FPGA Prototyping and Accelerated Verification of ASIPs

Jakub Podivinsky, Marcela Simkova, Ondrej Cekan, Zdenek Kotasek
Faculty of Information Technology, Brno University of Technology
Bozetechnova 2, 612 66 Brno, Czech Republic
Tel.: +420 54114-{1361, 1362, 1361, 1223}
Email: {ipodivinsky, isimkova, icekan, kotasek}@fit.vutbr.cz

***Abstract*—In current SoC verification, the trend is to create verification solutions that are tailored to specific issues in SoC or to specific architectures. The reason is that the complexity of these systems makes it difficult to use general verification approaches such as formal or simulation-based verification. This paper presents a solution that is targeted to one particular area - Application-Specific Instruction-Set Processors (ASIP) and multi-processor systems containing several ASIPs. We propose automated FPGA prototyping and accelerated verification of these systems while the accelerated verification environment corresponds to the principles of UVM (Universal Verification Methodology) therefore can easily be integrated. Automated generation of verification environments and acceleration of verification runnning on a real hardware platform makes this solution very unique and beneficial, not only in speed, but also in debugging specific hardware issues.**

Keywords*—*UVM, Acceleration, FPGA Prototyping, ASIP.

I. Introduction

The current embedded systems, such as *Systems on Chip* (SoC), *multi-processor systems* (MPSoC) or equipment for *Internet of Things* (IoT), are more and more complex. They usually consist of one or more processors (either *General Purpose Processors* (GPPs) or *Application Specific Instruction-set Processors* (ASIPs)) and various types of peripherals. An important phase in the development cycle of these systems is the verification of their functionality. Various approaches for verification currently exist, such as formal verification, assertion-based verification or simulation-based verification (also called functional verification). But in general, functional verification is easier to apply for hardware engineers as they are familiar with simulation tools and this approach does not require a deep knowledge of formal specifications. Moreover, standard languages, methodologies and libraries were defined for functional verification. The most commonly known are the SystemVerilog IEEE language standard [1], Universal Verification Methodology (UVM) [2] and the open-source UVM library (with all the basic components of verification environments). They work well for unit level verification, but for processors, SoC or MPSoC, they do not scale well. The reason is not only in the complexity of these systems, but also the fact that software embedded into processors must be taken into account as well [3], [4]. Moreover, their verification is time consuming and this can lead to undesirable prolongation of the time to market.

Therefore, because of its complexity, it seems to be the current state-of-the-art in SoC verification to come with a verification solution that is adjusted to SoC (digital vs. analogy, verification IPs, graph-based IP connections, etc.); or their application domain (e.g. multimedia, DSP applications, smart devices, etc.) and is often connected to the development tool of these systems. For example, Breker [5] introduces a graph-based approach to functional verification. Users capture with graphs the IP level scenarios as nodes and connections make the SoC level scenario. Cadence [6], Synopsys [7] or Mentor Graphics [8] provides verification IPs for more than 40 communication protocols and 60 memory interfaces in order to facilitate SoC verification. Duolog [9] focuses on IP integration problems and generates UVM verification environments from interface-based executable specification. Codasip company [10] provides Codasip Framework that is targeted to ASIP and MPSoC development and offers automated generation of UVM verification environments for these systems that are customized for a class of applications that run on their embedded processor(s).

In our previous work we focused on automated generation of UVM verification environments for ASIPs and MPSoC from their high-level description in *architecture description language* (ADL) [11]. This feature is now integrated into the Codasip Framework. Moreover, we designed an open-source framework HAVEN for FPGA acceleration of simulation-based verification of various systems [12].

Our current research is a continuation of our previous work. We designed and implemented a new feature for automated FPGA prototyping and accelerated verification of ASIPs and MPSoC. We realised that simulation-based verification of ASIPs and MPSoC is valuable, but it runs slowly when we need to evaluate thousands of embedded software applications. Therefore, in the accelerated version of verification we replicate the main principle of HAVEN and move the *Device Under Test* (DUT), which is ASIP or MPSoC, from the software simulation into an FPGA. All other parts such as loading package applications, a running reference model and scoreboarding, remain in the software. If a bug is detected in the accelerated version, we can use the pure software version of the verification environment running in the simulator (the non-accelerated version) for easier debugging of the problem. Another important benefit of using FPGA is that the ASIP prototype will run on real hardware. This helps us to uncover bugs, which are related to the placement of the design into real hardware and which are not detectable in simulation.

