THE DESIGN OF ON-LINE CHECKERS AND THEIR USE IN VERIFICATION AND TESTING

Zdeněk KOTÁSEK*, Martin STRAKA**
*Department of Computer Systems, Faculty of Information Technology
Brno University of Technology, 612 66, Brno, Czech Republic, e-mail: kotasek@fit.vutbr.cz
**Department of Computer Systems, Faculty of Information Technology
Brno University of Technology, 612 66, Brno, Czech Republic, e-mail: strakam@fit.vutbr.cz

ABSTRACT

In the article, a survey of our research activities the goal of which is to develop a methodology allowing to design on-line checkers for digital components and communication protocols are described. First, our experiments with PSL language and FoCs tool are demonstrated for simple RT circuits and communication protocols. 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 on-line checkers generated from PSL description demand more sources than the unit under check which is seen as unacceptable result. The principle of our methodology for generating VHDL descriptions of hardware checkers from the formal model is presented, too. The results and compare of both methodologies are described. The possibilities of utilizing these approaches in the design of Fault Tolerant Systems are described in conclusion.

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

1. INTRODUCTION

On-line checkers in digital system design can be used for several purposes: 1) design verification, 2) on-line testing, 3) fault-tolerant systems design (FTS). 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 checker monitors from declarative specifications written in Property Specification Language (PSL) standard [1]. Assertion-Based Verification (ABV) is emerging as a powerful methodology for design verification. 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) [2]. Using assertions plays a central role in the design-for-verification (DFV) methodology which is widely used in the industry [3]. 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) [4]. 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 [5].

Modern semiconductor technology applications are characterized by an increased demand for high availability and reliability. Self-Checking Circuits (SCC) and on-line checking are a widely used solution due to their ability to detect errors on-line during the normal operation. An SCC consists of a functional circuit (the Circuit Under Monitoring) whose outputs are monitored by a checker. The checker produces an error indication signal whenever the Circuit Under Monitoring produces a incorrect state in the output. In addition, in case of checker's internal faults, it must also provide an error indication and localization. The above requirements are covered by the Totally Self-Checking (TSC) and the Strongly Code-Disjoint (SCD) properties [6].

The problem of on-line testing is widely discussed in numerous papers, e. g. [7], [8]. In [9], 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 [10], the idea of combining self-test technology for production test and for on-line self test is presented.

In [11] the method of highly reliable digital circuit design method based on totally self checking monitors implemented in reconfigurable architecture is described. The bases of the self checking monitors are parity predictors. The parity predictor design method based on multiple parity groups is proposed. Proper parity groups are chosen in order to obtain minimal area overhead and to decrease the number of undetectable faults.

Field Programmable Gate Arrays (FPGA) are increasingly demanded by aircraft and spacecraft electronic designers because of their high flexibility in achieving multiple requirements such as high performance, low cost and fast turnaround time. In particular, SRAM-based FPGAs are very valuable for remote missions and long-time mission because of the possibility of being reprogrammed by user as many times as necessary in a very short period. These properties of FPGA circuits and a concurrent online testing becomes a strong feature in the design of Fault-Tolerant Systems and reliability design [12].

2. 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, and specification of communication protocols. 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.

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:

PSL (Property Specification Language), which was adopted by Accellera as IEEE 1850, is an attempt to provide a worldwide standard to endorse assertion based verification.

FoCs is a productivity tool for automatic generation of simulation monitors from formal specification in PSL.

ModelSim 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.

Xilinx ISE (Integrated Software Environment) is a powerful yet flexible integrated design environment that allows to design Xilinx FPGA and CPLD devices. 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.

3. PROPERTY SPECIFICATION LANGUAGE AND FOCS TOOL

PSL (Property Specification Language), which was adopted by Accellera as IEEE 1850, is an attempt to provide a worldwide standard to endorse assertion based verification [13]. 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. 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 [13]. The principle of verification process is shown in Figure 1. Tool for generating hardware checkers from PSL assertions is IBM's FoCs [14].

