Digital Systems Architectures Based on On-line Checkers

Martin Straka, Zdenek Kotasek, Jan Winter
Brno University of Technology
Faculty of Information Technology
Bozetechova 2, Brno, 612 66, Czech Republic
{strakam, kotasek, iwinter}@fit.vutbr.cz

Abstract

In this paper, a methodology for generating VHDL descriptions of hardware checkers is presented. It is shown how the methodology can be used to generate on-line checkers of communication protocols, counters, decoders, registers, comparators, etc. It is also demonstrated how a checker for more complex structures can be developed. We describe the possibilities of utilizing this approach in the design of Fault Tolerant Systems (FTS). Experimental results in terms of FPGA resources needed to synthesize different types of checkers are presented.

1. Introduction

On-line checkers in digital system design can be used for several purposes: a) design verification, b) on-line testing, c) fault-tolerant system design. For the purposes of design verification, methods exist which enable to synthesize monitors from declarative specifications written in PSL standard.

Different Fault Tolerant (FT) architectures are known to improve reliability in real-time systems, Triple Modular Redundancy and duplex systems can serve as examples. Real-time systems are often used in hazardous or remote applications, such as aircraft and spacecraft where the systems are highly susceptible to errors due to radiation [11]. In these applications, the length of mission plays an important role. Usual approach to provide fault tolerance is through redundancy, such as N-modular redundancy or duplex systems [5]. With the design of FTS, the concept of on-line testing is combined [8]. It can be implemented on different levels of digital systems, e. g. NoC [1]. The concepts combined with on-line/off-line testing are discussed in [13].

Assertion-Based Verification (ABV) is emerging as a powerful methodology for design verification [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 (Property Specification Language) or SVA (SystemVerilog Assertions). 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. Such sequences form the core of increasingly-used Assertion-Based Verification (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 [3].

One approach how to construct FT systems is through the use of checkers. From among languages which can be used to describe functions checked by checkers, PSL (Property Specification Language) and SVA (System Verilog Assertions) can be mentioned [10], [3]. While PSL is based on Sugar language from IBM, SVA combines features of Synopsys OVA, Motorola CBV, and Accelera PSL. The problem of on-line testing is widely discussed in numerous papers [16]. In [9], it is presented how path (min) delay faults when designing on-line testable circuits should be taken into account. The challenges that this poses to the existing on-line testing strategies are discussed. Examples showing the possible incorrect behaviour of a self-checking circuit as a result of this kind of faults are given. In [6], the idea of combining self-test technology for production test and for on-line self test is presented.

Protocols have grown larger and more complex with the advent of computer and communication technologies [12]. As a result, the task of conformance testing of protocol implementation has also become more complex. The study of DFT (Design For Testability) is a research area in which researchers investigate design principles that will help to overcome the ever increasing complexity of testing distributed systems. Testability metrics are essential for evaluating and

comparing designs. In [4], a new metric for testability of communication protocols is introduced, based on the detection probability of a default. The authors presents two approaches for improved testing of a protocol implementation once those faults that are difficult to detect are identified.

In [7], the author proposes a novel method for PSL language assertions simulation-based checking. The method uses a system representation model called High-Level Decision Diagrams (HLDD). Previous works have shown that HLDDs are an efficient model for simulation and convenient for diagnosis and debug. The presented approach proposes a temporal extension for the existing HLDD model aimed at supporting temporal properties expressed in PSL. Other contributions of the paper are methodology for direct conversion of PSL properties to HLDD and HLDD-based simulator modification for assertions checking support.

2. Motivation for the Research and Definition of the Problem

In our research we tried to evaluate the possibilities of constructing checkers of different functions which can possibly occur in a digital system. The architectures based on checkers can be possibly used in on-line testing methodologies or in FTS design. To be able to do so, it was necessary:

- to develop formal tools needed to describe functions to be checked by the checker,
- to develop a compiler to transform formal description of properties to be checked into synthesizable VHDL code,
- to synthesize the VHDL code into some platform,
- to evaluate the results, i.e. the sources needed to implement the checker into FPGA (number of CLBs/slices).
- to evaluate the possibilities of developing FT systems based on the usage of checkers in the design (i.e. the comparison of architectures based on checkers with other FT architectures).

As mentioned above, other tools exist for the description of conditions required to be fulfilled by the design, e.g. PSL and SVA languages. The software packages which exist to support them are intended to be used primarily for the design verification purposes.

