The design of hardware checkers for verification and diagnostic purposes

Martin Straka, Zdeněk Kotásek and Jan Winter
Brno University of Technology
Faculty of Information Technology
612 66, Brno, Czech Republic
{strakam, kotasek, iwinter}@fit.vutbr.cz

Abstract. In the paper, a survey of our research activities the goal of which is to develop a methodology allowing to design on-line checkers of digital components is described. First, our experiments with PSL language and FoCs tool are demonstrated. It is shown how PSL can be used to describe conditions to be checked by an on-line checker of a digital component. It is demonstrated that checkers generated from PSL description demand more sources than the unit under check which is seen as unacceptable result. The principles of our approach based on developing a formal language to describe the functions to be checked and a compiler which transforms the description into VHDL code are explained.

Keywords: on-line checker, on-line testing, verification, PSL, FPGA, FoCs, ModelSim

1 Introduction

Various papers deal with the use of on-line checkers either for verification or on-line testing purposes. For the purposes of design verification, methods exist which enable to synthesize monitors from declarative specifications written in Property Specification Language (PSL) standard. Assertion-Based Verification (ABV) is emerging as a powerful methodology for design verification [1]. In recent years more and more system designers discovered the importance of ABV in coverage driven, functional simulations to keep pace with ever-increasing complexity of modern systems on chip (SoC). Using assertions plays a central role in the design-for-verification (DFV) methodology which is widely used in the industry [2]. Using temporal logic, a precise description of the expected behavior of a design is modeled, and any deviation from this expected behavior is captured by simulation or by formal methods. Hardware verification assertions are written in verification languages such as PSL or SystemVerilog Assertions (SVA). When used in dynamic verification, a simulator monitors the Device Under Verification (DUV) and reports when assertions are violated. Information on where and when assertions fail is an important aid in the debugging process, and is the fundamental reasoning behind the ABV [3]. Such sequences form the core of increasingly-used ABV languages. A checker generator capable of transforming assertions into efficient circuits allows the adoption of ABV in hardware emulation. Method for generating checker circuits from Sequential Extended Regular Expressions (SEREs) with PSL is demonstrated in [4]. The problem of on-line testing is widely discussed in numerous papers, e. g. [5], [6]. In [6], it is presented how path (min) delay faults when designing on-line testable circuits should be taken into account. The challenges that it poses to the existing on-line testing strategies are discussed. Examples showing the possible incorrect behavior of a self-checking circuit as a result of this kind of faults are given. In [7], the idea of combining self-test technology for production test and for on-line self test is presented.

2 Motivation for the Research and Definition of the Problem

In our research we tried to evaluate the possibilities of constructing hardware on-line checkers of components which can possibly occur in digital systems covering various functions. On-line checkers can check simple circuits like counters, coders, comparators, their combinations, etc. The architectures based on checkers can be used in on-line testing methodologies on the RT (Register Transfer) level, verification of design or in FT design. In our research activities we concentrated primarily on assessing the features of PSL language and FoCs tool and their possible use for digital components on-line checkers design of various complexity. Based on the assessment, the need for the development of our own tools should be recognized.
As already mentioned, different tools exist for the description of conditions required to be fulfilled by the design, e.g. PSL and SVA languages. It is a widely referenced fact that the software packages which exist to support them are intended to be used primarily for the design verification purposes.

In our research, we had also a goal to gain all possible information about the following professional tools:

ModelSim SE (Special Edition) is UNIX, Linux, and Windows-based simulation and debug environment, combining high performance with the most powerful and intuitive GUI in the industry. ModelSim provides a comprehensive simulation and debug environment for complex ASIC and FPGA designs. Support is provided for multiple languages including Verilog, SystemVerilog, VHDL and SystemC. For more details see www.modelsim.com

FoCs (short for Formal Checkers, pronounced "fox") Property Checkers Generator is a productivity tool for automatic generation of simulation monitors from formal specifications. It greatly aids chip designers and verification engineers in the complex, costly task of verifying chip designs before submitting them to manufacturing. FoCs Property Checkers Generator is being used by internal IBM users, as well as by external customers. Users report a drastic improvement (up to 50%) in "testbench" development time. FoCs Property Checkers Generator has been made an official component of Blue Logic Methodology.

FoCs Property Checkers Generator takes properties written in the PSL/Sugar specification language and automatically translates them into checkers, or monitors, which in turn are integrated into the chip simulation environment. These checkers monitor the simulation results on a cycle-by-cycle basis for violation of the properties.

Each checker implements a state machine that enters and asserts an error state if the respective property fails to hold in a simulation run. FoCs Property Checkers Generator can also be used for coverage analysis, that is, to create checkers that track the occurrences of events of interest during simulation. FoCs Property Checkers Generator can produce code in Verilog, C++, and VHDL, and it supports the conventions of popular simulators such as Model Technology's ModelSim.

