HAVEN: An Open Framework for FPGA-Accelerated Functional Verification of Hardware

FIT BUT Technical Report Series

Marcela Šimková, Ondřej Lengál, and Michal Kajan

Technical Report No. FIT-TR-2011-05
Faculty of Information Technology, Brno University of Technology

Last modified: January 12, 2012

HAVEN: An Open Framework for FPGA-Accelerated Functional Verification of Hardware

Marcela Šimková, Ondřej Lengál, and Michal Kajan

FIT, Brno University of Technology, Czech Republic

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 this paper we present HAVEN, a freely available open functional verification framework that exploits the field-programmable gate array (FPGA) technology for cycle-accurate acceleration of simulation-based verification runs. HAVEN takes advantage of the inherent parallelism of hardware systems and moves the verified system together with transaction-based interface components of the functional verification environment from software into an FPGA. The presented framework is written in SystemVerilog and complies with the principles of functional verification methodologies (OVM, UVM), assertion-based verification, and also provides adequate debugging visibility, making its application range quite large. Our experiments confirm the assumption that the achieved acceleration is proportional to the complexity of the verified system, with the peak acceleration ratio being over 1,000.

1 Introduction

Today's highly competitive market of consumer electronics is very sensitive to the time it takes to introduce a new product (the so-called *time to market*). This has driven the demand for fast, efficient and cost-effective methods of *verification* of hardware systems. There is a variety of options applicable to this issue: (*i*) formal verification, (*ii*) simulation and testing, and (*iii*) functional verification. However, their nature and preconditions for the speed of the verification process are often limiting for verification of complex hardware systems.

Formal verification is an approach based on an exhaustive exploration of the state space of a system, hence it is potentially able to formally prove correctness of a system. The main disadvantages of this method are state space explosion for real-world systems and the need to provide formal specifications of the system's behaviour, which makes this method often hard to use.

Simulation and testing, on the other hand, are based on observing the behaviour of the verified system in a limited number of situations, thus it provides only a partial guarantee of the system's correctness. However, because tests often focus on the typical use of the system and on corner cases, this is often sufficient. Moreover, writing tests is usually faster and easier than writing formal specifications.

Functional verification is a simulation-based method that generates a set of constrained-random test vectors and compares the behaviour of the system for these vectors with the behaviour specified by a provided *reference model* (which is called *scoreboarding*). In order to achieve high level of coverage of a system's state space, it is necessary to (*i*) find a way how to generate test vectors that cover critical parts of the state space, and (*ii*) maximise the number of vectors tested. The generation of appropriate scenarios can be fully automated by an intelligent program that controls coverage results and chooses parameters or a pseudo-random number generator seed according to the achieved coverage. This approach is called *coverage-driven verification*. To facilitate the process of verification and to formally express the intended behaviour, internal synchronization, and expected operations of the system, *assertions* may be used. Assertions create monitors at critical points of the system without the need to create separate testbenches where these points would be externally visible. Assertions also guide the verification task and quicken the verification because they can provide feedback at the internal level of the device so that it is possible to locate the cause of a problem faster than from the output of a simulation. All abovementioned features are effective to check system correctness and maximise the efficiency of the overall verification process.

Simulation-based verification approaches including functional verification provide great opportunity to inspect the internal behaviour of a running system, but they suffer from the fact that software simulation of inherently parallel hardware is extremely slow when compared to the speed of real hardware. The gap between the speed of simulation and the speed of real hardware widens with the increasing complexity of hardware systems. An effort to increase efficiency and speed of simulation or functional verification poses a considerable challenge not only for research teams but also for the commercial sphere ([1,2,3]).

Building upon our experience with different verification approaches and existing studies dealing with acceleration issues, we introduce **HAVEN** (**H**ardware-**A**ccelerated **V**erification **EN**vironment), an open framework that exploits the inherent parallelism of hardware systems to accelerate their functional verification by moving the verified system together with several necessary components of the verification environment to a *field-programmable gate array* (FPGA). To provide advanced level of debugging capabilities, the framework adopts some formal techniques (assertion-based verification) and functional verification techniques (constrained-random stimulus generation, self-checking mechanisms) and enables partial signal observability to achieve appropriate debugging visibility while running in the FPGA. HAVEN is freely available and *open source*[1], and we welcome collaboration on its further development.

Currently, there already exist several approaches to acceleration of functional verification. Mentor Graphics' Veloce technology [1] accelerates simulation by synthesising the *design under test* (DUT) and placing it into an emulator. This provides simulation speed-up while maintaining full signal visibility. The maximum frequency of the emulator is claimed to be 1.5 MHz, which may still not be sufficient for some applications (e.g., applications that need to communicate using a high-speed interface). SEmulator [3] is a system that enables acceleration of simulation of a DUT using FPGA while sacrificing observability of the DUT's signals. Our approach is in many aspects similar

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

to the work of Huang *et al* [2] who also place the DUT with necessary components to an FPGA, but 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.

