Towards Beneficial Hardware Acceleration in HAVEN: Evaluation of Testbed Architectures

FIT BUT Technical Report Series

Marcela Šimková and Ondřej Lengál

Technical Report No. FIT-TR-2012-03
Faculty of Information Technology, Brno University of Technology

Last modified: January 4, 2013

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. *Coverage-driven verification* is an approach that provides the verifier with a detailed coverage feedback of verification runs so that new tests can be written to cover parts of the state space which had not been exercised so far. Formal *assertions* may be used with advantage during verification runs to provide further checks of internal synchronisation and expected operations of the system by creating implicit monitors at critical points of the system without the need to create separate testbenches where those points would be externally visible.

Simulation-based pre-silicon verification approaches including functional verification provide verifiers and designers with great comfort while debugging a failing component, checking assertions or performing coverage analysis. Values of internal signals are easily observable with arbitrary depth of history and it is easy to introduce precisely timed events to the verified component. In general, pre-silicon verification approaches have improved significantly during the last decade and a lot of new techniques, tools and verification methodologies have been developed. One of the negative consequences of Moore's law, which claims that the number of transistors on integrated circuits doubles approximately every two years, is that it is necessary to verify still more and more complex systems. However, as the performance of computer processors' cores has reached its limit (and the overall performance of computer processors is currently increased mainly by placing multiple cores in a single chip), software simulators of logical circuits cannot benefit from this increase much, because the simulation task is difficult to be parallelised.

Because of this 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. The reasons why these errors had not been found in the pre-silicon stage are, e.g. because the errors may appear after several hours of operation of the target device (a condition which may take years or even centuries to reproduce in a simulator), or because the errors might have been introduced by the synthesiser or caused by the discrepancy between the behaviour of a resource and the behaviour of its simulation model. 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]. Unfortunately, not many techniques standard for simulation can be directly used in post-silicon verification, as these techniques heavily rely on perfect observability of internal signals of the system, while in post-silicon, the observation of a system is often limited to the use of logic analysers, oscilloscopes, etc., and often only errors leading to some catastrophe (such as a system crash) are detectable. Further, it is also difficult to apply a sequence of events with precise timing as the post-silicon testing environment may not be able to deliver large amounts of data quickly enough.

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*. Transactions are sent to *drivers* that decode them and apply proper stimuli to the DUT. Since drivers can usually be written using synthesisable constructs, it is possible to accelerate the performance of a testbench by dividing the testbench into the synthesised part 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 [13] 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 [1], we introduced **HAVEN** (**H**ardware-**A**ccelerated **V**erification **EN**vironment), an open framework[1] for hardware-accelerated functional verification of hardware designs that tackles the bottleneck of the simulation speed of a highly parallel DUT by moving the DUT into a verification environment in an FPGA. Using this solution, we were able to achieve the acceleration ratio 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.

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

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 [1] 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 violation of expected internal behaviour, protocols' specifications, etc., HAVEN enables to connect *assertion checkers* implemented as finite-state automata to report violations of assertions to the user. Moreover, for verification runs which are not easily reproducible in the software simulator (e.g. runs with a very long trace), it is possible to observe values of signals directly in hardware using so-called *signal observers* and display the waveform to the user in the simulator.

HAVEN is advantageous especially in the (currently expanding) area of devices that use FPGAs, since the verification may run directly on the target platform and check the system *after synthesis* and not only a model in simulation. This is even more advantageous when the verified component uses some specialised resources of the FPGA (such as in-built multipliers), or a design which is already synthesised and its source code is not available, so that the real hardware is evaluated instead of simulation models which may contain inaccuracies or errors.

For the evaluation of HAVEN, we chose a set of components that use the FrameLink protocol (its description can be found in [2]) at their input and output interfaces. FrameLink is a frame-oriented point-to-point protocol developed for the use in network applications, which is based on Xilinx's LocalLink [10]. A frame on FrameLink may consist of several parts of an arbitrary length and there may be any number of delays inside a frame part and between parts.

Using the solution presented in [1], 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 (which is a relevant value for the case when test vectors are pre-generated and stored e.g. in a file). 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 of an arbitrary width (we used the fast hardware implementation of the

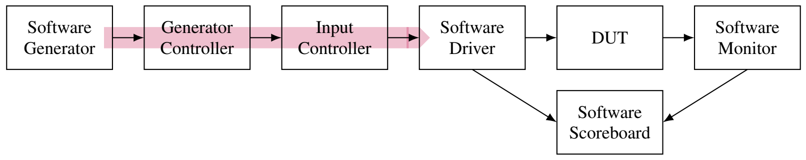

Fig. 1. Software version (**`SW-FULL`**).