Xilinx ISE (Integrated Software Environment) is a powerful yet flexible integrated design environment that allows you to design Xilinx FPGA and CPLD devices from start to finish. ISE includes our world class design entry, synthesis and implementation tools delivering the industry's fastest place and route times, highest performance, and most advanced design methodologies. For more details see www.xilinx.com

The paper is organized as follows. First of all, the introduction into PSL is presented. The methodology of checker design which is based on the use of PSL is described in section 4. Section 5 shows basic principles of our formal tool which we use to develop on-line checkers and the possibilities of utilizing the methodology for diagnostic purposes. The impact on the number of slices needed to implement the design into FPGA is evaluated for both methodologies. Conclusion and ideas for our future research are summarized in section 7.

3 Property Specification Language

The Property Specification Language, which was adopted by Accellera as IEEE 1850, is an attempt to provide a worldwide standard to endorse assertion based verification [8]. With PSL system designers are able to describe the properties of a system in a tight syntax and clear defined semantics. This enables the implementation of the whole specification in a form that can be verified. Furthermore PSL offers the opportunity to improve the quality of the verification process through functional coverage models which are based on formally specified properties. One of the main requirements of an assertion language is the ability of concise description of design behavior over multiple clocks. PSL supports Sequential Extended Regular Expressions (SEREs) to meet this requirement [2]. SEREs describe single or multi cycle behavior built from a series of Boolean expressions. It provides an easy and familiar way to capture sequential behavior. The syntax is derived from standard UNIX regular expressions. The first and foremost requirement of any temporal sequence is a neat way to describe the advance in time. PSL uses SERE concatenation to achieve this. For a complete review of PSL, which is beyond the scope of this paper, we refer the reader to the language reference manual [8]. An assertion checker is a circuit that captures the behavior of a given assertion, and can be included in the DUV for in-circuit assertion monitoring. A checker generator can be seen as a synthesizer of monitor circuits from assertions, for use in verification. Checkers should be compact, fast and should interfere as little as possible with the DUV, with which they share the resources. Tool for generating hardware checkers from PSL assertions is IBM’s FoCs (Formal Checkers) [9].

4 On-line Checkers Based on PSL

We did a research in the area of possible PSL use either for verification or diagnostic purposes. We decided to verify this idea on RTL components, like coders, decoders, multiplexers, register, etc. We tried to investigate how big the checker generated from PSL description is. During the research we were realizing that the area needed for checker is required to be smaller than the functional element.
To verify the idea, a counter was chosen. For the 4 bit counter, the functions of the checker were described in PSL. The counter has the following inputs: synchronization clock, asynchronous signal “reset” and synchronized signal “start” After the “reset” signal is activated, the outputs of the counter are reset to zero values. The counter starts counting after “start” signal is activated, the values which appear on its outputs are 0 – 15. The counter and its inputs/outputs are demonstrated in Figs. 1 and 2. The counter checker was designed to check the following functions:

- the sequences of counter states (outputs) 0 – 15 (the impact of clock signal),
- the state of the counter after “reset” signal is generated,
- counter activation after “start” signal is activated,
- the effect of “start” signal after which the counting is released,
- the concurrent occurrence of “start” and “reset” signals which is not allowed.

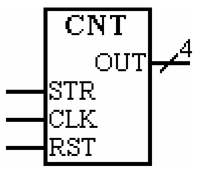


Fig. 1. Counter and its interface

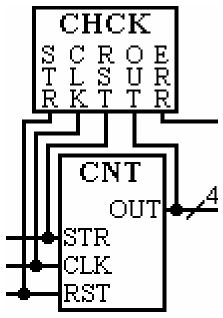


Fig. 2. Counter with checker

The functions of checker were described in PSL language. The description satisfies the requirements defined for entities supposed to be processed by FoCs. The description has the following form:

```
vunit count15{
  cover{[+]; COUNT=0; COUNT=1; COUNT=2; COUNT=3; COUNT=4;
  COUNT=5; COUNT=6; COUNT=7, COUNT=8; COUNT=9; COUNT=10;
  COUNT=11; COUNT=12; COUNT=13; COUNT=14; COUNT=15};
  Assert always{RST}|=>{COUNT=0};
  assert always{START}|=>{COUNT=1};
  assume always(not(RST & START));}
```

Then, the PSL description was converted into VHDL code of checker (FoCs was used for this purpose), the VHDL code was synthesized with Xilinx ISE application. The area needed to cover checker functions is represented by 120 slices. It can be stated that checker area is too big compared with the sources needed to cover counter functions (3 slices). Similar results were gained for other components (decoders, registers, their combinations, etc). We judged that it is so because codes generated by FoCs are supposed to be used primarily for verification purposes not for the implementation into physical design.