2 Design of the Verification Framework

HAVEN is a SystemVerilog verification framework that allows users to run either a *non-accelerated* or an *accelerated* version of the same testbench with a cycle-accurate time behaviour. The non-accelerated version runs entirely in a simulator, while the accelerated version uses an FPGA to accelerate verification runs. Providing two versions allows to use the framework efficiently in different stages of the design flow, from debugging base system functions in a simulator to stress testing with millions of test vectors using hardware acceleration. After creating the basic verification environment, switching between the two versions is as easy as changing a single parameter of the verification.

The non-accelerated version of the framework presents a similar approach to functional verification that is commonly used in verification methodologies. This version is highly efficient in the initial phase of the verification process when testing basic system functionality with a small number of transactions (up to thousands). In this phase it is desirable to have a quick access to the values of all signals of the system and to monitor the verification process in a simulator. Coverage statistics (code coverage, functional coverage, path coverage, etc.) provide a feedback about the state space exploration and allows the user to arrange constrained-random test cases properly to achieve even higher level of coverage. Despite all these advantages the application of the non-accelerated version is very inefficient for verification of complex systems and/or large number of test vectors. The rising complexity of verified hardware systems increases the time of simulation and also memory requirements on the storage of detailed simulation runs.

The accelerated version of the framework moves the DUT to a verification environment in the FPGA. As gate-level simulation takes the biggest portion of verification time, this approach may yield a significant acceleration of the overall process. Complex systems can be verified very quickly and with much higher number of transactions (in the order of millions and more). Behavioural parts of the testbench, such as planning of test sequences, generation of constrained-random stimuli, and scoreboarding, remain in the software simulator. This partitioning is possible because the generic nature of currently prevalent verification methodologies, such as *Open Verification Methodology* (OVM) or its extension *Universal Verification Methodology* (UVM), and transaction-based communication among their subcomponents enable to transparently move some of these components to a specialized hardware, while maintaining good readability for verification engineers.

As the whole framework is implemented in compliance with the paradigm of object-oriented programming, the reuse of verification components (thanks to the mechanisms of encapsulation, inheritance and polymorphism) may lead to a considerable increase in productivity. The framework offers a library of prepared basic and extended verification components (classes) that are organized into packages. Thanks to these, assembling

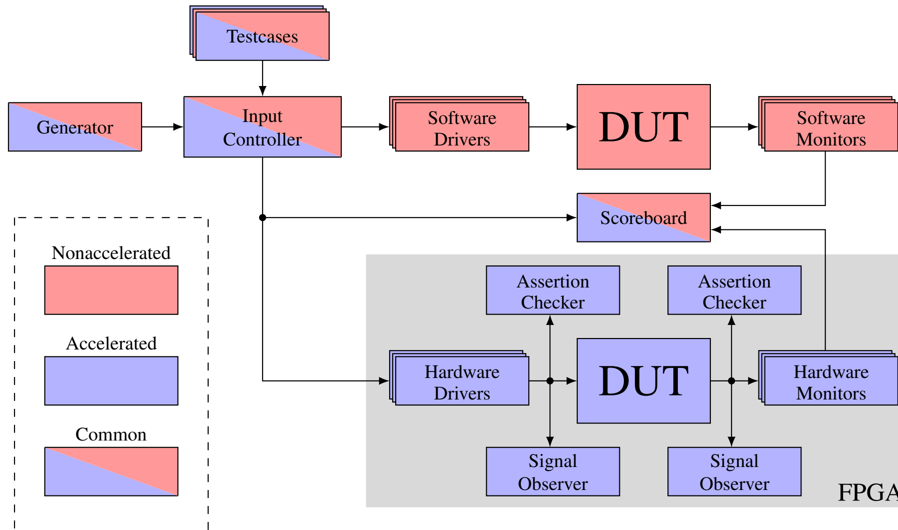

Fig. 1. The architecture of the accelerated version of HAVEN.

packages with new user-defined components or components inherited from base classes can be easily done.

The architecture of the accelerated version of the HAVEN framework is illustrated in Fig. 1. The verified hardware system (DUT) is synthesised and placed in the FPGA. *Testcases*, written by the user, hold parameters such as settings of generics of the system, the number of tested transaction, or options for the generator of random transactions. *Generator* produces constrained-random stimuli, which are typically random data and random delays generated in the ranges specified in the Testcase. *Scoreboard* dynamically predicts the response of the DUT and compares it with received transactions. The remaining blocks were designed for the purpose of acceleration and they are described in detail below. They usually consist of two parts, a hardware component in the FPGA and its software counterpart in the simulator, which communicate together using a generic protocol. The communication with the FPGA is mediated over SystemVerilog's *direct programming interface* (DPI) layer that enables cooperation between SystemVerilog and C code that calls proper system functions. The hardware components were carefully designed to maximise their performance and minimise FPGA resource consumption.