Mersenne Twister from [11] which provides a random vector in each clock cycle) and an adapter to the FrameLink format with a constraint solver. The seed of the generator as well as the constraints on the number/lengths of parts and delays can be set from the simulator using a simple configuration interface.

Hardware Scoreboard The Hardware Scoreboard is a component that selects data sent from hardware monitors corresponding to output transactions from the DUT and performs comparison of these data from several interfaces. Any discrepancy in the received data is reported to the user.

Transfer Function Hardware implementation of the Transfer Function depends on the verified component and can be performed in several ways. For components with an already existing reference hardware implementation, we can use this as the transfer function (this use case may be suitable e.g. for regression testing). In the case only a software implementation of the transfer function is available, it is possible to use a soft processor core (e.g. MicroBlaze [12]) and run the transfer function as a program on the processor. If the transfer function takes a long time to be evaluated, a block of processor cores working in parallel may be used.

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. This information is reported to the user so that she knows which cover points have not been triggered. The user can in turn e.g. write directed tests or change settings of the generator to target these points. 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`). This architecture is similar to the standard architecture of a functional verification testbench. All components of the verification environment are situated 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. The copy of a transaction is 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.

Hardware Generator version (`HW-GEN`). The architecture (in Fig. 2) is similar to the **`SW-FULL`** version with the exception of the Hardware Generator and the Constraint Solver, which are placed in the FPGA. Generated transactions are sent to software to be applied to the model of the DUT in the simulator and sent to the Software Scoreboard.

Hardware DUT version (`HW-DUT`). In this architecture (Fig. 3), the Software Generator is used and the flow of input transactions is sent to the verification environment in hardware. In addition, a copy of every transaction is sent to the Software Scoreboard for further comparison. The Hardware Driver and the Hardware Monitor fulfill the same functions as their software 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`). (Fig. 4) This architecture is similar to the **`HW-DUT`** version, but the generator is in hardware, as in the **`HW-GEN`** version.

Hardware version (`HW-FULL`). All core components of the verification environment in this architecture (Fig. 5) reside in the FPGA. The components in the software environment 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:

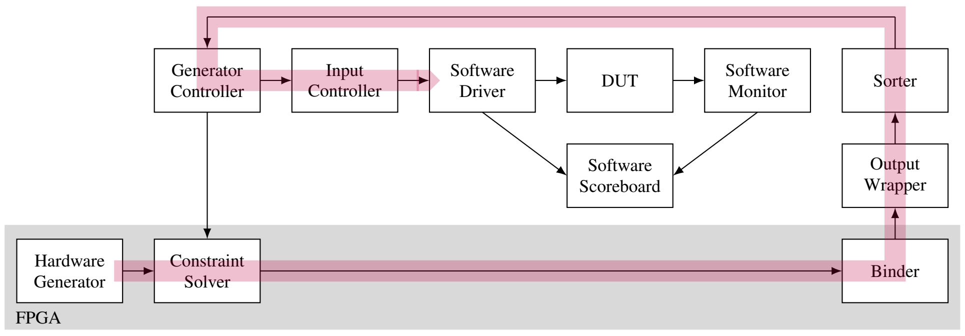

Fig. 2. Hardware Generator version (**`HW-GEN`**).

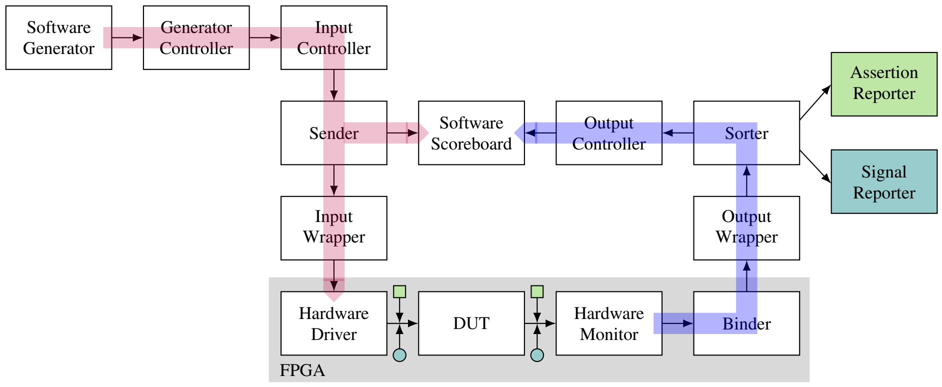

Fig. 3. Hardware DUT version (**`HW-DUT`**).

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 periodically send them to Signal Reporters in the simulator to be displayed as waveforms to the user.

Coverage Monitors check coverage as described in Section 2.

