HARDWARE ACCELERATED FUNCTIONAL VERIFICATION

Marcela Šimková
Master Degree Programme (2), FIT BUT
E-mail: xsimko03@stud.fit.vutbr.cz

Supervised by: Michal Kajan
E-mail: ikajan@fit.vutbr.cz

Abstract: Functional verification is a widespread technique for checking whether a hardware system satisfies a given correctness specification. The complexity of modern computer systems is rapidly rising and the verification process takes significant amount of time. It is a challenging process to find appropriate acceleration techniques. We introduce a strategy for acceleration of functional verification using FPGAs by targeting special components of the verification environment to the FPGA.

Keywords: functional verification, testbench, SystemVerilog, hardware acceleration, FPGA

1 INTRODUCTION

For verification of computer hardware a variety of options is available to engineers. *Simulation* is used especially in the early phase of implementation and debugging of base system functions. A more efficient approach based on simulation is *functional verification* with advanced features like self-checking mechanisms, coverage driven verification, constrained-random stimulus generation or assertion based verification. In order to achieve completeness of verification process some *formal techniques and tools* can be used. It has been observed that verification becomes the major bottleneck in development of hardware systems, as it takes up to 80 % of the overall development cost and time. Simulation-based approaches suffer from the fact that software simulation of inherently parallel hardware is extremely slow when compared to the speed of real hardware. The aim of this work is to design and implement a framework that exploits the inherent parallelism of hardware designs to accelerate functional verification of these designs in a FPGA.

2 SYSTEMVERILOG FUNCTIONAL VERIFICATION

The continuous growth in the complexity of hardware designs requires a modern, systematic and automated approach for creating functional verification test environments (testbenches). To deal with this issue the SystemVerilog programming language was designed. With increasing popularity of this language several verification methodologies such as OVM, VMM or UVM were created to help verification engineers build interoperable, scalable and reusable verification environments and components in SystemVerilog. The generic nature of these methodologies and transaction-based communication among their subcomponents make it possible to transparently move these subcomponents to a specialized hardware, while maintaining the same level of readability to verification engineers.

3 ANALYSIS

Nowadays we can observe significant effort of many researchers and companies to increase efficiency and speed of simulation and verification techniques. The notable examples are: Mentor Graphics's Veloce technology [1], FPGA prototyping of ASIC designs [2], or acceleration of simulation using an emulation board [3]. Our goal is to develop an acceleration framework that would enable writing high-level functional verification testbenches in SystemVerilog, which could be easily accelerated without the use of expensive specialized emulation boards as needed in the abovementioned approaches. To achieve high level of acceleration the most challenging part of this process is to map some

components of the verification environment into a reconfigurable hardware platform (FPGA) and to control communication between hardware and software parts. We chose the NetCOPE platform on the ComboV2 acceleration card [4] as the implementation environment and specially designed NetCOPE protocol for communication and data transport between hardware and software parts. The framework allows the user to run either non-accelerated or accelerated version of the same testbench, even with the same time behaviour.

- **Non-accelerated Version** — presents a similar approach as used in verification methodology libraries. This framework is useful in the initial phase of the verification process when debugging basic system functions. In this phase it is desirable to have a quick access to values of all signals of the design and to monitor the verification process in a simulator.
- **Accelerated Version** — moves the verified component 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. Behavioural parts of the testbench, such as planning of test sequences, generation of constrained-random stimuli, monitoring coverage reports, and scoreboarding, remain in the software simulator. If a bug occurs the user can start the failing verification scenario with the same time behaviour in the software version for comfortable debugging.

The components of the verification environment differ according to the selected version of the framework as illustrated in Figure 1.