2.1 Driver and Monitor

The software parts of both *Driver* and *Monitor* can be used independently of the hardware parts in the non-accelerated version of HAVEN when the system runs in a simulator. In such a case, the software part of Driver breaks data transactions down into individual signal changes and supplies them on the assigned input interface of the simulated DUT. The copy of the data transaction is sent to Scoreboard. The software part

of Monitor drives the simulated DUT's output interfaces, observes signal transitions, groups them together into high-level data transactions and also passes them to Scoreboard for comparison.

In the accelerated version of HAVEN it is necessary to attach corresponding hardware parts of Driver and Monitor to existing software parts. The main purpose of hardware parts is to manage input and output interfaces of the synthesised DUT which is in this version situated in the FPGA.

2.2 Assertion Checker

SystemVerilog Assertions (SVA) is a standardised language for specification of linear-time temporal properties of systems. Being a core part of SystemVerilog makes SVA easy to use for assertion-based verification of hardware systems. During the verification of a system, it is often useful to provide SVA formulae describing correct behaviour of interfaces of system's subcomponents, so that any violation of an interface protocol during a verification run is captured and reported. Moreover, for standard interfaces, such as PCI or HyperTransport, there are packages with their description (either in SVA or in another assertion language) already available.

Due to its linear-time nature, any SVA formula can be effectively transformed into a Büchi automaton, which is in turn easily synthesisable into a finite-state machine in an FPGA, as shown in [4]. HAVEN allows the user to connect different *Assertion Checkers* to the verified system. An Assertion Checker represents one or more assertions and whenever a violation of any of the assertions is detected, a special packet with information about the nature of the violation is sent to the software environment for further analysis of the error.

2.3 Signal Observer

When functional verification of a system detects an error, it is often convenient to observe internal states of the verified system in order to localise the source of the erroneous behaviour. While this is easy when the system runs in a simulator, observing internal states of a system in an FPGA is not directly possible. Therefore, HAVEN provides *Signal Observer*, a component that monitors values of signals in the system during a verification run. The values are stored into the standardised *Value Change Dump* (VCD) format and can be inspected using any compliant waveform viewer, e.g., ModelSim or GTKWave (in the former case, the file with the dump needs to be converted into a ModelSim internal format by `vcd2wlf`, a tool provided in the ModelSim distribution). For common interfaces, it is easily possible to define specialised components derived from Signal Observer such that the output waveform contains correctly named and grouped signals.

2.4 Platform

HAVEN is built upon NetCOPE[2], a free and open source platform for development of applications in FPGAs. NetCOPE provides abstraction over the type of the FPGA and

[2] `http://www.liberouter.org/netcope/`

the used acceleration card by defining a uniform interface for data transfers between the FPGA and the CPU. Although the focus of NetCOPE is primarily on network applications, it was successfully used for our purpose as well. Moreover, because NetCOPE provides a uniform interface over several protocols, its use makes it very easy to change the framework to use Ethernet or other supported communication protocol for data transfers instead of a system bus without any change to the verification environment itself.

2.5 Cycle-Accurate Behaviour

Our goal is to obtain the same time behaviour of verification runs in the non-accelerated and the accelerated version of the framework. The reason for this is clear: when a bug occurs in the accelerated version, it is possible to run the non-accelerated version with the same failing verification scenario and explore the origin of the failure in detail in the perfect debugging environment of a simulator. Simply placing the DUT in the FPGA is not an option, as the transfer of data through the system bus may be delayed, thus yielding behaviour different from the one obtained from the simulator. We solve this issue by placing the DUT into a separate clock domain and enabling/disabling the clock signal for this domain depending on the state of buffers for the input and output transactions. Thus the run of the DUT in the FPGA is guaranteed to result in the same waveform as the run in the simulator.

2.6 Error Detection

The framework monitors two types of errors: assertion failures and conflicts in Scoreboard such as missing or corrupted data or incorrect order of received transactions. If a bug is detected, the framework provides a short report about the nature of the failure, the simulation time when it occurred and the number of the received transaction which caused the inconsistency in Scoreboard. An example of error detection is given in Appendix B.