4 Evaluation

We performed a set of experiments using an acceleration card with the Xilinx Virtex-5 FPGA[2] supporting fast communication through the PCIe bus in a PC with two quad-

[2] We used Xilinx Virtex-5 XC5VLX155T (speed grade -2) with 24,320 slices, which is roughly equivalent to 155,000 logical gates.

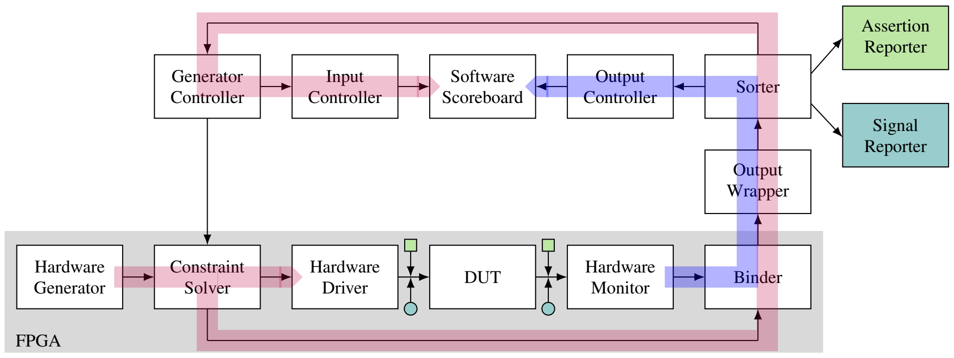

Fig. 4. Hardware Generator and DUT version (**`HW-GEN-DUT`**).

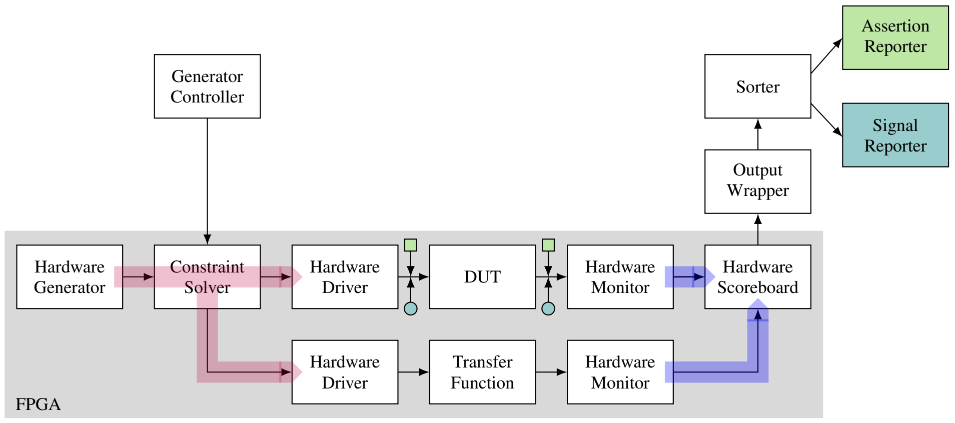

Fig. 5. Hardware version (**`HW-FULL`**).

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). Resources consumed by these components (in the number of occupied slices and portion of the used FPGA they took) are in Table 1. Table 2 shows resources consumed by hardware components of HAVEN; from the table it can be observed that the overhead of HAVEN is quite negligible.

For each of the components, Table 3 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, we measured for this case 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 4 in turn shows the acceleration ratio of each of the architectures of 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 that 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 FrameLink. 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 with 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$\times$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.

Table 1. Resource consumption of the evaluated components.

Component	**FIFO**	**HGEN**	**HGEN×2**	**HGEN×4**	**HGEN×8**	**HGEN×16**
Slices (of 24,320)	282	805	2,030	3,637	7,376	16,821
Slices [%]	1.16	3.31	8.35	14.95	30.33	69.17

Table 2. Resource consumption of HAVEN components. **GEN**: Hardware Generator and Constraint Solver, **SB**: Hardware Scoreboard, **CHECKER**: Assertion Checker, **COV**: Coverage Monitor, **OBSERVER**: Signal Observer.

Component	**DRIVER**	**MONITOR**	**GEN**	**SB**	**CHECKER**	**COV**	**OBSERVER**
Slices (of 24,320)	161	64	567	124	55	18	64
Slices [%]	0.66	0.26	2.33	0.88	0.23	0.07	0.26

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 hardware 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 fully accelerated testbed was over 100,000 while still performing assertion checking 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 to reach 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.

5.1 Acknowledgements

This work was supported by the Czech Ministry of Education (projects LD12036 and MSM 0021630528), the Czech Science Foundation (project 102/09/H042), the Brno City Municipality (Brno Ph.D. Talent programme) and the BUT FIT projects FIT-S-11-1 and FIT-S-12-1.

Table 3. 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 4. 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.