To assess the use of PSL for verification purposes, Menthor Graphics ModelSim SE tool was used. The PSL description was slightly modified, the description was then used as an input to FoCs tool. The description has the following form.

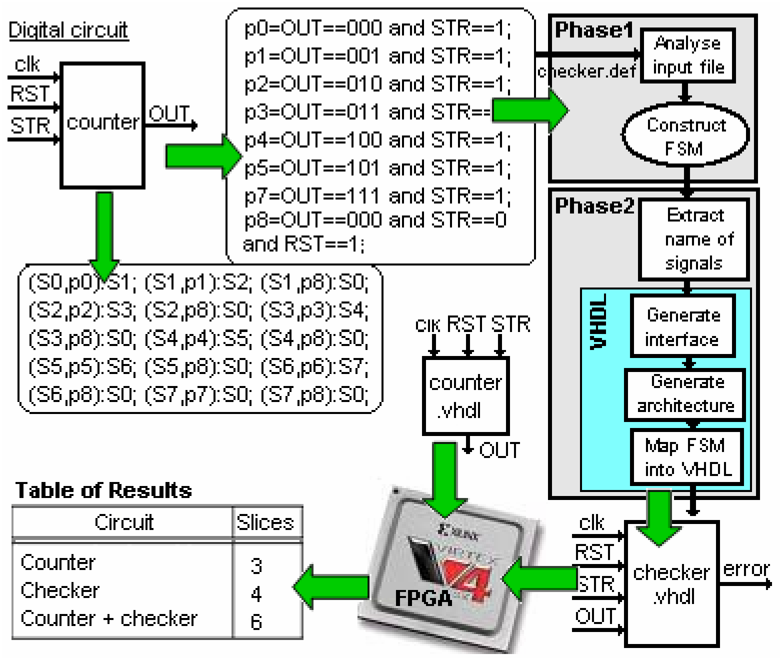
```
library modelsim_lib;
vunit count15(count15(beh_count)){
    default clock is (rising_edge(CLK));
    cover{[+]; COUNT="0000"; COUNT="0001"; COUNT="0010";
      COUNT="0011"; COUNT="0100"; COUNT="0101";
      COUNT="0110"; COUNT="0111"; COUNT="1000";
      COUNT="1001"; COUNT="1010"; COUNT="1011";
      COUNT="1100"; COUNT="1101"; COUNT="1110";
      COUNT="1111"};
    assert always {RST} |=> {COUNT="0000"};
    assert always{START}|=>{COUNT="0001"};
    assume always(not(RST and START));}
```

This description can be then transformed into HDL description which can be further utilized for emulation purposes (e.g. in ModelSim) or to generate resource efficient circuits suitable for hardware emulation (after being synthesized e.g. into FPGA together with the device under verification). Synthesizing resource efficient checker circuits is crucial even for emulation because checker circuits compete with the device under verification for resources.

5 On-line Checkers Design Based on Formal Model

The faults which possibly occur in digital devices can be described in many different ways. Typically, formal model or language is one of the possible ways how to describe fault states. For the description of possible function faults in digital circuits the dedicated language was defined. The main advantage of this approach is such that based on the language the checker can be generated automatically without the intervention of experienced designer.
The language description is composed of two parts. The first one defines the input alphabet symbols that uniquely specify the transitions between automata states. Each input symbol is defined as the set of conditions over the input and output signals. The second part of the language defines the transition function of automata. For each state and input symbol, the transition to the next state is defined. The syntax of definition language and formal description are described in [10]. The principle of our methodology which allows to develop checkers for counter is shown in Figure 3. First

of all, the function of circuit by means of our formal definitions is described, then it is translated into VHDL checker by our core generator. The circuit and his checker are then synthesized into FPGA.



Circuit	Slices
Counter	3
Checker	4
Counter + checker	6

Fig. 3 Demonstration of methodology principles for counter

6 Experimental Results

During this part of our research we aimed at gaining experience with PSL and FoCs tool. We did so because we needed to verify the possibility of utilizing PSL and FoCs as tools which can be used for the description of function to be checked by on-line checkers. Based on our experience and on many references it can be stated that FoCs is intended to be used primarily in the area of design verification and simulation [3] [11]. The results gained from experiments with PSL and FoCs represent for us a justification for the development of our own tool to be used for the design of on-line checkers of digital components with various complexity.

To be able to cover this intension, a language allowing to describe functions of a digital component which are supposed to be checked was created. Together with the formal tool, a compiler was developed which allows to transform the formal description into checker VHDL code. VHDL code can then be synthesized into FPGA. The principles of the formal tool were described in [12].

