Online Protocol Testing for FPGA Based Fault Tolerant Systems

Jiri Tobola, Zdenek Kotasek, Jan Korenek, Tomas Martinek, Martin Straka
Brno University of Technology
Faculty of Information Technology
Bozetechova 2, Brno, 612 66, Czech Republic
{xtobol01, kotasek, korenek, martinto, strakam}@fit.vutbr.cz

Abstract

In this paper, the methodology for automated design of checker for communication protocol testing is presented. Based on the level of checking, different design strategies can be performed - in the paper the lowest level is presented. The definition of dedicated language for the description of possible communication faults is presented. The core generator is used to produce VHDL code describing the behaviour of the checker.

1. Instructions

Fault-tolerance is an important system metric for many operating environments, from automotive to space exploration. The conventional technique for improving system reliability is through component replication, which usually comes at significant cost: increased design time, testing, power consumption, volume, and weight. Reconfigurable systems implemented using user-programmable logic elements such as FPGA are well suited for applications where high dependability is required. The problems combined with the design of dependable systems include error detection during system operation, fast fault location, quick recovery from temporary failures, and fast permanent-fault repair [4].

In [2] the method of highly reliable digital circuit design method based on totally self checking blocks implemented in FPGAs is described, parity predictors are used for this purpose. 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.

In [5], the authors present how an original method to synthesize monitors from declarative specifications written in the PSL (Property Specification Language) standard was developed. Monitors observe sequences of values on their input signals, and check their conformance to a specified temporal expression. The method implements both the weak and strong versions of PSL FL operators, and has been proven correct using the PVS theorem prover. The paper discusses the salient aspects of the proof of their prototype implementation for on-line design verification.

The problem of on-line testing is widely discussed in numerous papers. In [3], 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 [1], the idea of combining self-test technology for production test and for on-line self test is presented.

2 Motivation for the research

The complexity of the checker will be different based on the type of communication protocol fault supposed to be detected by the checker. The complexity of the checker will influence the area required on the chip and communication speed. As an important aspect of the methodology we saw that the alternative of automated design of the checker should be available to a designer. For this purpose, we felt the need for a language by means of which the checker will be described together with the need for core generator to compile checker description into VHDL code. Based on the detection of incorrect sequences in the communication protocol it can be judged that a fault occurred in the hardware and the hardware can be reconfigured to guarantee the correct operation. To provide this, certain features which are typical for fault tolerant systems were added to the architecture. Based on the experience gained during the research, the results were summarized and plans for the future accepted to enhance fault tolerant aspects of the design.

The final objective of our research can be characterised in the following way: 1) the development of fault tolerant hardware for Virtex-II Pro board which is now being used in different applications (FlowMon, Traffic scanner, etc.),

2) the development of both on-line and off-line strategies to detect hardware faults, 3) the development of self-repairing reconfiguration strategies to recover from hardware faults. We intend to develop three levels of the checker, the lowest one is presented in this paper.

3 Language Definition

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 communication protocol faults 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 communication protocol 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 initial state is labelled as *Start* and the error during the communication protocol is detected by the transition to *Error* state. If the automata is not completely defined, for uncovered input signal combinations it remains in the same state.

The input automata symbols are defined as conjunction of conditions $cond_1$, $cond_2$, ..., $cond_N$ separated by $\wedge$ symbol. Each condition contains single comparison operators ($<$, $>$, $<=$, $>=$, $==$ and $<>$) between signal and constant, and each condition can be negated using operator "!". Expression defined in this way is assigned to new input symbol. Syntactic structure is shown in the following example:

$$p_0 = cond_0 \wedge cond_1 \wedge cond_2 \wedge ... \wedge cond_N$$

The automata behaviour is described using transition function which is represented by a set of transitions in the form $(S_1, p0) \rightarrow S_2$. Based on the set of input characters and transition function, it is possible to formally construct the finite state machine and thus to design the checker. Example of the simple checker behavior is shown here:

$$p_0 = a_0 = 1 \wedge a_1 > 3 \wedge ...$$
$$p_1 = a_0 = 1 \wedge a_1 > 3 \wedge ...$$

$$(S_0, p_0) \rightarrow S_1$$
$$(S_1, p_1) \rightarrow S_3$$
$$(S_3, p_0) \rightarrow S_0$$
$$(S_3, p_1) \rightarrow S_{error}$$
$$S_{error} \rightarrow S_0$$

$$A = (Q, T, P, Start, Error),$$

Q is the set of states, T is the set of input symbols, P is used to define the following state for each state and input symbol. $Start$ is the starting state and $Error$ is the error state of the checker.

4 Core Generator

Core generator is a program which was created for automatic design of checker from the definition language. The generation process consists of two phases shown in Figure 1. In the first phase, input file is analysed and finite state machine (FSM) is constructed. During the second phase, FSM is mapped into VHDL description. The output of the core generator is a VHDL description of checker.

Input file analysis is started by reading input symbols and conditions. During the analysis, set of input symbols with conditions is constructed. Concurrently, the condition is parsed and a syntactic tree is created and stored for the usage in the mapping phase.

