable time and thus decrease the *time to market* of the developed product, functional verification has become the verification method of choice for many successful projects.

The main idea of functional verification is to *generate* a set of constrained-random test vectors and apply them to the verified system (called the *design under test*, or DUT) in a simulator. The observed response is then compared to the expected one as specified by a provided *transfer function*. In order to have a strong confidence in the correctness of the verified system, a high level of coverage of the system's state space needs to be achieved. This issue can be addressed in the following two ways: (*i*) to find a method how to generate test vectors that cover critical parts of the state space, and (*ii*) to maximise the number of the vectors tested. Simulation-based pre-silicon verification

* This work was supported by the Czech Science Foundation (project 102/09/H042), the Czech Ministry of Education (projects LD12036 and MSM 0021630528) and the BUT FIT projects FIT-S-11-1 and FIT-S-12-1. An extended version of this paper is available as the technical report [1].

approaches including functional verification provide verifiers and designers with great comfort while debugging a failing component, checking formal assertions or performing coverage analysis.

Because of the limitation in the speed of software simulation, even with a high effort devoted to the pre-silicon verification, some previously uncovered functional errors are recognised only after the system is manufactured. In order to eliminate as many remaining bugs as possible before the target device is fabricated, verification is currently applied even in the post-silicon phase of the development cycle when a prototype running at the frequency close to that of the target device is available [3].

In recent years, several approaches that addressed the issue of performance of pre-silicon verification appeared. The first approach discussed in [4,5,6] translates VHDL or Verilog testbenches, which contain not directly synthesisable behavioural constructs, using advanced synthesis techniques into the synthesisable subset of the corresponding language. Note that these techniques are limited since some of the non-synthesisable constructs, such as reading from a file or evaluation of recursive functions, still cannot be synthesised. With the advent of higher-level *hardware verification languages* (HVLs) for writing testbenches, with SystemVerilog being the most prominent, automatic synthesis of testbenches that use advanced features, such as *constrained-random stimulus generation*, *coverage-driven* and *assertion-based* verification, has become even more infeasible.

However, soon after the introduction of HVLs, several *transaction-based* methodologies emerged, e.g. SystemVerilog-based VMM, OVM, and UVM. These methodologies use higher-level of abstraction and group sequences of stimuli applied to the DUT into *transactions* that are delivered to drivers of the DUT. Then it is possible to accelerate the performance of a testbench by dividing the testbench into the synthesised part (including the drivers) that is placed in a hardware emulator, and the behavioural part that runs on a CPU, such that the two parts communicate using simple channels. Solutions that use emulators to accelerate functional verification has been provided by major companies that focus on tools for hardware verification. Examples of these emulator-based solutions are Mentor Graphics' Veloce2 technology [7] and Cadence's TBA [8] that use emulators running on frequencies in the order of MHz. Synopsys [11] provide solution for prototyping of ASICs based on *field-programmable gate arrays* (FPGAs). A similar approach is taken by Huang *et al* [9]; their proposal is also to place the DUT with necessary components in an FPGA, and in addition provide limited observability of the DUT's signals. Nevertheless, to the best of our knowledge, there is currently still no available working implementation based on their proposal. Unfortunately, we could not perform a detailed comparison of these solutions as they are not available to us.

The authors of [3] relate pre-silicon and post-silicon verification in terms of achieving *coverage closure*. Instead of observing values of internal signals, the approach presented for post-silicon verification observes the behaviour of a post-silicon exerciser (which is not given by a set of test vectors but rather by a test template) in the pre-silicon simulation environment and determines the probability of the exerciser hitting certain cover points in a given number of clock cycles.

We focus our research on bridging the gap between pre- and post-silicon verification using hardware acceleration with functional verification features. In [2], we introduced

HAVEN (**H**ardware-**A**ccelerated **V**erification **EN**vironment), an open framework[1] for hardware-accelerated functional verification of hardware that tackles the bottleneck of the simulation speed of a DUT by moving the DUT into a verification environment in an FPGA. Using this solution, we were able to achieve the speed-up of over 1,000.

In the currently presented paper, we describe the new features added to HAVEN in order to support seamless transition from pre- to post-silicon verification using several architectures of the verification testbed. The user can start with the pure software version of the functional verification environment to debug base system functions and discover the main bulk of errors. Later, when the simulation cannot find any new bugs in a reasonable time, the user can start to incrementally move some parts of the verification environment from software to hardware, with each step obtaining a different trade-off between the acceleration ratio and the debugging comfort.

The rest of the paper is structured as follows. In Section 2, we give a detailed description of the main features of HAVEN. In Section 3 we propose several architectures of the HAVEN testbed and in Section 4 we evaluate them using a set of experiments. Section 5 concludes the paper and gives directions for future work.