The paper is organized as follows. First of all, our methodology which we use to describe conditions to be fulfilled by the design is described in section 3. The section 4 deals with the demonstration of the compiler which we use to convert formal description to VHDL code. The principles of communication protocol checker design are presented in section 5 - two approaches are demonstrated and discussed. Then, the basic ideas of utilizing the methodology for the design of digital components like counters, decoders, multiplexers and combinations of them are described (section 6). In section 7 - experimental results - the impact on the number of slices needed to implement the design into FPGA is evaluated for each design. This information is important for the comparison of architectures based on checkers with other architectures like TMR. This comparison is important for the evaluation of possible applications of on-line checkers. As a result, it should be clearly stated whether TMR architectures can be replaced by other architectures containing on-line checkers which do not require so many sources.

3. Formal Description of Design Conditions

As stated above, we developed our tool to define conditions to be satisfied. The conditions are then compiled into checker VHDL code which can be then integrated into the resulting design together with the functional unit which will be checked by the checker. We have done so even when it is known that tools based on the use of PSL exist. They generate checker VHDL code which is primarily supposed to be used for hardware simulation of conditions to be used for design verification purposes, not for the synthesis into some platform and the use as a checker. Therefore, we do not see PSL and its software support as a proper alternative to fulfill our objectives.

To describe communication protocol specification or sequential component behavior, the definition language uses FSM construction. This approach combines the description of sequential component behavior and the description of communication protocol or sequential component states. A communication protocol description must contain also the description of sequences of signals. Thus, the language for communication protocol specification consists of two types of tools: for the description of either combinational or sequential logic. Our formal approach is based on the following definitions:

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 $S0 \in Q$, T is a finite set of input symbols, P is a next state (or transition) function: $P : Q \times T \rightarrow Q$ and $Serr$ is the finite state and $Serr \in Q$. Furthermore $Q \cap T = \emptyset$.

A condition is formally defined as a $C(i) = Sig \times Oper \times Int$, where Sig is a name of control signal, $Oper \in (<, >, <=, =, ==, <>)$ is a comparison operator between controlled signal and $Int \in N$ numeric constant. $i \in 1, 2, 3, ...$

An input automata symbols are defined as conjunction of conditions, formally defined as a $p(n) = \bigwedge_{i=1}^{X} C(i)$,

where $n \in 1, 2, 3, ..., N$ *and* $p(n) \in T$ *and* $X = \sum(control\ signals\ in\ checking\ protocol)$

A transition function is represented by a set of transitions: $P(n) : Q \times T \rightarrow Q$*, where* $n \in 1, 2, 3, ..., N$ *and* Q *is a finite set of states*

4. The Compiler from Checker Formal Description to VHDL Code

As mentioned above, we are developing the compiler from formal language into VHDL specification. The VHDL code is synthesized and the checker design is then evaluated in terms of the sources needed to implement the design into FPGA. So far, we concentrated on certain components and their checkers, and possible combinations of functional components and their checkers. We do not concentrate on constructions which can possibly appear in the formal description but on components which can appear in the design. We do it in this way because we primarily need to verify the volume of checker circuitry generated by our compiler from checker formal description. Thus, we worked with such components as counters, registers, decoders, multiplexers, etc. and the impact of their design on resulting checker size. It is important for our research because the implementation of checker represents an additional area overhead compared to other FT architectures which do not use checkers, but are based on other topologies, like TMR based architectures.

5. Communication Protocol Checker Design

We used two approaches in this part of our research, both of them were verified on the design of Locallink protocol checker [15], [14]. The LocalLink protocol is used especially for FPGA components interconnection and it has been integrated to many IP Cores.

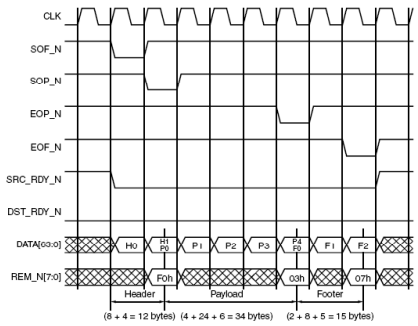

Figure 1. LocalLink protocol specification

The LocalLink is based on synchronous point-to-point communication protocol which transfers data in the form of packets. The specification of LocalLink communication protocol is shows in Figure 1. The LocalLink formal description of conditions to be checked has the following form ($p0 - pn$ are input symbols of automata composed of conditions over control signals combinations, transition function (Sn, pn) describes correct sequences of states):