Regarding acceleration of verification in general (not necessarily for processors), there already exist some commercial

978-1-4799-6780-3/15 $31.00 © 2015 IEEE
DOI 10.1109/DDECS.2015.33

solution which are quite similar to our work. Mentor Graphics Veloce technology [13] accelerates simulation by synthesising the DUT and placing it into a proprietary emulator. Emulation and acceleration of verification also offers Cadence company in their Cadence Palladium Series [14]. However, our solution is different as it aims exclusively at verification and FPGA-prototyping of ASIPs and MPSoC. But as mentioned above, targeting verification to a specific domain can be much more precise. At the same time, we support system-level verification as we are able to verify not only the hardware architecture of ASIPs, but also various software applications that are executed on them.

This paper is organized as follows. The architecture of the accelerated verification environment is described in Section II. The case study in Section III shows the process of generating the verification environment for a selected ASIP (the accelerated and non-accelerated version) as well as experimental results for different verification runs. Section IV summarizes the results and proposes our plans for future research.

II. Architecture of the Accelerated Environment

In order to get an idea on how the accelerated verification environment may look like, we prepared a simple demonstration model. The verified system (DUT) is a simple MPSoC consisting of two ASIPs: ASIP1 and ASIP2. ASIP1 receives input data and functions as a pre-processor for ASIP2, ASIP2 sends results to its output ports. But of course, DUT can be represented by any other ASIP or MPSoC.

The non-accelerated UVM verification environment in Figure 1 (in SystemVerilog) for the demonstrated MPSoC is generated automatically in Codasip Framework. The accelerated verification environment in Figure 2 is derived from the non-accelerated version. It should be noted that almost all UVM components are moved into the FPGA, except for the Reference Model and Scoreboard. Nevertheless, we aim at designing consistent verification architecture in FPGA too. Therefore, UVM Agents and their inbuilt components are just replaced by HW Agents. We believe that consistent FPGA verification architecture is beneficial not only for the automated generation of the accelerated version, but it also remains understandable for verification engineers.

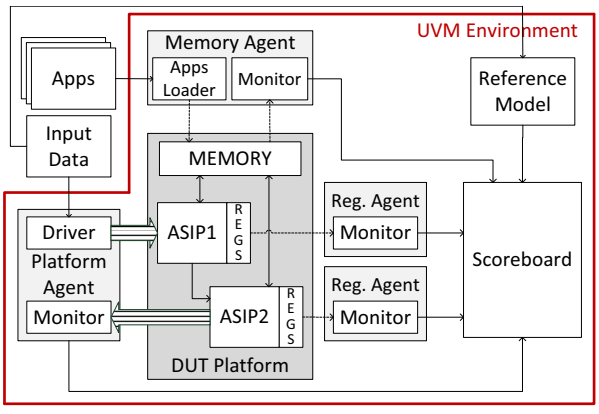

Fig. 1. The architecture of the non-accelerated verification environment.

In the accelerated version, UVM testbench, Reference Model and Scoreboard are running in software simulation and the remaining parts are running in FPGA. Communication between the software and hardware parts of the verification environment is accomplished using the framwork HAVEN (for more details please see [12]). More details about the components of both parts are provided below in the following subsections.

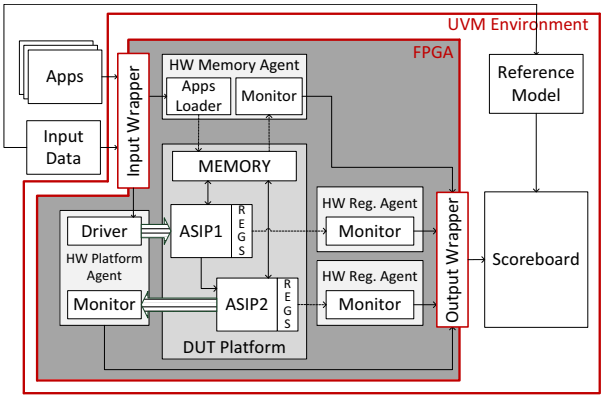

Fig. 2. The architecture of accelerated verification environment.