2 The HAVEN Verification Framework

HAVEN [2] is a SystemVerilog verification framework that allows to speed up functional verification runs using an FPGA-based accelerator. The DUT that is being verified is synthesised and placed into a testbed in the FPGA, and generated transactions are passed to the accelerator instead of the model of the DUT in the software simulator. The cycle accuracy is maintained in the accelerator so that a failed accelerated verification run can be easily reproduced in the perfect debug environment of the simulator. In order to be able to detect a violation of expected internal behaviour, protocols' specifications, etc., and debug a failing component directly in hardware, HAVEN enables to connect *assertion checkers* and *signal observers* that check validity of assertions and observe values of internal signals and display them as waveforms.

Using the solution presented in [2], we were able to achieve the acceleration ratio of over 140 when we included the time for generation of test vectors in software and over 1,000 when we did not include it (for pre-generated test vectors). During the evaluation, we observed that the main performance bottlenecks were generation of constrained-random transactions, maintaining transactions in the scoreboard and comparing them to the outputs of the DUT. In this paper, we address these issues and extend HAVEN with even better support for hardware acceleration by providing hardware implementations of the following components of the verification environment:

Hardware Generator The Hardware Generator consists of a random number generator (we used the fast hardware implementation of the Mersenne Twister from [10] which provides a random vector in each clock cycle) and an adapter to the desired format with a constraint solver. The generator seed as well as parameters of transactions can be set from the simulator using a configuration interface.

Hardware Scoreboard Compares data from the DUT and the Transfer Function unit.

Transfer Function Hardware implementation of a transfer function depends on the verified component and can be performed in several ways. For components with

[1] `http://www.fit.vutbr.cz/~isimkova/haven/`

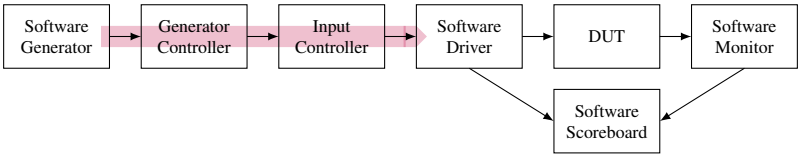

Fig. 1. Software version (**`SW-FULL`**).

an already existing reference hardware implementation, we can use this as the transfer function. If only a software implementation of the transfer function is available, it is possible to use a soft processor core and run the function on the processor.

Coverage Monitor In order to be able to guarantee reaching coverage closure in larger designs, the Coverage Monitor may be used to check whether given points of the DUT's state space have been covered. The component is connected to the wires which are to be checked and periodically sends the information about triggered cover points to the simulator. Since this component uses a register for every cover point, it is recommended for monitoring coverage of so far not covered points only.

3 Architectures of HAVEN

In this section we show how the components presented in the previous section may be (together with the components from [2]) assembled to create several different testbed architectures, each suitable for a different use case and a different phase of the overall verification process. We start our description with the non-accelerated version running solely in the simulator and proceed by moving components of the verification environment into hardware in several steps.

Software version (`SW-FULL`). All components of the verification environment are in the software simulator (Fig. 1). The Software Generator produces input transactions which are propagated to the Software Driver and further supplied on the input interface of the DUT. Copies of transactions are sent to the Software Scoreboard where the expected output is computed using a reference transfer function. The Software Monitor drives the output interface of the DUT and sends received output transactions also to the Software Scoreboard to be compared to the expected ones.

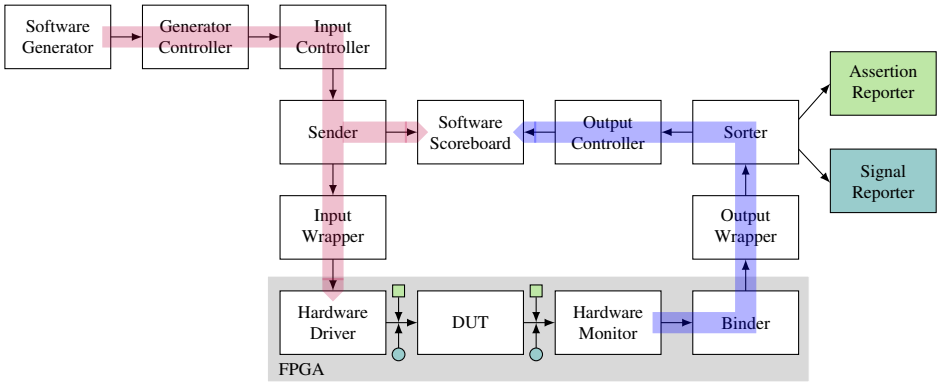

Fig. 2. Hardware DUT version (**`HW-DUT`**).