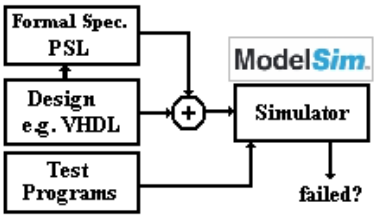

Fig. 1 Demonstration of methodology principles for PSL

FoCs (short for Formal Checkers), Property Checkers Generator is a productivity tool for automatic generation of simulation monitors from formal specifications. 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. Demonstration of methodology principles with FoCs tool is shown in Figure 2.

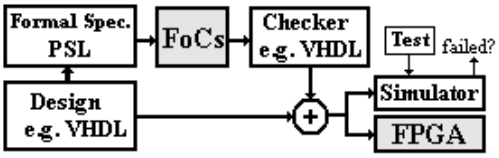

Fig. 2 Demonstration of methodology principles with FoCs tool.

4. ON-LINE CHECKERS DESIGN BASED ON FORMAL MODEL

The faults which possibly occur in digital devices can be described in many different ways. Usually, to describe errors in communication protocols and digital circuits, formal models such as grammars, Finite States Machine (FSM), or formal languages are used. As a result of our research a language was developed which allows to describe possible failures or correct states in communication protocols or simple digital circuits. The description is then used as an input to automatic generator which develops checker description in VHDL language. The main advantage of this approach is such that based on the description the checker can be generated automatically without the intervention of experienced designer. The language description is composed of two phases.

When a protocol or circuit is checked, then not only the combinations of control signals must be monitored but also their sequences and data correctness. The checker behavior must therefore have features of sequential behavior which can be described by means of FSM. The definition of language for digital circuit errors detection therefore arises from the formal description of FSM – (Definition 1):

Definition 1. *A deterministic Finite State Machine is an initialized complete deterministic machine that can be formally defined as a 5-tuple A = (Q, T, P, S0, Serr), where Q is a finite set of states, S0 is the initial state and S0 ∈ Q, T is a finite set of input symbols, P is a next state (or transition) function: P: Q x T → Q and Serr is the finite state Serr ∈ Q. Furthermore Q ∩ T = 0.*

The first part of the language defines the conditions and input symbols of automata. The Definition 2 defines the conditions over the input control signals (the syntax of formal description):

Definition 2. *A condition is formally defined as a X = Sig x Oper x Int, where Sig is the name of control signal, Oper ∈ (<; >; <=; =; ==; <>) is the comparison operator between controlled signal and Int ∈ N is a numeric constant.*

The Definition 3 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 syntax of formal description is here:

Definition 3. *An input automata symbols are defined as conjunction or disjunction of conditions, formally defined as a C: Xi (and X_{i+1})* (or X_{i+1})*, where C ∈ T and i=1,2,..M, where M = ∑(control signals in checking protocol or circuit).*

The second part of the language defines the transition function of automata (Definition 4). For each state and input symbol, the transition to the next state is defined. The syntax of definition language and formal description are here:

Definition 4. *A transition function which is represented by a set of transitions in the form and is formally defined as a P : Q x T → Q.*

As the first step of the input file analysis, the symbols of the files are analyzed together with conditions assigned to them. The set containing all input symbols is created and the syntax analysis of conditional statements is performed. For each conditional statement a syntax tree is formed which is then used during mapping the conditions onto the description in VHDL language. As the result of the analysis, an FSM is constructed, A = (Q,T,P,S0,Serr). The steps of generation process are shown in Figure 3.

The second phase starts with creating the interface of the checker. The names of signals are extracted from transition conditions. The conditions are then mapped onto VHDL processes. The interface signals are the input to the process, the output of the process is the only signal, whose name reflects one of input symbols. The contents of the process is generated from the syntax tree developed in the first phase of the analysis. The mapping of FSM into VHDL is performed by means of two processes. One of them operates as a register in which current state is stored and the second process describes the combinational logic reflecting transition conditions.

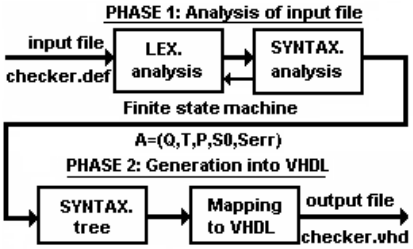

Fig. 3 Phasis of core generator processing.