A. Software Part of the Verification Environment

The main components of the software part are Reference Model and Scoreboard. Reference Model is generated automatically from the high-level specification. Scoreboard compares results of the Reference Model to the results of DUT (received from the hardware part through the Output Wrapper component). In particular, we compare the content of memories and register fields when the specific data set is processed, and we continuously check data from the output ports. The role of Input Wrapper is to send applications (they are loaded to ASIPs and define their functionality) and input data.

The applications are obtained from our designed and implemented stimuli generator which is also at the forefront of our overall research plan. The stimuli generator is especially expected to be used in functional verification. The conception of the stimuli generator is designed for versatile purposes. The aim of the generator is changing its input parameters and achieving its different behavior and thus its different outputs. The generator is based on constraint solving [15] and it takes problem specification as an input. For our purposes of the ASIPs verification, the generator takes the assembly instruction set specification and constraints for this instruction set as the input. Instruction set defines what is to be generated and constraints defines how it has to be generated. When generator is operating, it must deal with numerous conditions and restrictions (constraints). At a processor, it is needed to deal mainly with jump instructions, memory access instructions and latencies for each instruction. Thanks to the constraints, there are reduced possible invalid outputs. The output from the generator is an assembly program (stimulus) which is transformed into machine code and passed into the Input Wrapper. The basic principle of the presented generator are show in Figure 3.

The key part of the generator is the definition of the constraints and their fast interpretation. We defined 20 constraints for valid generating an assembly code. For example absolute jump instructions need 6 types of constraints for ensuring valid and unique label generation in whole program. The generator does not work with semantics of instructions. This allows to focus on more application domains with different stimuli. Therefore the generator is not limited only for processors and can be used in many areas. We are able to generate valid assembly programs for RISC (Reduced Instruction Set Computing) and VLIW (Very Long Instruction Word) processors so far. Some more information about the presented generator is in [16].

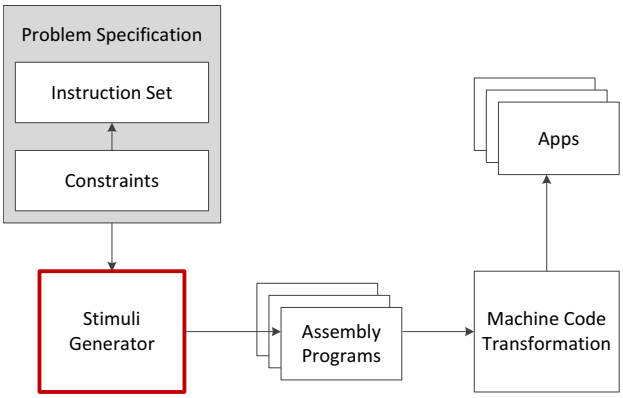

Fig. 3. The basic principle of the generator.

B. Hardware Part of the Verification Environment

Hardware components are currently implemented manually in VHDL. During the following months we plan to generate them automatically in VHDL/Verilog from the high-level specification (similarly as we generate UVM verification environments in the non-accelerated version). A short description of the main hardware components follows.

Hardware Agents are similar to UVM Agents and their main components are Drivers and Monitors. Drivers drive input ports of DUT and Monitors, which on the other hand, collect data from output ports. In Figure 2 you can see the Hardware Memory Agent, two Hardware Register Agents and the Hardware Platform Agent. The Hardware Memory Agent is connected to the main memory. It contains the Driver called the Application Loader that drives the loading of applications into the program part of the memory at the beginning of computation. The second component is the Monitor that takes an image of the memory at the end of computation and sends it to the software Scoreboard for comparison to reference results. The Hardware Register Agent contains only the Monitor that takes an image of register fields at the end of computation and sends it to the software Scoreboard. The Hardware Platform Agent is active during the whole computation; it contains the Driver that during the computation stimulates input ports of ASIP1 with data and Monitor that sends the valid output data of ASIP2 to the software Scoreboard.

III. The Case Study

We performed our experiments and measures with the DUT consisting of one ASIP called Codix RISC [17]. The aim of these experiments was to evaluate and compare the performance of the non-accelerated version of verification to the FPGA-accelerated version.

