Acceleration of Functional Verification in the Development Cycle of Hardware Systems

Marcela Šimková
Computer Science and Engineering, 1-th class, (full-time study)
Supervisor: Zdeněk Kotásek

Faculty of Information Technology, Brno University of Technology
Božetěchova 2, Brno 612 66

isimkova@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 this paper I present two techniques to target the issue of acceleration. The first one exploits the field-programmable gate array (FPGA) technology for cycle-accurate acceleration of simulation-based verification runs. This approach 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 performed 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. The second acceleration technique is based on utilizing genetic algorithm in coverage-driven verification (CDV) methodology. CDV allows to measure completeness and effectiveness of the verification process, in particular it provides detailed feedback about which features of the design have been appropriately checked. It is assumed that genetic algorithm can accelerate CDV by leading the generation of input test vectors in order to cover areas of the system which were not explored by previously applied tests.

Keywords. Functional Verification, Acceleration.

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 the 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 difficult to be 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. 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 accelerate 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. Moreover, assertions created for functional verification can be used in tools utilizing a formal method called *bounded model-checking*, e.g. Questa [4]. In this tool, static and dynamic verification techniques are applied on the model of the system in order to find a proof that an assertion was never violated by any input vector. All above mentioned features are effective to check system correctness and maximise the efficiency of the overall verification process.

2 Need for Acceleration

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. Therefore, hardware acceleration can be seen as the first progressive step in the whole acceleration process.

However, hardware acceleration itself is not enough. The reason is that applying huge number of input test vectors to the system without any knowledge about the state space exploration of the system cannot check all corner cases and interesting scenarios. On that account the generation of appropriate tests can be driven manually by a verifier who controls coverage results and chooses parameters or a pseudo-random number generator seed according to achieved coverage. Nevertheless, I believe that achieving coverage closure can be fully automated by an intelligent program, e.g. genetic algorithm. Therefore, part of my Ph.D. thesis will be concerned with the interconnection of hardware acceleration and CDV that will be running automatically under the control of a genetic algorithm.

The paper is organized as follows. The next section contains related work focusing on the acceleration of verification runs in emulators and hardware accelerators. The two following sections describe the framework for hardware acceleration of functional verification and the proposal of CDV environment driven by a genetic algorithm.

3 Related Work

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. Currently, there already exist several approaches to acceleration of functional verification. Mentor Graphics' Veloce technology [3] 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 [6] is a system that enables acceleration of simulation of a DUT using FPGA while sacrificing observability of the DUT's signals. Similar approach is presented in the work of Huang *et al* [5] who place the DUT with necessary components to an FPGA but in addition provide limited observability of the DUT's signals.

4 HAVEN: Hardware Accelerated Verification Environment

Building upon my experience with different verification approaches and existing studies dealing with acceleration issues, together with my colleagues I implemented **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, coverage analysis) and enables partial signal observability to achieve appropriate debugging visibility while running in the FPGA what is not implicitly supported in hardware. HAVEN is freely available, *open source*[1] and allows users to run either a *non-accelerated* or an *accelerated* version (at the frequency around 125 MHz) of the same testbench with a cycle-accurate time behaviour.

The framework monitors two types of errors: assertion failures and conflicts against a golden model, 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.

4.1 The Non-accelerated Version vs. the Accelerated Version of HAVEN

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 the verification of complex systems, large number of test vectors or regression testing.

The accelerated version of the framework moves the DUT to a verification environment in the FPGA. As RTL 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. The communication between the software and the hardware part of the verification environment is mediated using a generic protocol. Nevertheless, the readability for verification engineers remains the same and the tests can be assembled at a high level of abstraction without the need to change any hardware-level code.