3 Experimental Results

We performed a set of experiments using the COMBOv2 LXT155 acceleration card[3] equipped with the Xilinx Virtex-5 FPGA in a server with two quad-core Intel Xeon E5420@2.50 GHz processors and 10 GiB of RAM. The data throughput between the acceleration card and the CPU was measured to be over 10 Gbps for this configuration. We used Mentor Graphics' ModelSim SE-64 6.6a as the SystemVerilog interpreter and in the case of the non-accelerated version also as the DUT simulator. Unfortunately, we were not able to compare HAVEN to other solutions for acceleration of functional verification, because these are mostly not freely available commercial products.

We evaluated the performance of HAVEN on two hardware components that use FrameLink (a communication protocol described in Appendix A) as their input and

[3] `http://www.liberouter.org/card_combolxt.php`

Table 1. Acceleration of verification including the time of transaction generation.

Trans.	FIFO			HGEN		
	NAV [s]	AV [s]	Acc [-]	NAV [s]	AV [s]	Acc [-]
50,000	49	26	1.885	176	37	4.757
100,000	99	52	1.904	353	74	4.770
200,000	197	104	1.894	706	149	4.738
500,000	492	258	1.907	1,760	378	4.656
Trans.	**HGENx2**			**HGENx4**		
	NAV [s]	**AV [s]**	**Acc [-]**	**NAV [s]**	**AV [s]**	**Acc [-]**
50,000	446	37	12.054	614	37	16.595
100,000	897	74	12.122	1,241	74	16.770
200,000	1,795	149	12.047	2,461	148	16.628
500,000	5,181	379	13.670	6,979	378	18.463
Trans.	**HGENx8**			**HGENx16**		
	NAV [s]	**AV [s]**	**Acc [-]**	**NAV [s]**	**AV [s]**	**Acc [-]**
50,000	1,318	37	35.622	5,281	37	142.730
100,000	2,680	75	35.733	10,591	74	143.122
200,000	5,282	149	35.450	21,148	149	141.933
500,000	13,538	377	35.910	53,248	377	141.241

output interface: a simple FIFO buffer and a hash generator (HGEN) which computes the hash value of input data using the Bob Jenkins's Lookup2 hash algorithm [5]. In order to fully exploit the capabilities of the accelerated version of HAVEN it is necessary to verify a complex system. For this purpose we also built systems with 2, 4, 8, and 16 parallelly working HGEN units. We focused on verification of a large number of very short data transactions (1–36 B).

The results of our experiments are given in the following tables. Table 1 compares the times of the whole verification runs of the non-accelerated version (**NAV**) to the times of runs of the accelerated version (**AV**) and the acceleration ratio (**Acc**). During the experiments, we observed that a considerable amount of time is taken by generating transactions, therefore we also measured the times of verification runs without the time of transaction generation, as this value is the same for both the accelerated and the non-accelerated version. These results are given in Table 2. Fig. 2 shows the relation between the complexity of the verified component and the acceleration ratio (without the time of transaction generation).

Table 3 summarises the number of Virtex-5 slices used by the verification core of the accelerated version with the verified component (column **Slices**) and the total number of occupied slices of the FPGA together with NetCOPE (column **Total slices**); the total number of slices of the used FPGA (Xilinx Virtex-5 XC5VLX155T) is 24,320. Column **Build time** gives the time it took to generate the firmware for the FPGA. It can be observed that this time increases significantly as the total resource consumption approaches the capacity of the FPGA. The computed *break-even* number of transactions, which is, loosely speaking, the number of transactions for which the acceleration starts to be beneficial, is further given in column **B-E Transactions**. Formally, this number is

Table 2. Acceleration of verification without the time of transaction generation.

Trans.	FIFO			HGEN		
	NAV [s]	AV [s]	Acc [-]	NAV [s]	AV [s]	Acc [-]
50,000	24	1	24.000	144	5	28.800
100,000	49	2	24.500	289	10	28.900
200,000	97	4	24.250	578	21	27.524
500,000	242	8	30.250	1,440	58	24.828
Trans.	**HGENx2**			**HGENx4**		
	NAV [s]	**AV [s]**	**Acc [-]**	**NAV [s]**	**AV [s]**	**Acc [-]**
50,000	414	5	82.800	582	5	116.400
100,000	833	10	83.300	1,177	10	117.700
200,000	1,667	21	79.381	2,333	20	116.650
500,000	4,861	59	82.390	6,659	58	114.810
Trans.	**HGENx8**			**HGENx16**		
	NAV [s]	**AV [s]**	**Acc [-]**	**NAV [s]**	**AV [s]**	**Acc [-]**
50,000	1,286	5	257.200	5,249	5	1,049.80
100,000	2,616	11	237.818	10,527	10	1,052.70
200,000	5,154	21	245.429	21,020	21	1,000.95
500,000	13,218	57	231.895	52,928	57	928.56

Table 3. Properties of verified components.