The non-accelerated verification environment is generated automatically in Codasip Framework. All parts are in SystemVerilog (except of DUT in VHDL or Verilog) and are simulated in Mentor Graphics ModelSim SE-64 10.0c simulator on the server with two quad-core Intel Xeon E5620@2.40 GHz processors and 24 GiB of RAM. The accelerated verification environment contains the DUT, Hardware Platform Agent, Hardware Monitor Agent and Hardware Register Agent on the FPGA site (Xilinx Virtex-5 FPGA) and Scoreboard and Reference Model on the software site (simulated again in ModelSim on the server). The amount of consumed FPGA resources (slices) is the following: 1,428 (5.8%) for the Codix RISC processor and 1,669 (6.9%) for the hardware verification environment.

Results of these experiments are depicted in Table I. We have the measured verification time (the accelerated and the non-accelerated version) for a different number of application programs. Moreover, the acceleration ratio was computed.

TABLE I. The acceleration ratio and the run time of verification.

Number of programs [-]	Run time		Acceleration ratio [-]
	Non-accelerated [s]	FPGA-accelerated [s]	
500	3458	2010	1.719
1,000	6841	3974	1.721
2,000	13634	7917	1.722
4,000	27208	15784	1.723
6,000	40845	23682	1.724
8,000	54384	31451	1.729
10,000	67965	39372	1.726

The measured values are also presented in the graphs of Figure 4 and 5. The acceleration ratio is on average 1,7x and slowly grows up with the number of the evaluated test programs. Because we expected better results, we have performed some specific additional measurements and have identified candidates for further improvements.

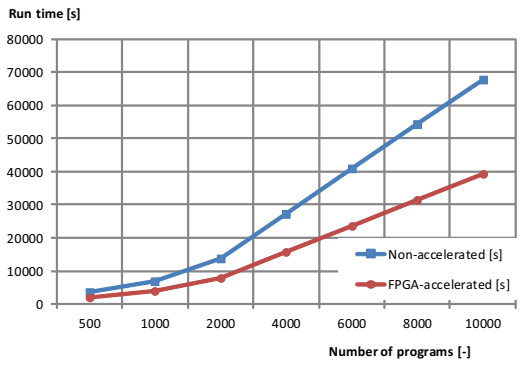

Fig. 4. The relation between the runtime of the non-accelerated and the FPGA-accelerated verification and the number of processor programs.

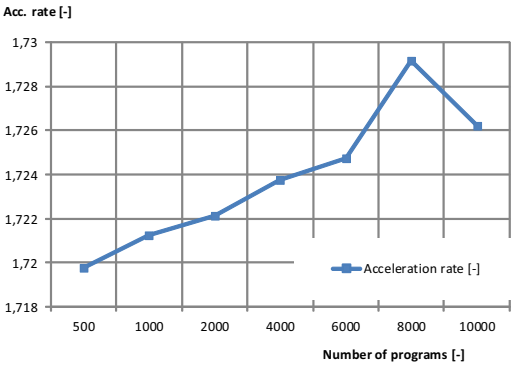

Fig. 5. The relation between the acceleration ratio and the number of processor programs.

The software site of the verification environment is still too complex and the runtime of the accelerated verification depends on the speed of the SW verification environment. The graphical reprezentation of this analysis is shown in Figure 6. The DUT consumes longest time in the non-accelerated verification. But on the other hand, the time consumption of DUT in the FPGA-accelerated verification is shortest. In this case, the runtime of the whole verification is based on the time consumption of the SW verification environment which is the same for both the FPGA-accelerated and the non-accelerated version. There we see a space for possible improvments.

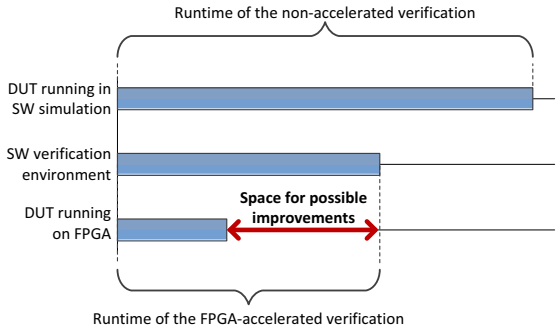

Fig. 6. The graphical representation of the time consuption of the SW verification environment, the simuletd DUT and FPGA-accelerated DUT.

The second problem is that we have executed a huge number of small applications, but only with the basic data sets. For precise verification, every application should be evaluated with more transaction data. The computation burden will be higher, so the acceleration will be more beneficial.