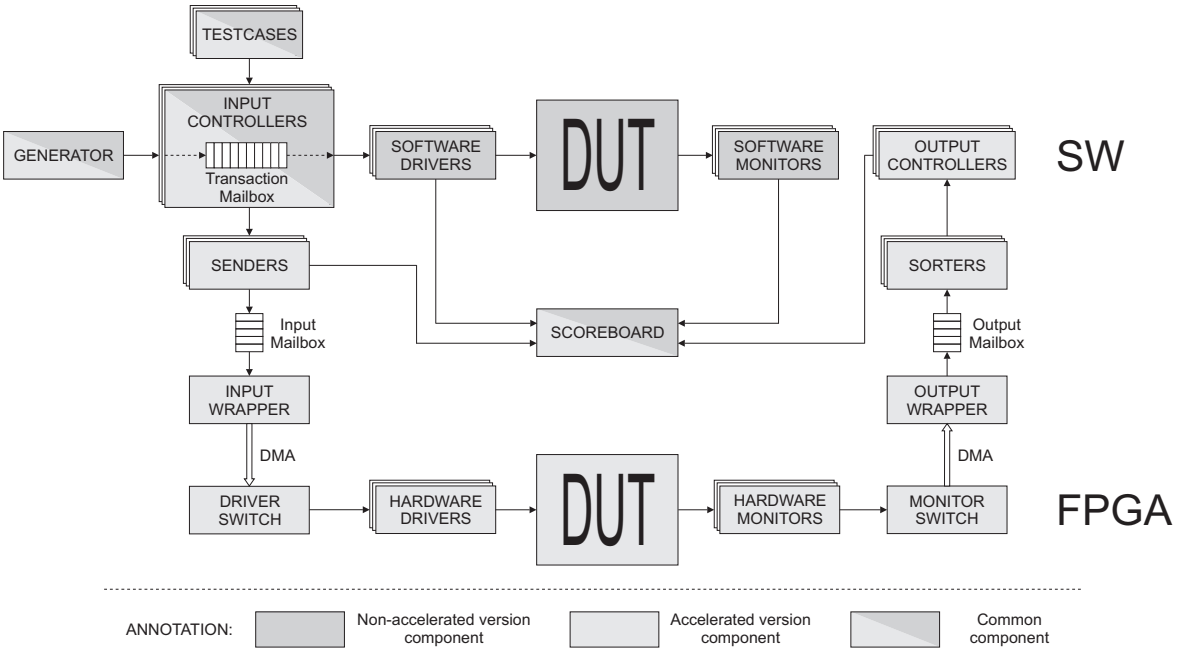

Figure 1: Framework verification environment.

- *Design Under Test (DUT)* is either run in simulator (for the non-accelerated version) or synthesized and placed in the FPGA (for the accelerated version).
- *Testcases* are written by the user, who creates verification environment from available or her own components, sets generics and parametres like transaction count, verification framework version, delays, etc. The Testcase contains instructions for Input Controller to manage constrained-random stimulus generation, sending of directed stimuli or synchronization between transactions. We support two types of transactions: data transactions, which are either randomly generated or direct, and control transactions (`start, wait, stop`).
- *Input Controllers* are created for selected interface protocols. Every controller interprets instructions from testcases, calls Generator if random stimuli are required, and hands over the control to Sofware Driver (in non-accelerated version) or to Sender (in accelerated version) for following processing of transactions.

- *Generator* produces constrained random stimuli and delays between and inside transactions.
- *Sofware Drivers* receive instructions from assigned Input Controllers and data transactions through Transaction Mailbox. The driver breaks transactions down into individual signal changes and supplies them on input interface of the simulated DUT. The copy of the data transaction is sent to Scoreboard. The driver may also inject errors or add delay parts.
- *Software Monitors* drive simulated DUT's output interfaces and observe signal transitions and group them together into high-level data transactions and pass them to Scoreboard.
- *Senders* receive instructions from assigned Input Controllers and data transactions through Transaction Mailbox. The copy of the data transaction is sent to Scoreboard. According to the instruction, Sender builds data and control transactions for specific component in hardware verification environment. In order to deliver the transaction to correct Hardware Driver, Sender provides each transaction with NetCOPE protocol header. Created transactions are buffered in Input Mailbox.
- *Input and Output Wrapper* build communication channel with hardware platform. Input Wrapper uses direct memory access (DMA) for transmission of transactions prepared in Input Mailbox and Output Wrapper uses polling and DMA to get transactions from hardware and buffers them in Output Mailbox.
- *Sorters* take transactions from Output Mailbox and extract NetCOPE protocol header. According to the header, Sorter classifies transactions to data and control. Data transactions are sent in correct format to proper *Output Controller*, which sends them to Scoreboard.
- *Scoreboard* in non-accelerated version compares transactions received from Software Driver and Sofware Monitor, in the case of accelerated version compares transactions received from Sender and Output Controller. If they do not match, Scoreboard reports an error. Scoreboard contains special functions for preprocessing (data transformation before sending to DUT) and postprocessing (data transformation before sending to comparison function in Scoreboard).
- *Driver Switch* routes received transactions to proper Hardware Driver according to NetCOPE protocol header and *Monitor Switch* collects transactions from Hardware Monitors and adds NetCOPE protocol header for correct delivery to sofware units.
- *Hardware Drivers and Monitors* perform similar function as their non-accelerated counterparts.

4 CONCLUSION

The aim of this work is to develop a framework for hardware acceleration of functional verification. We strongly believe that the proposed hardware-software strategy is the best decomposition of the task of functional verification between hardware and software with mapping behavioural parts of the testbench to the software and RTL logic to hardware. The non-accelerated version should work as a good stepping-stone towards the accelerated version by providing a useful debugging environment.

ACKNOWLEDGEMENT

This work was supported by the FIT grant FIT-11-S-1 and the research plan MSM0021630528.

REFERENCES

[1] Veloce: Web pages of Mentor Graphics. URL: `http://www.mentor.com/emulation`

[2] Juergen Jaeger. Perform high-speed, low-cost prototyping of ASIC designs. In EE Times India, Oct 2009. Available at URL: `http://www.embeddeddesignindia.co.in/ART_8800588052_2800004_TA_ccc30d90.HTM`

[3] Andreas Schwarztrauber. SEmulation: Use your emulation board as a hardware accelerator for ModelSim SE. In Verification Horizons, page 31, Dec 2009. Available at URL: `http://www.mentor.com/products/fv/verificationhorizons/upload/horizons-dec-09.pdf`

[4] NetCOPE platform: Web pages of Liberouter. URL: `http://www.liberouter.org/netcope/index.php`