5. CHECKER FOR SIMPLE CIRCUIT BASED ON PSL AND FORMAL MODEL

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 comparable to that one of 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 "RST" and synchronized signal "STR" After the "RST" signal is activated, the outputs of the counter are reset to zero values. The counter starts counting after "STR" signal is activated, the values which appear on its outputs are 0 – 15. The counter and its inputs/outputs are demonstrated in Figure 4 and counter with checker is demonstrated in Figure 5.

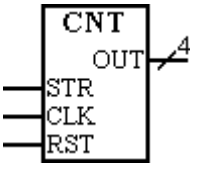

Fig. 4 Counter and its interface.

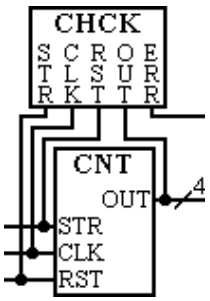

Fig. 5 Counter and its checker.

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 "RST" signal is generated,
- counter activation after "STR" signal is activated,
- the effect of "STR" signal after which the counting is released,
- the concurrent occurrence of "STR" and "RST" signals which is not allowed.

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:

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

This PSL 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.

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 92 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, multiplexers, 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. It covers significantly more functions than needed for on-line checker for diagnostic purposes.

The same functions of checker were described in our formal model too. The description for 3-bits counter has the following form:

```
C0: OUT==000 and RST==0 and STR==0;
C1: OUT==001 and RST==0 and STR==0;
C2: OUT==010 and RST==0 and STR==0;
C3: OUT==011 and RST==0 and STR==0;
C4: OUT==100 and RST==0 and STR==0;
C5: OUT==101 and RST==0 and STR==0;
C6: OUT==110 and RST==0 and STR==0;
C7: OUT==111 and RST==0 and STR==0;
C8: RST==0 and STR==1;
C9: RST==1 and STR==0 and OUT==000;
```

```
(S0,C0):S1; (S1,C1):S2; (S1,C9):S0; (S2,C2):S3; (S2,C9):S0;
(S3,C3):S4; (S3,C9):S0; (S4,C4):S5; (S4,C9):S0; (S5,C5):S6;
(S5,C9):S0; (S6,C6):S7; (S6,C9):S0; (S7,C7):S0; (S7,C9):S0;
A=(Q,T,P,S0,Serr)
```

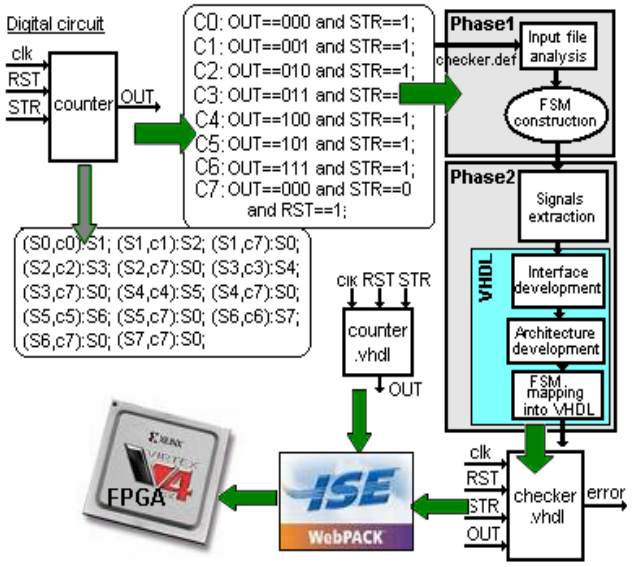

Fig. 6 Demonstration of methodology principles for counter.

The principles of methodology based on formal model which allows to develop FSM checkers for counter is shown in Figure 6. 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 by XILINX ISE tool.

6. CHECKER FOR COMMUNICATION PROTOCOL BASED ON PSL AND FORMAL MODEL

Very often it is reported that FPGA based designs are constructed as fault tolerant designs with the possibility of recovering from errors by means of reconfiguration procedures. In our opinion, testing proper function of communication protocol can increase significantly the diagnostic quality of the design. The idea of technique which allows to develop checkers for communication protocol is shown in Figure 7.