IV. Conclusions and Future Research

In this paper, the environment for FPGA-prototyping and accelerated verification of ASIPs and MPSoC were presented. The case study shows that by means of acceleration on FPGA we are able to detect errors faster (1,7x) and debug not only the software model but directly the hardware prototype on FPGA. As for the great advantage of the accelerated verification environment we see its correspondence to UVM, so it is easily understandable for verification engineers. However, the acceleration ratio was not so good as we expected so we should find a way how to optimize the SW verification environment even more so it will not be a bottlenck in the whole system.

In our future research we intend to interconnect the accelerated verification environment with our fault injector that also operates on FPGA. In this way we will connect our research in the verification area to our research in fault-tolerant systems design [16]. The aim is to create a robust platform for validation of FT methodologies in which the introduced stimuli generator will be also applied.

Acknowledgment

This work was supported by the following projects: National COST LD12036 - ”Methodologies for Fault Tolerant Systems Design Development, Implementation and Verification”, project Centre of Excellence IT4Innovations (ED1.1.00/02.0070), EU COST Action IC1103 - MEDIAN - Manufacturable and Dependable multIcore Architectures at Nanoscale and BUT project FIT-S-14-2297.

References

[1] *IEEE Std. 1800-2005, IEEE Standard for SystemVerilog— Unified Hardware Design, Specification, and Verification Language*, 2005. [Online]. Available: http://ieeexplore.ieee.org/xpl/standardstoc.jsp?isnumber=33132&isYear=2005

[2] S. Rosenberg and K. Meade, *A practical guide to adopting the universal verification methodology (UVM)*. Cadence Design Systems, 2013.

[3] R. Backasch, C. Hochberger, A. Weiss, M. Leucker, and R. Lasslop, “Runtime verification for multicore soc with high-quality trace data,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 2, p. 18, 2013.

[4] R. Drechsler, C. Chevallaz, F. Fummi, A. J. Hu, R. Morad, F. Schirrmeister, and A. Goryachev, “Future soc verification methodology: Uvm evolution or revolution?” in *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 372.

[5] Breker. (2014, Dec.) Treksoc. [Online]. Available: http://www.brekersystems.com/products/treksoc/

[6] Cadence. (2014, Dec.) Verification IP. [Online]. Available: http://ip.cadence.com/ipportfolio/verification-ip

[7] Synopsys. (2014, Dec.) Verification IP. [Online]. Available: http://www.synopsys.com/Tools/Verification/FunctionalVerification/VerificationIP/Pages/default.aspx

[8] Mentor Graphics. (2014, Dec.) Mentor verification IP. [Online]. Available: http://www.mentor.com/products/fv/verification-ip

[9] Duolog. (2014, Dec.). [Online]. Available: http://www.duolog.com/products/

[10] Codasip. (2014, Dec.) Codasip framework. [Online]. Available: http://www.codasip.com

[11] M. Šimková, Z. Přikryl, Z. Kotásek, and T. Hruška, “Automated functional verification of application specific instruction-set processors,” in *Embedded Systems: Design, Analysis and Verification*. Springer, 2013, pp. 128–138.

[12] M. Šimková and O. Lengál, “Towards beneficial hardware acceleration in haven: evaluation of testbed architectures,” in *Hardware and Software: Verification and Testing*. Springer, 2013, pp. 266–273.

[13] Mentor Graphics. (2014, Dec.) Veloce2. [Online]. Available: http://www.mentor.com/products/fv/emulationsystems/veloce

[14] Cadence. (2014, Dec.) Palladium xp series. [Online]. Available: http://www.cadence.com/products/sd/palladium_xp_series

[15] L. Kotthoff, “Constraint solvers: An empirical evaluation of design decisions,” *CoRR*, vol. abs/1002.0134, 2010. [Online]. Available: http://arxiv.org/abs/1002.0134

[16] J. Podivinsky, O. Cekan, M. Simkova, and Z. Kotasek, “The evaluation platform for testing fault-tolerance methodologies in electro-mechanical applications,” in *Digital System Design (DSD), 2014 17th Euromicro Conference on*. IEEE, 2014, pp. 312–319.

[17] Codasip. (2014, Dec.) Codix RISC. [Online]. Available: https://www.codasip.com/products/codixrisc/