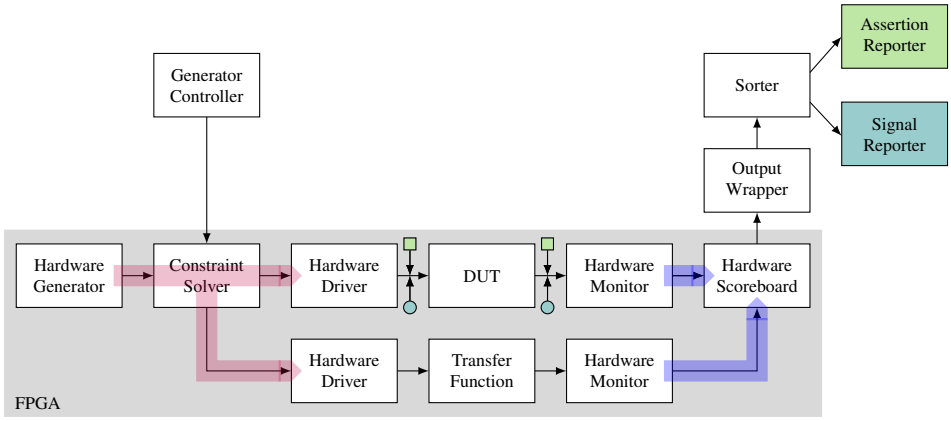

Fig. 3. Hardware version (**`HW-FULL`**).

Hardware Generator version (`HW-GEN`). The architecture is similar to the **`SW-FULL`** version with the exception of the Hardware Generator and the Constraint Solver, which are placed in the FPGA and send generated transactions to the simulator.

Hardware DUT version (`HW-DUT`). In this architecture (Fig. 2), the Software Generator sends input transactions to the verification environment in hardware. In addition, a copy of every transaction is passed to the Software Scoreboard for further comparison. The Hardware Driver and the Hardware Monitor fulfill the same functions as their counterparts in the **`SW-FULL`** version, but they drive the input and output interfaces of the DUT running in the FPGA. The output transactions produced by the DUT are directed from the Hardware Monitor to the Software Scoreboard.

Hardware Generator and DUT version (`HW-GEN-DUT`). This architecture is similar to the **`HW-DUT`** version but the generator is in hardware, as in **`HW-GEN`**.

Hardware version (`HW-FULL`). All core components of the verification environment in this architecture (Fig. 3) reside in the FPGA. The components in the software only set constraints for the Constraint Solver and report assertion failures, coverage statistics, or display waveforms of signals from hardware components.

For those architectures of the HAVEN testbed that place the DUT into the FPGA (**`HW-DUT`**, **`HW-GEN-DUT`**, **`HW-FULL`**), it is possible to use the following optional components in hardware:

Assertion Checkers (illustrated by squares in figures) detect assertion violations of the DUT in hardware and report them to Assertion Reporters in the simulator, which in turn display them to the user.

Signal Observers (illustrated by circles in figures) store values of signals in hardware and send them to Signal Reporters in the simulator to be displayed as waveforms.

Coverage Monitors check coverage as described in Section 2.

Table 1. Results of experiments: times for verifying 100,000 transactions (in seconds).

Component	FIFO	HGEN	HGEN×2	HGEN×4	HGEN×8	HGEN×16
SW-FULL	199.	319.	1,126.	1,617.	2,539.	5,650.
HW-GEN	268.	308.	1,101.	1,984.	3,274.	7,534.
HW-DUT	65.	45.	48.	48.	48.	48.
HW-GEN-DUT	74.	22.	12.	12.	13.	13.
HW-FULL	0.0148	0.0205	0.0205	0.0239	0.0341	0.0410

Table 2. Results of experiments: acceleration ratios.

Component	FIFO	HGEN	HGEN×2	HGEN×4	HGEN×8	HGEN×16
HW-GEN	0.743	1.036	1.023	0.815	0.776	0.750
HW-DUT	3.062	7.089	23.458	33.688	52.896	117.708
HW-GEN-DUT	2.689	14.500	93.833	134.750	195.308	434.615
HW-FULL	13,429.	15,564.	54,925.	67,626.	74,347.	137,875.

4 Evaluation

We performed a set of experiments using an acceleration card with the Xilinx Virtex-5 FPGA supporting fast communication through the PCIe bus in a PC with two quad-core Intel Xeon E5620@2.40 GHz processors and 24 GiB of RAM, and Mentor Graphics' ModelSim SE-64 10.0c as the simulator. We evaluated the performance of the architectures of HAVEN presented in the previous section on several hardware components: a simple FIFO buffer and several versions of a hash generator (HGEN) which computes the hash value of input data, each version with a different level of parallelism (2, 4, 8, and 16 units connected in parallel)[2].