Component	Slices	Total slices	Build time [s]	B-E transactions	B-E time [s]
FIFO	420 (1.7 %)	9,362 (38.5 %)	1,473	3,116,000	3,078
HGEN	947 (3.9 %)	9,787 (40.2 %)	1,724	622,000	2,188
HGENx2	2,152 (8.8 %)	11,315 (46.5 %)	1,895	222,000	2,061
HGENx4	3,762 (15.4 %)	12,938 (53.2 %)	2,340	196,000	2,486
HGENx8	7,448 (30.6 %)	16,304 (67.0 %)	3,390	131,000	3,488
HGENx16	15,778 (64.9 %)	22,096 (90.9 %)	7,909	75,000	7,965

defined as the number $trans_{be}$ such that

$$\frac{trans_{be}}{\text{trans_per_sec}(NAV)} = build_time + \frac{trans_{be}}{\text{trans_per_sec}(AV)} \tag{1}$$

where $build_time$ is the build time of the firmware in seconds and $\text{trans_per_sec}(AV)$ and $\text{trans_per_sec}(NAV)$ are the average numbers of transactions processed in a second by the accelerated and the non-accelerated version respectively. It is easy to deduce the following equation for computing $trans_{be}$:

$$trans_{be} = build_time \cdot \frac{\text{trans_per_sec}(AV) \cdot \text{trans_per_sec}(NAV)}{\text{trans_per_sec}(AV) - \text{trans_per_sec}(NAV)} \tag{2}$$

Lastly, the column **B-E time** gives the time at which the break-even number of transactions is reached (i.e., the time of the run of the non-accelerated version).

In Fig. 3 we give the graph of the amount of time saved for given number of transactions when the accelerated version is used, which is again computed from the average

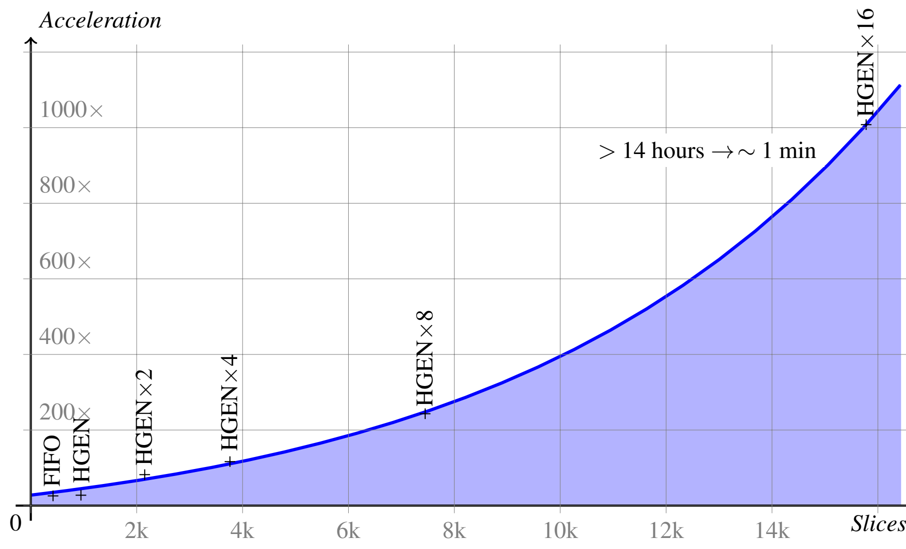

Fig. 2. Relation between the acceleration ratio and the complexity of the verified component.

number of transactions processed per second. Note that this time linearly depends on the number of tested transactions. Also note that the crosspoint with the x axis is the break-even number of transactions. Fig. 4 shows the amount of additional transactions that could be verified in given time when the acceleration is used. The graphs in these figures show values that include the build time of the firmware, which we consider fair for the case when the system is being verified after some change in its internals. However, if the tests run in parallel, the overhead of building the firmware drops. Moreover, for the case when only new test scenarios are added, the build time overhead does not apply and the acceleration is advantageous from the very beginning.