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

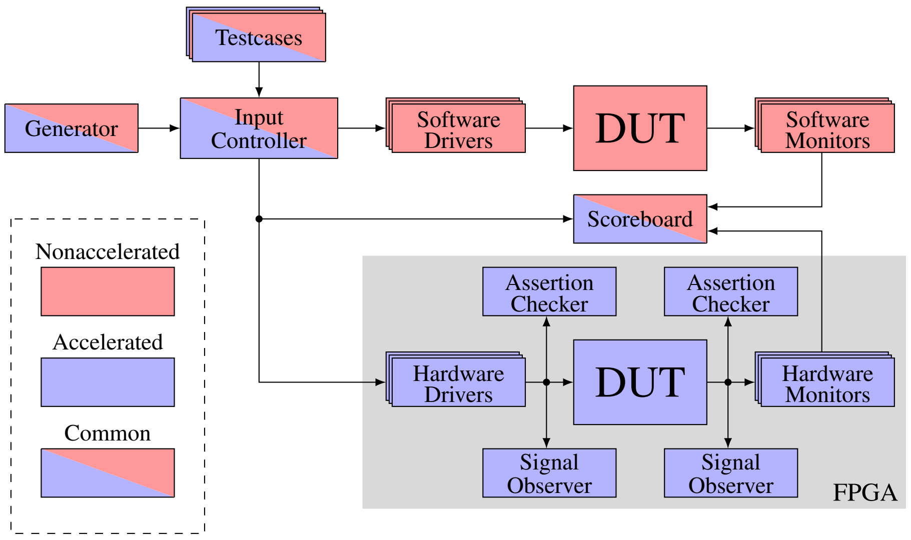

Figure 1: The architecture of the HAVEN framework.

As illustrated in Fig. 1 some components of the framework are shared in both versions. On the other hand, the use of other components strictly depends on the selected version of the framework. The description of these components and details about the architecture of the framework can be found in the paper [1] or in the technical report [2].

4.2 Experimental Results

Experiments were performed using the COMBOv2 LXT155 acceleration card 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 and the clock frequency of the acceleration environment was 125 MHz. Mentor Graphics' ModelSim SE-64 6.6a was used as the SystemVerilog interpreter and in the case of the non-accelerated version also as the DUT simulator.

The performance of HAVEN was evaluated on two hardware components: a simple FIFO buffer and a hash generator (HGEN) which computes the hash value of input data. In order to fully exploit the capabilities of the accelerated version of HAVEN it is necessary to verify a complex system. For this purpose, the systems with 2, 4, 8, and 16 parallelly working HGEN units were built.

The results of measurements are shown in Table 1. The average acceleration ratio of the accelerated version computed from all verification runs for every verified system is given in the row **Acceleration**, whereas the row **Acceleration w/o trans.** represents the average acceleration ratio without the time of transaction generation (as this is the same for both versions). The rows **Slices** and **Slices %** describe the total consumed amount of resources at the FPGA (in Virtex-5 slices and percent respectively); the total number of slices on the used FPGA is 24,320 and the overhead of the communication layer is about 35 % of FPGA resources. The resource consumption of overhead modules (driver, monitor, etc.) is negligible (below 2 %). The row **Build time** gives the time it took to build the firmware of the accelerated version and the row **B-E transactions** holds the number of transactions for which the accelerated version starts to be beneficial (i.e., when the acceleration compensates for the build time of the firmware). The details about the experiments and further discussion can be found in [2].

Table 1: Results of experiments with HAVEN.

Component	FIFO	HGEN	HGENx2	HGENx4	HGENx8	HGENx16
Acceleration	2.38	4.73	12.47	17.11	35.68	142.26
Acceleration w/o trans.	38.25	27.51	81.97	116.39	243.09	1,008.00
Slices	9,362	9,787	11,315	12,938	16,304	22,096
Slices %	38.5	40.2	46.5	53.2	67.0	90.9
Build time [s]	1,473	1,724	1,895	2,340	3,390	7,909
B-E transactions	3,116,000	622,000	222,000	196,000	131,000	75,000

5 Achieving Coverage Closure Using Genetic Algorithm

In functional verification, the correctness of a hardware system is typically evaluated using input test vectors which are applied to the input interface of the verified system and are produced by a pseudo-random number generator. The correct form of these vectors is accomplished using constraints. After inserting these vectors into the system it is possible to monitor the behaviour of the system automatically in a software simulator and check the output values against the golden model. However, without the feedback about which design features specified in a verification plan have been exercised by tests (so-called *coverage*)we cannot say that the system is verified, because we do not know if all intended functions, corner cases or synchronization processes were appropriately checked.