$$
\begin{aligned}
p0 &= SRC\_RDY\_N == 0\ and\ DST\_RDY\_N == 0 \\
&\quad and\ SOF\_N == 0\ and\ SOP\_N == 1\ and \\
&\quad EOP\_N == 1\ and\ EOF\_N == 1; \\
p1 &= SRC\_RDY\_N == 0\ and\ DST\_RDY\_N == 0 \\
&\quad and\ SOF\_N == 1\ and\ SOP\_N == 0\ and \\
&\quad EOP\_N == 1\ and\ EOF\_N == 1; \\
p2 &= SRC\_RDY\_N == 0\ and\ DST\_RDY\_N == 0 \\
&\quad and\ SOF\_N == 1\ and\ SOP\_N == 1\ and \\
&\quad EOP\_N == 0\ and\ EOF\_N == 1; \\
p3 &= SRC\_RDY\_N == 0\ and\ DST\_RDY\_N == 0 \\
&\quad and\ SOF\_N == 1\ and\ SOP\_N == 1\ and \\
&\quad EOP\_N == 1\ and\ EOF\_N == 0; \\
p4 &= SRC\_RDY\_N == 0\ and\ DST\_RDY\_N == 0 \\
&\quad and\ SOF\_N == 1\ and\ SOP\_N == 1\ and \\
&\quad EOP\_N == 1\ and\ EOF\_N == 1; \\
p5 &= SRC\_RDY\_N == 0\ or\ DST\_RDY\_N == 0;
\end{aligned}
$$

$$
\begin{aligned}
&(S0, p5) : S0; (S0, p0) : S1; \\
&(S1, p5) : S1; (S1, p1) : S2; (S1, p4) : S1; \\
&(S2, p5) : S2; (S2, p2) : S3; (S2, p4) : S2; \\
&(S3, p5) : S3; (S3, p3) : S0; (S3, p4) : S3;
\end{aligned}
$$

In the first approach we consider the protocol as an entity which cannot be partitioned into communication slices. Locallink checker was developed and the requirements on FPGA sources evaluated. The checker of LocalLink is shown in Figure 2.

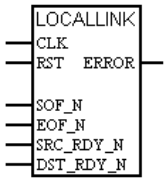

Figure 2. Checker for LocalLink protocol

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. 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 on Figure 3. 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 signal during protocol phase1 when header is transmitted. The second checker (CCH_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.

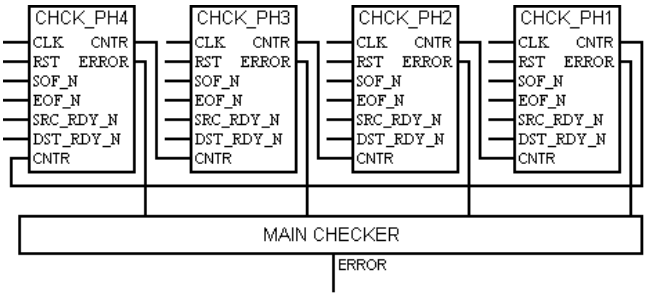

Figure 3. Module based checker of LocalLink protocol

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.

6. On-line Checkers for Simple Circuits

Our methodology is based on constructing checkers for basic digital circuits and their combinations. For this purpose a specialized language was developed which allows to describe properties to be checked. Different levels of properties can be described.

6.1 Counter Decoder Checker Design

A counter and its decoder are two components which are frequently used in digital systems design (see Figure 4). The counter counts clock signals (CLK), when RST signal is generated, the status is changes to zero state. It is a sequential component, to describe the sequence of states, our formal language was used. The checker will then have a character of FSM. A simple form of the checking can be provided through checking the sequence of states and the zero state after RST is generated. A more sophisticated checking procedure will be based on checking all possible combinations of signals in all states. If the counter contains Start (STR) and Stop signals, then proper values on both of these inputs must be checked. Both alternatives are compared later in this text in terms of slices needed to implement the checker into FPGA.