For each of the components, Table 1 gives the wall-clock time it took to verify the component for 100,000 input transactions for each architecture of the HAVEN testbed (because of issues with precise measurements of the time for the **HW-FULL** architecture, for this case we measured the time it took to verify the component for 1,000,000,000 input transactions and computed the average time for 100,000 transactions). Table 2 in turn shows the acceleration ratio of each of the architectures of the HAVEN testbed against the **SW-FULL** architecture.

We can observe several facts from the experiments. First, they confirm that the time of simulation (**SW-FULL**) increases with the complexity of the verified DUT, so it is not feasible to simulate complex designs for large numbers of transactions. Second, we can observe that it is not reasonable to use the simulator with hardware acceleration of the transaction generator only (**HW-GEN**), at least for simple input protocols, which is the case of our protocol. In this case, the overhead of communication with the accelerator is too high. However, for the case when the DUT is also in hardware (**HW-GEN-DUT**), hardware generation of transactions is (with the exception of the FIFO unit) advantageous compared to software generation (**HW-DUT**). Lastly, we can observe that the major speed-up of the hardware version (**HW-FULL**) makes this version preferable to use for very large amounts of transactions, e.g. when trying to reach coverage closure. Running verification of HGEN×16 for a billion transactions, which took less than 7 minutes in this version, would take more than 21 months in the **SW-FULL** version.

[2] Further details about the components and the experiments can be found in [1].

5 Conclusions and Future Research

In this paper, several extensions of the HAVEN verification framework were presented. These extensions allow the user to incrementally move parts of a verification environment into an FPGA-based accelerator and thus accelerate the verification process. Several architectures of the HAVEN testbed allow the user to choose the most suitable version for the preferred trade-off between acceleration ratio and debugging capabilities. The best speed-up achieved in our experiments for the case that used the **`HW-FULL`** testbed was over 100,000 while still performing assertion and coverage analysis.

In the future, we wish to extend HAVEN with a technique to automatically drive generation of test vectors to target coverage holes given by continuously measured coverage information. As a result, we expect to obtain a set of input test vectors or settings of the software generator which would achieve a high level of coverage in regression testing. These could also be used in the hardware generator, thus improving its ability of reaching coverage closure. Such generators might also be useful in post-silicon validation as they are closer to the speed of real hardware. A challenging direction is to develop a technique for representation of triggered cover points that would be feasible to be used in hardware Coverage Monitors for a large amount of cover points, as the currently used technique does not scale well. In addition, our future effort will lead also to the integration of HAVEN into various research areas, especially into diagnostics, where we wish to explore the capability of functional verification to improve the quality of fault-tolerant systems. Collaboration on any of these issues is welcome.

References

1. M. Šimková, and O. Lengál. Towards Beneficial Hardware Acceleration in HAVEN: Evaluation of Testbed Architectures. Technical Report FIT-TR-2012-03, FIT BUT, 2012. `http://www.fit.vutbr.cz/~ilengal/pub/FIT-TR-2012-03.pdf`
2. M. Šimková, O. Lengál, and M. Kajan. HAVEN: An Open Framework for FPGA-Accelerated Functional Verification of Hardware. To appear in *Proc. of HVC'11*, LNCS 7261, Springer.
3. A. Adir, A. Nahir, A. Ziv, Ch. Meissner, and J. Schumann. Reaching Coverage Closure in Post-silicon Validation. In *Proc. of HVC'10*, p. 60–75, 2010.
4. R. Henftling, A. Zinn, M. Bauer, M. Zambaldi, and W. Ecker. Re-Use-Centric Architecture for a Fully Accelerated Testbench Environment. In *Proc. of DAC'03*, p. 372–375, 2003, ACM.
5. M. R. Kakoee, M. Riazati, and S. Mohammadi Generating RTL Synthesizable Code from Behavioral Testbenches for Hardware-Accelerated Verification. In *Proc. of DSD'08*, p. 714–720, 2008, IEEE.
6. Y.-I. Kim, and C.-M. Kyung. Automatic Translation of Behavioral Testbench for Fully Accelerated Simulation. In *Proc. of ICCAD'04*, p. 218–221, 2004, IEEE.
7. Mentor Graphics. Veloce2. 2012. `http://www.mentor.com/products/fv/emulation-systems/veloce/`
8. Cadence. Transaction-based Acceleration (TBA). 2012. `http://www.cadence.com/products/sd/pages/transactionacc.aspx`
9. C.-Y. Huang, Y.-F. Yin, C.-J. Hsu, T. B. Huang, and T. M. Chang. SoC HW/SW Verification and Validation. In *Proc. of ASPDAC'11*, IEEE, 2011.
10. HT-LAB. Mersenne Twister, MT32: Pseudo Random Number Generator for Xilinx FPGA. 2007. `http://www.ht-lab.com/freecores/mt32/mersenne.html`
11. Synopsys. FPGA-Based Prototyping. 2012. `http://www.synopsys.com/Systems/FPGABasedPrototyping/Pages/default.aspx`