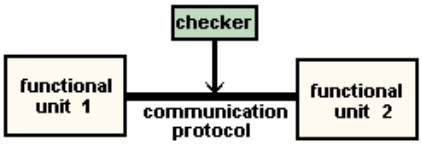

Fig. 7 The idea of checking procedure.

The checker is supposed to operate on different levels of detecting communication protocol faults:

1. The check of protocol control signals and their correct combinations.

2. The check of correct sequences of control signals and evaluation of transitions between communication protocol states.

3. The check of contents of data.

The complexity of the checker will be different based on the type of communication protocol fault supposed to be detected by the checker. As an important aspect of the methodology we saw that the alternative of automated design of the checker should be available to a designer.

The proposed approach for generating checker structure was tested on LocalLink (LL) communication protocol developed by Xilinx company which is used especially for FPGA components interconnection. The LL protocol has been integrated to many IP Cores. The LL is based on synchronous point-to-point communication protocol which transfers data in the form of packets. To the LL advantages generic data width of transferred data belongs which is a very important aspect for stream processing applications. The example of LL communication protocol is shown in Figure 8.

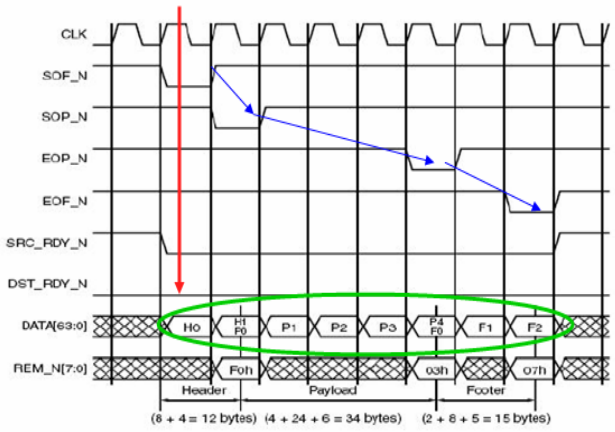

Fig. 8 LocalLink Protocol Timing Diagram.

Additionally, LL offers upstream and downstream flow control, efficient link bandwidth utilization and optional parity checking. The LL interface contains six control signals, data bus and signals identifying the number of valid bytes available in the last data word. Two control signals (SRC_RDY_N and DST_RDY_N) participate in the flow control, allowing both communication sides (source and destination component) can stop the communication. Other four control signals are used for identifying the structure of transferred packet. SOF_N specifies the start of frame, SOP_N identifies the end of the header and the beginning of packet payload, EOP_N determines the end of the payload and the start of the footer. Finally, EOF_N specifies the end of the frame. All control signals are active in L level. Detailed specification of Local Link protocol is available in [15].

Firstly, we defined correct control signals combinations from LL protocol specification. The next part covers data monitoring transported by means of the protocol, checking the condition rules describing the contents of data. An example: the first transported byte must contain 0xAB (Start-of-Frame Delimiter), the ninth byte must have the value which is lower than 124 (the width of the word is 4 bytes). The last type of rules considers the sequences of control signals.

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

```
vunit locallink_timing{
default clock is (rising_edge(CLK));
assume always{ [*] ; SRC_RDY_N & DST_RDY_N & SOF_N &
SOP_N & EOP_N & EOF_N };
assert always{ [*] ; !SRC_RDY_N & !DST_RDY_N & !SOF_N &
SOP_N & EOP_N & EOF_N } |=> { [*] ; !SRC_RDY_N &
!DST_RDY_N & SOF_N & !SOP_N & EOP_N & EOF_N };
assert always { [*] ; !SRC_RDY_N & !DST_RDY_N & SOF_N &
!SOP_N & EOP_N & EOF_N } |=> { [*] ; !SRC_RDY_N &
!DST_RDY_N & SOF_N & SOP_N & !EOP_N & EOF_N };
assert always { [*] ; !SRC_RDY_N & !DST_RDY_N & SOF_N &
SOP_N & !EOP_N & EOF_N } |=> { [*] ; !SRC_RDY_N &
!DST_RDY_N & SOF_N & SOP_N & EOP_N & !EOF_N };
…}
```