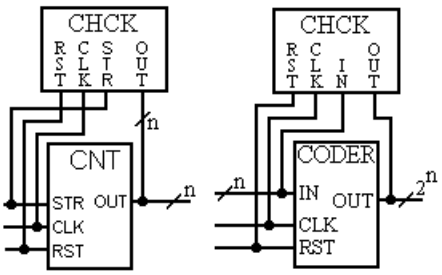

Figure 4. Counter and decoder checkers

On the contrary, a decoder is a purely combinational component, its checker must certify whether the outputs of the counter are correct related to decoder inputs. The requirements on FPGA sources are also presented in experimental results. The counter formal description of conditions to be checked has the following form:

$$
\begin{aligned}
p0 \quad & = OUT == 000 \; and \; STR == 1; \\
p1 \quad & = OUT == 001 \; and \; STR == 1; \\
p2 \quad & = OUT == 010 \; and \; STR == 1; \\
p3 \quad & = OUT == 011 \; and \; STR == 1; \\
p4 \quad & = OUT == 100 \; and \; STR == 1; \\
p5 \quad & = OUT == 101 \; and \; STR == 1; \\
p7 \quad & = OUT == 111 \; and \; STR == 1; \\
p8 \quad & = OUT == 000 \; and \; STR == 0 \\
& \quad and \; RST == 1;
\end{aligned}
$$

$$
\begin{aligned}
&(S0, p0) : S1; (S1, p1) : S2; (S1, p8) : S0; \\
&(S2, p2) : S3; (S2, p8) : S0; (S3, p3) : S4; \\
&(S3, p8) : S0; (S4, p4) : S5; (S4, p8) : S0; \\
&(S5, p5) : S6; (S5, p8) : S0; (S6, p6) : S7; \\
&(S6, p8) : S0; (S7, p7) : S0; (S7, p8) : S0;
\end{aligned}
$$

6.2 Counter and its Decoder Checker Design

In Figure 5, a structure with counter and its decoder is shown. The checker checks both components. As an advantage we see the fact that two components are checked with one checker. The formal description of conditions to be checked has the following form:

$$
\begin{aligned}
p0 \quad & = OUT == 00000001 \; and \; TMP == 000; \\
p1 \quad & = OUT == 00000010 \; and \; TMP == 001; \\
p2 \quad & = OUT == 00000100 \; and \; TMP == 010; \\
p3 \quad & = OUT == 00000100 \; and \; TMP == 011; \\
p4 \quad & = OUT == 00001000 \; and \; TMP == 100; \\
p5 \quad & = OUT == 00010000 \; and \; TMP == 101; \\
p7 \quad & = OUT == 01000000 \; and \; TMP == 110; \\
p8 \quad & = OUT == 10000000 \; and \; TMP == 111; \\
p9 \quad & = OUT == 10000000 \; and \; TMP == 000 \\
& \quad and \; RST == 1;
\end{aligned}
$$

$(S0, p0) : S1; (S1, p1) : S2; (S1, p9) : S0;$
$(S2, p2) : S3; (S2, p9) : S0; (S3, p3) : S4;$
$(S3, p9) : S0; (S4, p4) : S5; (S4, p9) : S0;$
$(S5, p5) : S6; (S5, p9) : S0; (S6, p6) : S7;$
$(S6, p9) : S0; (S7, p7) : S8; (S7, p9) : S0;$
$(S8, p8) : S0; (S8, p9) : S0;$

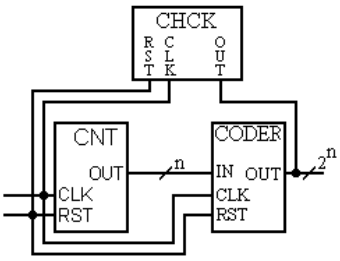

Figure 5. Counter and its decoder checker

6.3 Serialiser - Register with Multiplexer

In Figure 6, a circuit which converts parallel data into serial form is shown. It consists of a register and multiplexer. Data is loaded into register by CLK signal, EN_WR must be active. The checker checks the correctness of data on multiplexer output.

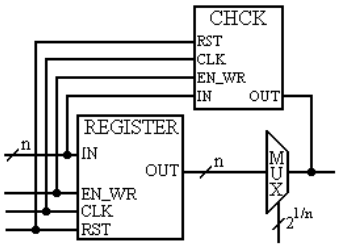

Figure 6. Serialiser checker

6.4 TMR and Duplex with Checkers

Fault tolerant system is often implemented as TMR. TMR system is based on duplicating functional units and evaluates the results by means of majority element (see Figure 7). Another alternative of FT system is in duplex system with one or two checkers. The outputs of duplex system are evaluated by means of comparator or multiplexor. The comparison of both techniques for basic digital components described in previous sections is available in section 7. Even when it happens that the checker requires more resources to cover all its functions, it can be still acceptable solution because typical duplex systems need to implement comparator into the design to indicate that an error occurred while for TMR based architectures the resulting design contains a voter which can be possibly more resources demanding.