To target this issue, functional verification allows defining coverage points in the system in order to monitor the intended functionality of the system according to the specification. For this purposes, there exist different types of coverage which can be measured, e.g. functional coverage, code coverage, toggle coverage, statement coverage, FSM coverage, etc. After coverage points are defined, the simulator offers coverage analysis and produces exhaustive statistics about which coverage points were hit during all verification runs. If there are holes in coverage even after huge number of verification runs, the verification effort is directed to preparation of suitable test scenarios which will be able to cover unexplored areas in the design. This approach is called *coverage-driven verification*. The idea of covering holes is demonstrated in Fig. 2. However, as the complexity of current hardware systems rises, it is challenging to accomplish this task manually. For this reason, every technique which helps to achieve maximum coverage automatically in a reasonable time is highly demanded.

According to my knowledge, there already exist attempts to automate CDV, e.g. by data mining or by adapting some formal verification techniques [7, 8]. These are often inbuilt in proprietary industry tools and typically require in-depth knowledge about the design architecture. Unfortunately, producers of these tools are usually not loquacious about the techniques their tools use to achieve the maximum

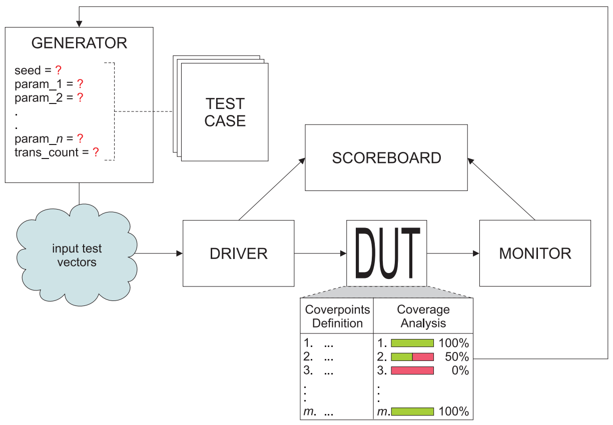

Figure 2: Coverage-driven verification.

coverage. In my Ph.D. thesis I wish to address the issue of automation of achieving maximum coverage using an intelligent program, e.g. genetic algorithm.

6 Conclusions

In this paper, two techniques for acceleration of functional verification were presented. The first one is based on hardware acceleration in a framework HAVEN which is to the best of my knowledge the only open and free solution available. The framework allows users to easily accelerate verification testbenches by moving the verified DUT from the simulator into FPGA. 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.

6.1 The goals of the future research and the Ph.D. thesis

In the future I wish to utilize the second proposed acceleration technique which uses a genetic algorithm in the automation of CDV. In my Ph.D. thesis I plan to propose not only different strategies for acceleration of functional verification but also a technique that builds proper sets of input test vectors with high level of coverage for regression testing in pre-silicon and post-silicon phase of a system design. Moreover, I want to apply these techniques to improve the quality of the fault-tolerant system design and to accelerate testing of final fault-tolerant systems. In order to do this I need to evaluate the connection between the set of input test vectors produced by functional verification and input test vectors produced by other tools, e.g. ATPG, to find the relation between the functional and fault coverage.

Acknowledgment

This work was supported by The Ministry of Education, Youth and Sports (project LD12036), Brno Ph.D. Talent Scholarship Programme, the BUT FIT project FIT-S-11-2 and the project GD102/09/H042.

References

[1] M. Šimková, O. Lengál, M. Kajan. HAVEN: An Open Framework for FPGA-Accelerated Functional Verification of Hardware. To appear in Proc. of Haifa Verification Conference 2011 — HVC'11, Haifa, Israel, volume 7261 of LNCS, Springer-Verlag.

[2] M. Šimková, O. Lengál, 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] Mentor Graphics. Veloce. 2011. `http://www.mentor.com/products/fv/emulation-systems/veloce/`

[4] Mentor Graphics. Questa. 2012. `http://www.mentor.com/products/fv/questa/`

[5] 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*, p. 297–300, IEEE, 2011.

[6] A. Schwarztrauber. SEmulation: Use Your Emulation Board as a Hardware Accelerator for ModelSim SE. In *Verification Horizons*, p:31–34, Mentor Graphics, 2009.

[7] Li-C. Wang, Pouria Bastani, Magdy S. Abadir Design-silicon timing correlation: a data mining perspective In *Proc. of DAC '07*, pages 384 – 389, ACM New York, NY, USA, 2007, ISBN: 978-1-59593-627-1.

[8] Synopsys. VCS. 2012. `http://www.synopsys.com/Tools/Verification/FunctionalVerification/Pages/VCS.aspx`