7 Conclusions and Future Research

Hardware verification aims to ensure that a design fulfills its given specification by either formal or dynamic (simulation based) techniques. Assertion-Based Verification (ABV) is quickly emerging as the dominant methodology for performing hardware verification in practice. Assertions are statements added to the source code that specify how a design should behave. Hardware assertions are typically written in a verification language such as PSL (Property Specification Language) or SVA (SystemVerilog Assertions). In dynamic verification, a simulator can monitor the Device Under Verification (DUV) and report assertion violations. It can be concluded that PSL is supposed to be primarily used in design verification methodologies. In our opinion, PSL cannot be used as a tool for the description of properties to be covered by hardware on-line checker. This is the experience we gained as a result of experimenting with PSL and FoCs tools.

It can be summarized that in our research we have covered the following goals:

- to investigate tools for generating hardware checkers from PSL assertions into VHDL code,
- to verify the possibility of utilizing the checkers developed from PSL descriptions for on-line testing, area overhead being the criterion,
- to evaluate the results and experience gained in previous steps,
- based on previous steps, to develop formal tool for the description of functions to be checked by hardware checker,
- to develop a compiler to transform formal description of properties to be checked into synthesizable VHDL code,
- to compare the effectiveness of our formal tool for generating checkers with checkers based on PSL assertions on the RT level, area overhead being the criterion.

So far, the effectiveness of tool (in terms of the resources needed to cover the functions of the checker) was tested on communication protocol checker and RTL components checkers. The methodology was developed with the goal of lower extent of resources needed to cover the functions of the checker compared with the resource needed to cover the functions of the component under checking. Our experiences will be now utilized in the activities aiming at creating a methodology of FTS design based on the use of on-line checkers possibly combined/compared with TMR based architectures.

Acknowledgments. This work was supported by the Research Project No. MSM 0021630528 - Security-Oriented Research in Information Technology and by GACR project No. 102/05/H050 - Integrated Approach to Education of PhD Students in the Area of Parallel and Distributed Systems (Grant Agency of the Czech Republic).

References

1. M. Boule, J.-S. Chenard, and Z. Zilic. Assertion checker in verification, silicon debug and in-field diagnosis. In ISQED ’07: Proceedings of the 8th International Symposium on Quality Electronic Design, pages 613–620, Washington, DC, USA, 2007. IEEE Computer Society.
2. H. Obereder and M. Pfaff. Behavioral synthesis of property specification language (PSL) assertions. RSP '07: Proceedings of the 18th IEEE/IFIP International Workshop on Rapid System Prototyping, 2007, 0-7695-2834-1, pages 157—160, IEEE Computer Society, Washington, DC, USA.
3. M. Boule and Z. Zilic. Automata-based assertion-checker synthesis of PSL properties. volume 13, pages 1–21, New York, NY, USA, 2008. ACM.
4. K. Morin-Allory and D. Borrione. Proven correct monitors from PSL specifications. In DATE ’06: Proceedings of the conference on Design, automation and test in Europe, pages 1246–1251, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.
5. S.-Y. Yu and E. J. McCluskey. On-line testing and recovery in TMR systems for real-time applications. In ITC ’01: Proceedings of the 2001 IEEE International Test Conference, pp 240, Washington, DC, USA, 2001. IEEE Computer Society.
6. C. Metra, M. Omana, D. Rossi, J. M. Cazeaux, and T. Mak. Path (min) delay faults and their impact on self-checking circuits’ operation. In Proceedings of the 12th IEEE International Symposium on On-Line Testing (IOLTS06), pages 17–22, Corno, Italy, 2006. IEEE Computers Society.
7. C. Galke, M. Grabow, and H. T. Vierhaus. Perspectives of combining on-line and off-line test technology for dependable systems on a chip. volume 00, page 183, Los Alamitos, CA, USA, 2003. IEEE Computer Society.
8. Accellera, “Property Specification Language Reference Manual, ”www.eda.org/vfv/docs/PSL-v1.1.pdf, 2004.
9. IBM, „FoCs - Formal Checkers - a Produktivity Tool, Version 1.0“ http://www.haifa.ibm.com/projects/verification/focs/focs2.pdf, 2008.
10. M. Straka, Z. Kotasek and J. Winter.: Digital Systems Architectures Based on On-line Checker, to appear at EUROMICRO DSD 2008, Parma, Italy.
11. Marc Boule, Zeljko Zilic: Efficient Automata-Based Assertion-Checker Synthesis of SEREs for Hardware Emulation. ASP-DAC 2007, pp. 324-329.
12. Straka Martin, Tobola Jiří, Kotásek Zdeněk: Checker Design for On-line Testing of Xilinx FPGA Communication, In: The 22nd IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, Rome, Italy, IEEE CS, 2007, s. 152-160, ISBN 0-7695-2885-6.