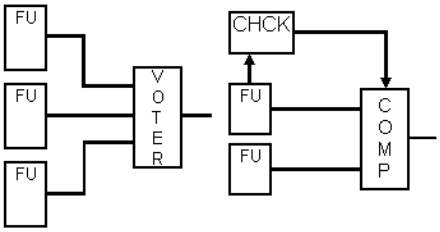

Figure 7. TMR and Duplex with checker

7. Experimental Results

The experiments were performed XILINX FPGA platform. The components were synthesized into Virtex2Pro and Virtex5. We compared the number of slices needed to cover the function and checker implementation. Table 1 demonstrates these requirements. It can be seen that a checker requires more sources than the component being checked.

Virtex2Pro - XC2VP2 -	**Circuit** [slices]	**Checker** [slices]
Counter - complete checking	2	14
Counter - states checking	2	12
Decoder	4	5
Counter+decoder	5	13
Serialiser	3	4
Shift register	4	6
Voter	7	-
Comparator	5	-

Table 1. Resources usage in slices for Virtex2Pro

Interesting results were gained for the implementation into FPGA Virtex5 (see Table 2). It is evident from the table that a new technology and the option to use LUTs with 6 inputs allows to implement N-variable functions with a lower number of slices. For some components the sources needed for checker implementation is not significantly higher than the sources needed for the functional unit.

We performed additional experiments to compare FT techniques, namely TMR with duplex techniques based on the use of checkers. For the comparison, both techniques were implemented into FPGA Virtex2Pro and Virtex5. The following structures were implemented: TMR system, duplex system with one checker and duplex system with two checkers. All combinations of components in Table 3 were implemented as TMR with voter. As a metric, the sources needed for the implementation in FPGA Virtex2Pro and Virtex5 of TMR system, duplex with one checker and duplex with two checkers were used. All types of digital com-

Virtex5 - XCV50E -	Circuit [slices]	Checker [slices]
Counter - complete checking	2	7
Counter - states checking	2	4
Decoder	4	4
Counter+decoder	5	6
Serialiser	3	3
Shift register	3	4
Voter	7	-
Comparator	5	-

Table 2. Resources usage in slices for Virtex5

ponents in Table 3 were implemented as TMR with voter.

Virtex2Pro - XC2VP2 -	TMR [slices]	DUPL.+1CH [slices]	DUPL.+2CH [slices]
Counter - complete checking	12	22	35
Counter - states checking	12	18	30
Decoder	20	17	22
Counter+decoder	26	25	26
Serialiser	12	13	15
Shift register	16	18	20

Table 3. Virtex2Pro resources usage in slices for TMR and Duplex

It can be derived from the table that good results can be gained in duplex architectures with one checker. In some configurations, it requires less sources than TMR based architectures. On the other hand, duplex architectures with two checkers provide a good solution to cover checked functions, the sources needed are much higher than for TMR. A similar situation holds for FPGA Virtex5, the results are summarized in Table 4.

Virtex5 - XCV50E -	TMR [slices]	DUPL.+1CH [slices]	DUPL.+2CH [slices]
Counter - complete checking	9	14	17
Counter - states checking	9	11	15
Decoder	20	18	20
Counter+decoder	20	22	26
Serialiser	12	13	14
Shift register	13	15	18

Table 4. Virtex5 resources usage in slices for TMR and Duplex

The last set of experiments was performed for Locallink communication protocol. We compared the requirements on the number of sources for both FPGA types and different levels of communication protocol checking. We checked the phases of the communication protocol with a checker generated for each phase. The Table 5 demonstrates the number of slices needed for different levels of checking procedure.

LocalLink - checker -	Virtex5 [slices]	Virtex2Pro [slices]
Single - only combinations	3	4
Complex - only combinations	3	4
Single - all states	5	8
Complex - all states	7	9

Table 5. Resources usage in slices for LocalLink

As single checker (Table 5) we understand a separate checker which checks either combination or states. A complex checker consists of several checkers, each of them checking particular LocalLink protocol segment (see Figure 5).

8. Conclusions and Future Research

In this paper, a technique for automated design of checkers for different types of digital components and its combinations is presented. A methodology for automated design of communication protocol is demonstrated in the paper as well.