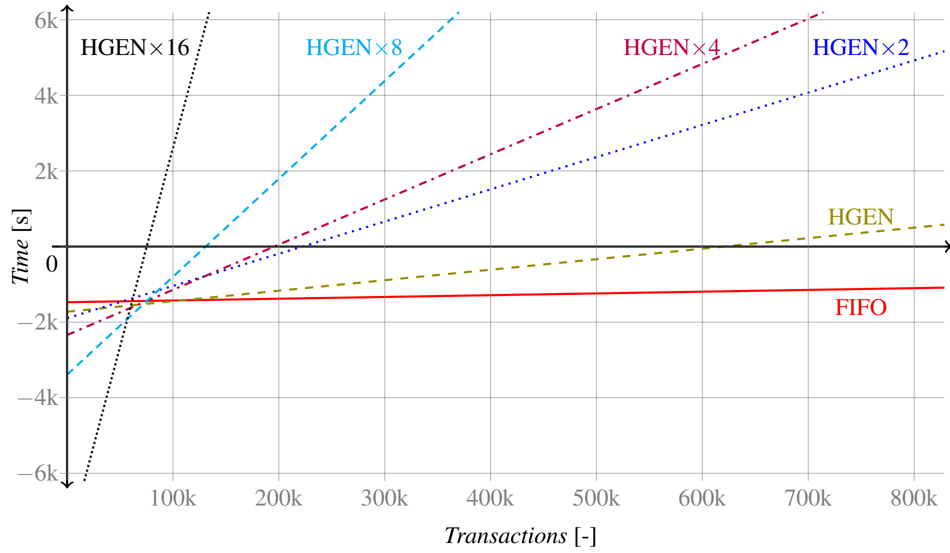

Fig. 3. The time saved for verification of given number of transactions for the accelerated version.

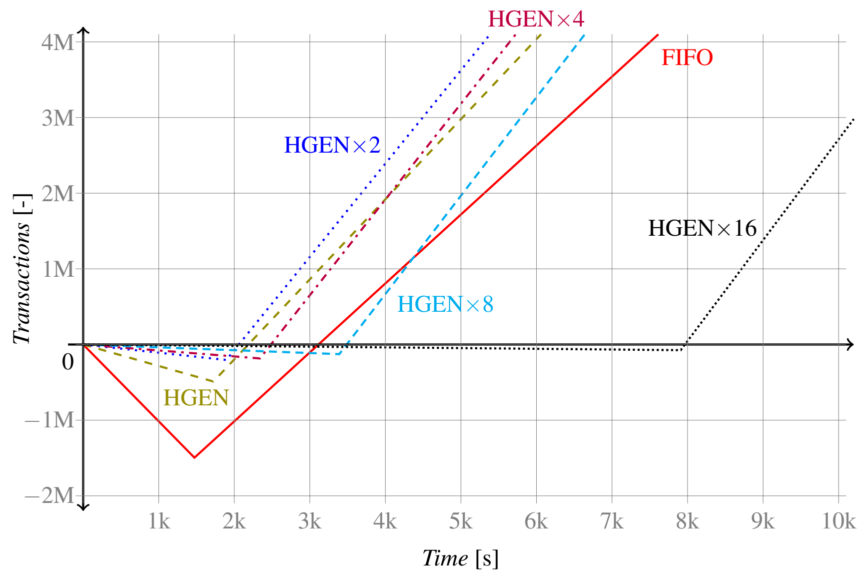

Fig. 4. The number of additional transactions verified for given time for the accelerated version.

4 Conclusion

We presented an open framework for FPGA-accelerated functional verification of hardware systems, which is to the best of our knowledge currently the only open and free solution available. The framework allows users to easily accelerate SystemVerilog testbenches by moving the verified DUT from the simulator into FPGA. The components of the framework can be easily incorporated as blocks into testbenches that use OVM, UVM or any other transaction-based verification methodology. The experiments and their results show that by mapping the RTL logic into an FPGA instead of using a software simulator the acceleration ratio of over 1,000 can be achieved.

In the future, we wish to continue improving HAVEN. As currently only manual creation of Assertion Checkers is supported, we believe that implementing (and perhaps improving) the procedure proposed in [4] for synthesis of SVA assertions, both on interfaces and inside systems, is a reasonable step. The results of our experiments show that another challenging issue is hardware-accelerated generation of test vectors, which requires solving often quite complex constraints. In order to comply with current industry standards, providing the SCE-MI [6] interface for the accelerated testbench is desirable. We welcome collaboration on any of these issues and hope that the community can benefit from our contribution.

Acknowledgements This work was supported by the Czech Science Foundation (projects P103/10/0306 and 102/09/1668), the Czech Ministry of Education (projects COST OC10009 and MSM 0021630528), Reduced Certification Costs Using Trusted Multi-core Platforms project, Artemis JU, RECOMP #100202 and the BUT FIT project FIT-11-S-1.