The description in formal model for LL has the following form:

```
C0 : SRC_RDY_N==0 and DST_RDY_N==0 and SOF_N==0
     and SOP_N==1 and EOP_N==1 and EOF_N==1;
C1 : SRC_RDY_N==0 and DST_RDY_N==0 and SOF_N==1
     and SOP_N==0 and EOP_N==1 and EOF_N==1
     and DATA_0[7 downto 0]==0xAB;
C2 : SRC_RDY_N==0 and DST_RDY_N==0 and SOF_N==1
     and SOP_N==1 and EOP_N==0 and EOF_N==1
     and DATA_1[7 downto 0]<124;
C3 : SRC_RDY_N==0 and DST_RDY_N==0 and SOF_N==1
     and SOP_N==1 and EOP_N==1 and EOF_N==0;
C4 : SRC_RDY_N==0 and DST_RDY_N==0 and SOF_N==1
     and SOP_N==1 and EOP_N==1 and EOF_N==1;
C5 : SRC_RDY_N==0 or DST_RDY_N==0;
```

```
(S0,C5):S0;  (S0,C0):S1; (S1,C5):S1;  (S1,C1):S2;  (S1,C4):S1;
(S2,C5):S2;  (S2,C2):S3;  (S2,C4):S2; (S3,C5):S3;  (S3,C3):S0;
(S3,C4):S3;  A=(Q,T,P,S0,Serr)
```

In the first approach we consider the protocol as an entity which cannot be partitioned into communication segments. Locallink checker was developed and the requirements on FPGA sources evaluated. The checker of LocalLink is shown in Figure 9. Recently, we have developed a methodology which allows to partition the communication protocol into time segments and develop the checker for each segment separately. This approach allows to assemble selected segments and their checkers together.

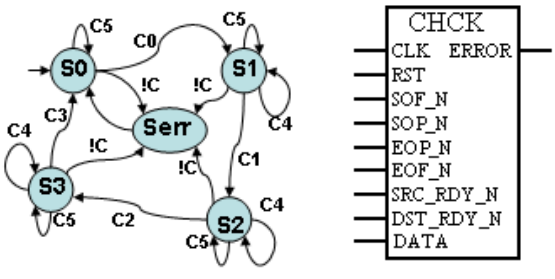

Fig. 9 FSM checker for LocalLink Protocol.

It allows the user to develop checkers which check only the most important segments of the communication and thus can reduce the circuitry needed. The checker which checks combinations of control signals participating on communication protocol is seen in Figure 10.

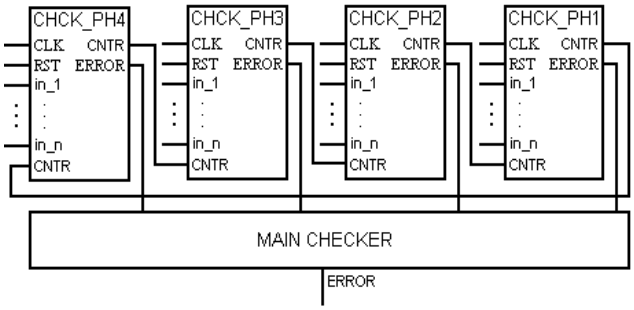

Fig. 10 Segment-checker for LocalLink Protocol.

Each checker checks certain part of the protocol and then the checker structure consists of modules, each of them checking certain part of the protocol. The first module (CHCK_PH1) checks the combinations of control signals during protocol phase1 when header is transmitted. The second checker (CHCK_PH2) checks the protocol during data transmission while the third one is responsible for checking the phase during which footer is transmitted (CHCK_PH3). The last module detects the final phase of the protocol and the idle period of the communication protocol. The error outputs of all modules are evaluated by the main checker which then generates the error signal of the system together with the identification of the module which identified the error. We then compared the results and analyzed possible use of both approaches.