References

1. 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.
2. M. Šimková, O. Lengál, and M. Kajan. HAVEN: An Open Framework for FPGA-Accelerated Functional Verification of Hardware. Technical Report FIT-TR-2011-05, FIT BUT, 2011. `http://www.fit.vutbr.cz/~ilengal/pub/FIT-TR-2011-05.pdf`
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, Springer.
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. Xilinx. LocalLink User Interface. 2012. `http://www.xilinx.com/products/intellectual-property/LocalLink_UserInterface.htm`
11. HT-LAB. Mersenne Twister, MT32: Pseudo Random Number Generator for Xilinx FPGA. 2007. `http://www.ht-lab.com/freecores/mt32/mersenne.html`
12. Xilinx. MicroBlaze Soft Processor Core. 2012. `http://www.xilinx.com/tools/microblaze.htm`
13. Synopsys. FPGA-Based Prototyping. 2012. `http://www.synopsys.com/Systems/FPGABasedPrototyping/Pages/default.aspx`

A Encountered Issues

In this appendix, we briefly describe issues that we encountered while developing and extending the HAVEN framework and the approaches we chose to deal with them.

A.1 Optimisations

SystemVerilog is a high-level programming language that is usually interpreted by an HDL simulator. This may be one the reasons why, according to our experience, is the performance of a SystemVerilog program considerably inferior to the performance of a C program with the same functionality. Fortunately, SystemVerilog allows a program to call an external C program using the so-called *Direct Programming Interface* (DPI). Using a C implementation of critical tasks often significantly improves the speed of simulation.

A.2 Multithreading

Some software tasks in certain architectures of HAVEN are mutually independent and call for being run in parallel. An example of such an architecture and a task is the **`HW-DUT`** version, in which (*i*) data generated in the Software Generator are sent from the simulator to the accelerator, and (*ii*) data from the accelerator are delivered to the Software Scoreboard. These two tasks are natural to run in parallel (they are actually *needed* to run in parallel because the sizes of transaction buffers in the accelerator are limited and, therefore, output transactions need to be received at the output periodically, otherwise the accelerator would be blocked), however, most interpreters of SystemVerilog do not support creation and management of truly parallel threads. Although the SystemVerilog language supports parallel threads, these threads run in parallel in the *simulation time* while being usually run sequentially in the *real time*, switching context in the nonpreemptive way (in order to switch a context, a thread needs to issue some blocking command, for instance reading from an empty queue or issuing the wait command).

We address this by issuing the `"#10ns;"` statement (that pauses the thread for 10 ns of the simulation time) after sending every 100,000 transactions to the accelerator (we observed that the performance for this setting was the best) to enforce a context switch. However, in future, we wish to optimise this by creating a truly parallel wrapper in C connected to the simulator via DPI.

A.3 Preventing Deadlock in Hardware

As mentioned in [1], we maintain cycle-accuracy of accelerated verification by the means of clock gating in the case the Hardware Driver does not have enough data to send to the DUT, or when the Hardware Monitor is blocked. In some cases, e.g. in the **HW-FULL** version, a special care needs to be taken so that this would not lead to a deadlock, which may occur in the following case.

Suppose the Transfer Function is a reference implementation of the verified DUT and both may contain several processed transactions inside (e.g. they use a FIFO). In case the maximum number of transactions inside the units varies significantly, the following condition may happen. Suppose that the maximum number of transactions in the DUT is larger than the maximum number of transactions in the Transfer Function. Then, it may happen that the Transfer Function blocks its input, which in turn stops the Hardware Driver of the Transfer Function, and, consequently via back-pressure, also the Hardware Generator. The Hardware Generator cannot deliver data to the DUT's Hardware Driver that in turn stops the clock of the DUT. Because the DUT is stopped and still contains some transactions that are being processed, these are not passed to the Hardware Scoreboard which is filled up with output transactions from the Transfer Function, thus blocking the Hardware Monitor of the Transfer Function. The Hardware Monitor of the Transfer Function than in turn stops the clock of the Transfer Function. The Transfer Function cannot accept data from its Hardware Driver and the hardware environment is in the state of a deadlock. We solved this solution by adding the option to insert buffers in front of the Hardware Drivers so that the Hardware Drivers would not be able to block the Hardware Generator.

A.4 Verification of Multiple DUTs

The versions of HAVEN that place the DUT into hardware (**HW-DUT**, **HW-GEN-DUT**, **HW-FULL**) support the feature of verifying multiple DUTs (consider, e.g., a generic data width component where each DUT is a specialisation of the component for a specific data width) at once. The DUTs can be put into the FPGA in parallel (as in the **HW-FULL** version) and their input interfaces exercised at the same time during a single verification run.