References

1. Mentor Graphics. Veloce. 2011. http://www.mentor.com/products/fv/emulation-systems/veloce/
2. 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.
3. A. Schwarztrauber. SEmulation: Use Your Emulation Board as a Hardware Accelerator for ModelSim SE. In *Verification Horizons*, 5:31–34, Mentor Graphics, 2009.
4. S. Das, R. Mohanty, P. Dasgupta, P. P. Chakrabarti. Synthesis of System Verilog Assertions. In *Proc. of DATE'06*, EDAA, 2006.
5. B. Jenkins. The Lookup2 hash algorithm. http://burtleburtle.net/bob/c/lookup2.c
6. Accelera Interface Technical Committee. SCE-MI. 2011. http://www.accellera.org/activities/itc/

A FrameLink

FrameLink is a synchronous point-to-point frame-oriented protocol for communication between hardware components, originally developed at the Liberouter[4] project. Communication over FrameLink consists of frames (delimited by signals `SOF_N` and `EOF_N`, which stand for *start of frame* and *end of frame* respectively), which can further be composed of several parts (delimited by signals `SOP_N` and `EOP_N`, which stand for *start of part* and *end of part* respectively). The source and the destination of one FrameLink connection need to share the same `CLOCK` and `RESET` signals. Note that all control signals are active in `0`.

The actual data transfer takes place when both the source and the destination are ready to communicate, which is signalled by their `SRC_RDY_N` and `DST_RDY_N` signals; if both of these signals are active, the source sends the data and the destination receives it. More detailed description of FrameLink signals follows (`src` means that the signal is driven by the source and `dst` means that it is driven by the destination component):

- **DATA[DATA_SIZE-1:0]** (`src`): Vector of signals that bear data, $\texttt{DATA_SIZE} \in \{8, 16, 32, 64, 128, \ldots\}$.
- **REM[REM_SIZE-1:0]** (`src`): Sets validity of bytes in `DATA`, i.e., defines the address of the last valid byte. This signal is valid only when the control signal `EOP_N` is active. $\texttt{REM_SIZE} = \log_2(\texttt{DATA_SIZE}/8)$.
- **SOF_N** and **EOF_N** (`src`): Delimit the start and end of a frame.
- **SOP_N** and **EOP_N** (`src`): Delimit the start and end of a frame part.
- **SRC_RDY_N** (`src`): Signals that the source is ready to send data.
- **DST_RDY_N** (`dst`): Signals that the destination is ready to receive data.

B Example of Use

This section gives examples of the use of the accelerated version of HAVEN.

B.1 Assertion Checker

Thanks to the assertion analysis, any detected discrepancy in the observed behaviour results in an error, which is reported near the origin of the functional defect. In the non-accelerated version of HAVEN it is recommended to take advantage of all possibilities offered by a simulator to explore the issue. In ModelSim, an assertion error is directly detectable from the simulation waveform as illustrated in Fig. 5. The origin of the failure can be examined in detail in the assertions window (Fig. 6).

[4] `http://www.liberouter.org`

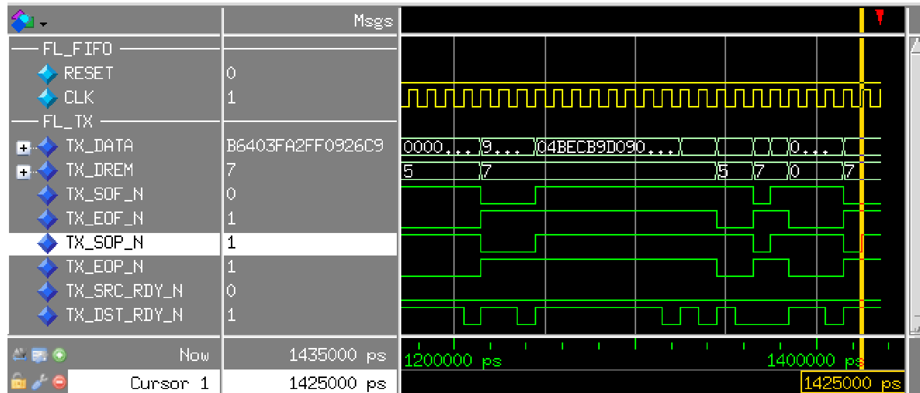

Fig. 5. Assertion failure in the waveform of the ModelSim simulator.

Name	Assertion Type	Failure Count
/RX/assert__RESETSRC	Concurrent	0
/RX/assert__SOFSOP	Concurrent	0
/RX/assert__EOFEOP	Concurrent	0
/RX/assert__NoDataAfterEOP	Concurrent	0
/RX/assert__EOPMatchSOP	Concurrent	0
/RX/assert__EOFMatchSOF	Concurrent	0
/TX/assert__RESETDST	Concurrent	0
/TX/assert__SOFSOP	Concurrent	1
/TX/assert__EOFEOP	Concurrent	0

Fig. 6. The list of assertions in the assertions window of the ModelSim simulator.