It is important to state that the methodology described in this paper and demonstrated on the design of hardware checkers for LocalLink protocol can be used for other protocols as well.

7. EXPERIMENTAL RESULTS

During this part of our research we aimed at gaining experience with PSL, FoCs tool and formal model. 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 hardware checkers implemented into design. Based on our experience and on other references it can be stated that FoCs is intended to be used primarily in the area of design verification and simulation. 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.

The experiments were performed on XILINX FPGA platform. The components and checkers were synthesized into Virtex5. We compared the number of slices needed to cover the function and checker implementation. Table 1 demonstrates these requirements. The meaning of the columns is as follows: 1st column – the slices needed to cover the function of circuit, 2nd column - the slices needed to cover the function of checker, which is based on formal description (FSM) and 3rd column - the slices needed to cover the function of checker, which is realized as segment-based FSM. It can be seen that a checker requires more sources than the component being checked.

Table 1 The slices needed to cover the function and checker implementation for Virtex5.

Virtex5 – XC5VLX50T	**Circuit**	**FSM Checker**	**Segment Checker**
Circuit	[slices]	[slices]	[slices]
COUNTER 8bits	**4**	**15**	**11**
COUNTER 16bits	**7**	**34**	**23**
COUNTER 32bits	**11**	**79**	**45**
COUNTER input 8bits	**9**	**21**	**13**
COUNTER input 16bits	**25**	**47**	**29**
COUNTER input 32bits	**71**	**87**	**57**
DECODER 4bits	**2**	**5**	**3**
DECODER 8bits	**5**	**7**	**5**
MULTIPLEXOR 4bits	**4**	**8**	**-**
MULTIPLEXOR 8bits	**7**	**10**	**-**

Comparison of results for checker created by FoCs tool and formal methodology are summarized in Table 2. 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.

Table 2 The slices needed for checker based on formal model and FoCs tool.

Virtex5 - XC5VLX50T	FSM checker	FoCs checker
Circuit	[slices]	[slices]
COUNTER4b - only states	5	48
COUNTER4b - full checking	7	92
DECODER4b	5	66
SHIFTER4b	4	52

The last set of experiments was performed for Locallink (LL) communication protocol. We compared the requirements on the number of sources for both types of checker methodology design and different levels of communication protocol checking. We checked the phases of the communication protocol with a checker generated for each phase. The Table 3 demonstrates the number of slices needed for different levels of checking procedure.

Table 3 The slices needed for LL checker based on formal model and FoCs tool.

LocalLink - correct states	FSM checker	FoCs checker
Virtex5 - XC5VLX50T	[slices]	[slices]
LL - only combination (1)	3	29
LL - combin. and sequenc. (2)	7	42
LL - all states with data (3)	16	-

It can be recognized from tables, that the area covered by our on-line checkers is not always smaller than the counter. We do not see this as a negative aspect of the methodology – diagnostics and testing always requires additional hardware and additional costs because it delivers to the design additional features which are important for the design quality.

8. CONCLUSIONS

Hardware verification aims to ensure that a design fulfils 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.

The research we have done in the area of on-line checkers has additional consequences which were described in [16]. We used Markov dependability model to demonstrate that the identification of faulty module increases dependability parameters of the system, like system availability or MTBF/MTTR parameters. To be able to do so, a checker or any other diagnostic tool (e.g. on-line test) can be used. This is the trend we are going to follow in our future research activities. The goal is to develop a methodology which will allow to design a system with required dependability parameters based on various architectures with on-line identification of faulty modules. Different possibilities of faulty modules identification will be taken into account in the methodology. One of the possible approaches based on the use of on-line checkers was described in this paper.

ACKNOWLEDGMENTS

This work was supported by the Research project No. MSM 0021630528 - Security-Oriented Research in Information Technology c GACR Project No. 102/09/1668 – SoC circuits reliability and availability improvement and GACR project No. 102/09/H042 - Mathematical and by the Engineering Approaches to Developing Reliable and Secure Concurrent and Distributed Computer Systems.

REFERENCES

[1] 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*, pp. 1246–1251, Leuven, Belgium, 2006.