For the purposes of our research, we developed a formal language which allows to describe properties of digital components and communication protocols to be checked. The software which allows to compile the properties to be checked into VHDL description was then developed. We also experimented with FoCs tool to verify the possibility of using PSL language for checker design. Unfortunately, the designs gained from FoCs are resulting in too many slices needed to implement the design into FPGA. In our opinion, it is so because VHDL codes gained from FoCs are primarily supposed to be used for verification purposes.

The goal of our research was to evaluate the developed checkers in terms of the area overhead and its comparison with the area needed to cover the function being checked. Various combinations of digital components and its checkers were developed as either TMR or duplex architectures. The results can be summarized in the following way:

- The complexity of the configuration based on duplex architecture with one checker results in the same complexity as TMR.
- Duplex architecture with two checkers requires more sources than TMR and thus appears to be inconvenient for the use in the design of FT systems.

It can be concluded that the methodology described in this paper can be used for the design of architectures where the use of on-line checkers is required. The methodology presented in this paper starts with formal description of the properties to be checked and results in VHDL checker code and its synthesis with a professional design tool.

In the future research we intend to deal with applications in which long lifetime is required, and the resources in FPGA can be possibly exhausted. To solve this problem, the faulty configuration has to be reconfigured into a new fault tolerant design with smaller area. The methodology will allow to develop a sequence of architectures, each architecture covering the required function and equipped with certain level of diagnostic circuitry. For each level, dependability parameters will be evaluated. The lifetime of these architectures will depend strongly on the area available in FPGA and sources needed to cover the function.

Acknowledgements

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] P. S. Bhojwani and R. N. Mahapatra. A robust protocol for concurrent on-line test (colt) of noc-based systems-on-a-chip. In *DAC '07: Proceedings of the 44th annual conference on Design automation*, pages 670–675, New York, NY, USA, 2007. ACM.

[2] M. Boule, J.-S. Chenard, and Z. Zilic. Assertion checkers 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.

[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] A. Chung and T. Huang. Two approaches for the improvement in testability of communication protocols. In *ICIS '05: Proceedings of the Fourth Annual ACIS International Conference on Computer and Information Science*, pages 562–565, Washington, DC, USA, 2005. IEEE Computer Society.

[5] S. D'Angelo, G. R. Sechi, and C. Metra. Transient and permanent fault diagnosis for fpga-based tmr systems. In *DFT '99: Proceedings of the 14th International Symposium on Defect and Fault-Tolerance in VLSI Systems*, pages 330–338, Washington, DC, USA, 1999. IEEE Computer Society.

[6] 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.

[7] M. Jenihhin, J. Raik, A. Chepurov, and R. Ubar. Temporally extended high-level decision diagrams for psl assertions simulation. In *ETS '08: Proceedings of the 13th IEEE European Test Symposium 2008*, pages 61–68, Los Alamitos, USA, 2008. IEEE Computer Society.

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

[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 *Proceedings of the 12th IEEE International Symposium on On-Line Testing (IOLTS06)*, pages 17–22, Corno, Italy, 2006. IEEE Computer Society.

[10] 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.

[11] R. Oliveira, A. Jagirdar, and T. J. Chakraborty. A tmr scheme for seu mitigation in scan flip-flops. In *ISQED '07: Proceedings of the 8th International Symposium on Quality Electronic Design*, pages 905–910, Washington, DC, USA, 2007. IEEE Computer Society.

[12] A. Petrenko, R. Dssouli, and H. König. On evaluation of testability of protocol structures. In *Proceedings of the IFIP TC6/WG6.1 Sixth International Workshop on Protocol Test systems VI*, pages 111–124, Amsterdam, The Netherlands, The Netherlands, 1994. North-Holland Publishing Co.

[13] J. Savir. On-line and off-line test of airborne digital systems: a reliability study. In *ITC '00: Proceedings of the 2000 IEEE International Test Conference*, page 35, Washington, DC, USA, 2000. IEEE Computer Society.

[14] M. Straka, J. Tobola, and Z. Kotasek. Checker design for on-line testing of xilinx fpga communication protocols. In *DFT '07: Proceedings of the 22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, pages 152–160, Washington, DC, USA, 2007. IEEE Computer Society.

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

[16] 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*, page 240, Washington, DC, USA, 2001. IEEE Computer Society.