Independently of the simulator and even in the accelerated version, HAVEN prints a short report to the transcript file with the description of circumstances that led to the assertion failure, the time of the detected discrepancy and the sequence number of the affected transaction. In the accelerated version of HAVEN this information is provided by Assertion Checker. Given example demonstrates an erroneous situation during the verification of FrameLink FIFO with a deliberately introduced bug. The reports obtained from the non-accelerated version (Fig. 7) and from the accelerated version (Fig. 8) of HAVEN show that the failure is invoked by the violation of FrameLink protocol when the signal `SOF_N` is not active at the same time as the signal `SOP_N`, which means that there is some data that does not belong to any part of the FrameLink frame (which is forbidden). Note that the reported times of failures differ because whereas the simulator reports an assertion failure as soon as it happens, the created hardware Assertion Checker reports all assertion failures only once for each erroneous frame.

```
############ TEST CASE 1 ############
#
# START TIME: Tue Aug 23 13:54:43 CEST 2011
# ** Error: TX_SOF_N is not active in the same time as TX_SOP_N.
#    Time: 1435 ns Started: 1435 ns  Scope: testbench.TX File:
# ------------------------------------------------------------
# --  TRANSACTION TABLE
# ------------------------------------------------------------
# Size:              28
# Items added:       53
# Items removed:     25
# ------------------------------------------------------------
```

Fig. 7. Assertion report from the non-accelerated version of HAVEN.

```
############ TEST CASE 1 ############
#
# START TIME: Tue Aug 23 14:57:11 CEST 2011
#
# !!!!!! Assertion error !!!!!!
# Violation of FrameLink protocol at checker:     170
# Time of violation:                             1530 ns
# Violated transaction #:                          26
# Tue Aut 23 14:57:12 CEST 2011
#
# ------------ ASSERTION REPORT ------------
# TX FrameLink Assertion Error: SOF_N without SOP_N
# TX FrameLink Assertion Error: Data between EOP_N and SOP_N
# TX FrameLink Assertion Error: No EOP_N before SOP_N
# ------------------------------------------------------------
```

Fig. 8. Assertion report from the accelerated version of HAVEN.

B.2 Signal observer

For debugging purposes, the accelerated version of HAVEN includes the Signal Observer to ensure adequate signal visibility. Fig. 9 compares the waveform obtained from the simulation of the non-accelerated version with the waveform received from the Signal Observer in the accelerated version.

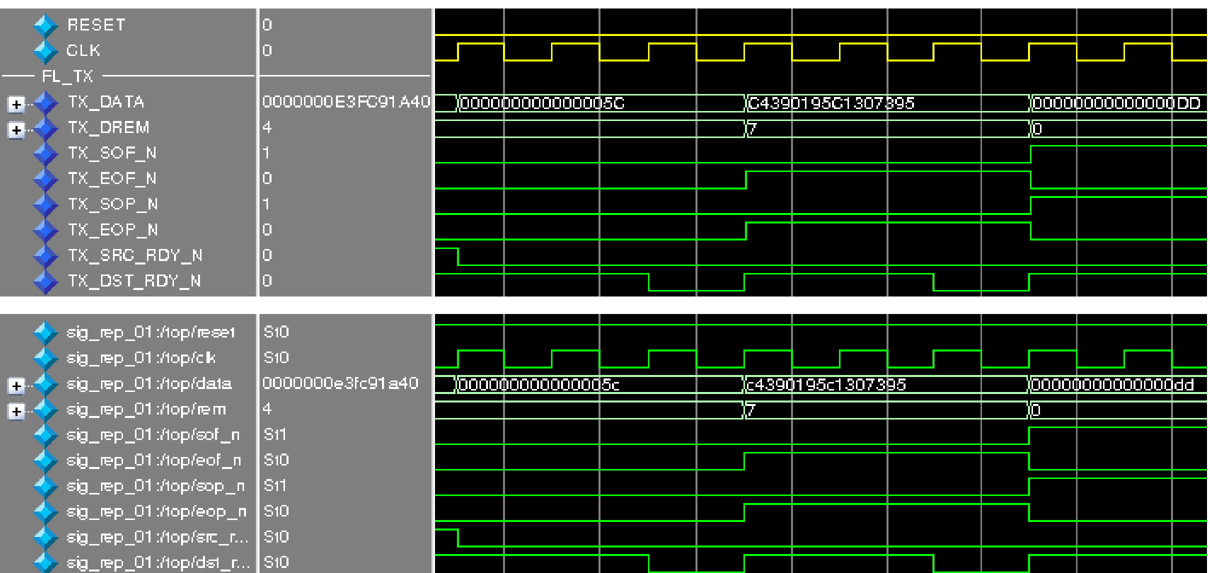

Fig. 9. Simulation waveform (upper part) compared to the waveform obtained from Signal Observer (lower part)