[2] 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*, pp. 613–620, Washington, DC, USA, 2007.

[3] M. Boule and Z. Zilic: Efficient Automata-Based Assertion-Checker Synthesis of SEREs for Hardware Emulation. *ASP-DAC 2007*, pp. 324-329.

[4] H. Obereder and M. Pfaff: Behavioral synthesis of property specification language (PSL) assertions. *In RSP '07: Proceedings of the 18th IEEE/IFIP International Workshop on Rapid System Prototyping*, pp. 157—160, IEEE Computer Society, Washington, DC, USA, 2007.

[5] M. Boule and Z. Zilic: Automata-based assertion-checker synthesis of PSL properties. *ACM journal volume 13*, pp. 1–21, New York, USA, 2008.

[6] S. Matakias, Y. Tsiatouhas, T. Haniotakis, A. Arapoyanni and A. Efthymiou: Fast, Parallel Two-Rail Code Checker with Enhanced Testability. *In Proceedings of the 11th IEEE international on-Line Testing Symposium,* IOLTS. IEEE Computer Society, Washington, DC, pp. 149-156, 2005.

[7] M. Straka, Z. Kotasek and J. Winter.: Digital Systems Architectures Based on On-line Checker. *In DSD´08: 11th EUROMICRO Conference on Digital System Design*. Parma, Italy. pp. 81-87, IEEE Computer Society, 2008.

[8] 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. 240-246, Washington, DC, USA,IEEE Computer Society, 2007.

[9] 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 IOLTS06:*

Proceedings of the 12th IEEE International Symposium on On-Line Testing, pp. 17–22, Corno, Italy, IEEE Computers Society, 2006.

[10] C. Galke, M. Grabow, and H. T. Vierhaus: Perspectives of combining on-line and off-line test technology for dependable systems on a chip. *In IOLTS03: International Symposium on On-Line Testing*, pp. 183-189, Los Alamitos, CA, USA, IEEE Computer Society, 2003.

[11] P. Kubalik, P. Fiser, and H. Kubatova: Fault tolerant system design method based on self-checking circuits. *In IOLTS06: Proceedings of the 12th IEEE International Symposium on On-Line Testing*, pp. 185–186, Corno, Italy, IEEE Computer Society, 2006.

[12] M. Straka, J. Tobola and Z. Kotásek: 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, pp. 152-160, IEEE CS, 2007.

[13] Accellera, "Property Specification Language Reference Manual, "www.eda.org/vfv/docs/PSL-v1.1.pdf, 2004.

[14] IBM, „FoCs - Formal Checkers - a Produktivity Tool, Version 1.0. http://www.haifa.ibm.com/ projects/verification/focs/focs2.pdf, 2008.

[15] Xilinx Inc. 2100 Logic Drive. *LocalLink Interface Specification*. San Jose, September 2006.

[16] M. Straka and Z. Kotasek: High Availability Fault Tolerant Architectures Implemented into FPGAs. *to appear at 12th EUROMICRO Conference on Digital System Design*, Patras, Greece. 8 pages, September 2009.

Received May 4, 2009, accepted September 2, 2009

BIOGRAPHIES

Zdeněk Kotásek was born in 1947. He received his MSc. and PhD. degrees (in 1969 and 1991) from Brno University of Technology (BUT), both in computer science. Between 1969 and 2001, he worked at Department of Computer Science of the the Faculty of Electrical Engineering and Computer Science (FEE), since 2002 at the Department of Computer Systems (DCSY) of the Faculty of Information Technology (FIT), both at BUT. He is an Associate Professor at BUT since 2000 and the head of the DCSY (since 2005). His research interests include digital circuit diagnostics and testing, testability analysis and design and synthesis for testability and reliability. He is an IEEE member (since 2003).

Martin Straka was born in 1981. In 2006 he graduated (MSc) at the department of Computers Systems of the Faculty of Information Technology, Brno University of Technology. In 2006 he started his PhD studies at the Department of Computers Systems. His scientific research is focused on fault tolerant systems design and on-line testing of FPGA based systems.