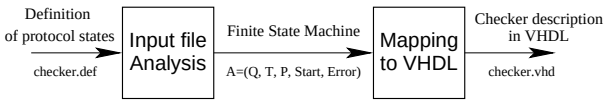

Figure 1. Two phasis of core generator processing

When the set of input symbols T is constructed, the analysis process continues by reading FSM transitions. Source and destination states are then extracted from the transitions and both states are added to the set of states Q. After that it is checked if the input symbol which is used to trigger the transition, is in the set T. If the input symbol is not included in T, an error is reported and the core generator is closed. At the end of input file analysis an automaton $A = (Q, T, P, Start, Error)$ is constructed.

The mapping phase starts by developing checker interface. First, used signals are extracted from every symbol condition. All extracted signals are used as the input to FSM transitions and have to be part of the checker interface. The error signal is the only output of checker and has to be included into the component interface. All FSM inputs and error output are signals which are used for the VHDL entity generation.

The generation of VHDL architecture is started by mapping conditions to VHDL processes. Inputs to the process are signals used in the condition and an output is signal with the same name as the input symbol associated to the condition. Process contents is generated from the syntax tree constructed during the analysis phase. FSM is mapped in two VHDL processes. The first process is a current state register and the second process describes the next state logic. In

both cases, the description is similar to the language templates for synthesis. Synthesis tools can recognize FSM from the VHDL description and find appropriate implementation for target technology.

5 Evaluation of the Methodology

The proposed approach for generating checker structure was tested on Local Link communication protocol [6] developed by *Xilinx* company which is used especially for FPGA components interconnection. The Local Link protocol has been integrated to many IP Cores. The Local Link

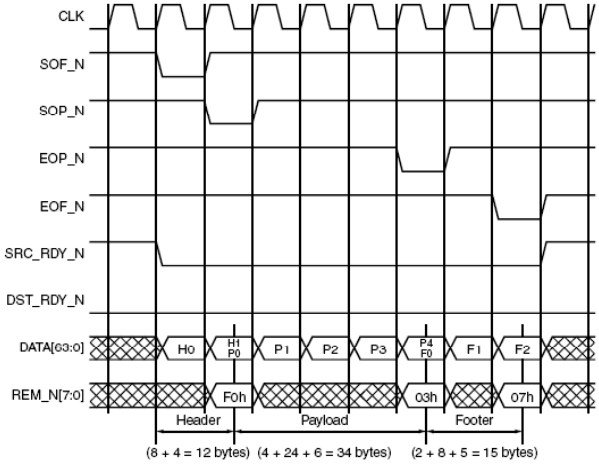

Figure 2. Local Link Protocol Timing Diagram

(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. Six control signals are used for identifying the structure of transferred packet. The example of Local Link communication protocol is shows in Figure 2. Detailed specification of Local Link protocol is available in [6].

For the purposes of checker evaluation, three different levels of diagnosis were chosen: (1) The lowest level, where checker checks the protocol control signals and detects states that do not meet the protocol specification. (2) The second level verifies the correct sequence of control signals and evaluates the transitions between communication protocol states. (3) At the third level, the content of data is verified whether it satisfies specified conditions and rules.

In the first case, the checker only checks the protocol control signals without the information about the communication protocol status. From the Local Link timing diagram (see Figure 2), it is obvious, that some of the control signals cannot be active simultaneously. Forbidden combinations can be summarized in the following rules:

1. Active SOF and EOF are forbidden because header and footer have to be present.
2. Active SOF and EOP are forbidden because footer cannot start before header.
3. Active SOP and EOP are forbidden because the information about header and payload reminder cannot be available at the same time.
4. Active SOP and EOF are forbidden because frame cannot end without footer.
5. Active EOP and EOF are forbidden because the information about payload and footer reminder cannot be available at the same time.

This list of the rules can be easily rewritten into the language (defined in section 3) as follows:

$$
\begin{aligned}
p_0 &= SRC\_RDY\_N = 0 \wedge DST\_RDY\_N = 0 \wedge \\
&\quad SOF\_N = 0 \wedge EOF\_N = 0 \\
p_1 &= SRC\_RDY\_N = 0 \wedge DST\_RDY\_N = 0 \wedge \\
&\quad SOF\_N = 0 \wedge EOP\_N = 0 \\
p_2 &= SRC\_RDY\_N = 0 \wedge DST\_RDY\_N = 0 \wedge \\
&\quad SOP\_N = 0 \wedge EOP\_N = 0 \\
p_3 &= SRC\_RDY\_N = 0 \wedge DST\_RDY\_N = 0 \wedge \\
&\quad SOP\_N = 0 \wedge EOF\_N = 0 \\
p_4 &= SRC\_RDY\_N = 0 \wedge DST\_RDY\_N = 0 \wedge \\
&\quad EOP\_N = 0 \wedge EOF\_N = 0
\end{aligned}
$$

$$
\begin{aligned}
&(S_0, p_0) \rightarrow S_{error}, (S_0, p_1) \rightarrow S_{error} \\
&(S_0, p_2) \rightarrow S_{error}, (S_0, p_3) \rightarrow S_{error} \\
&(S_0, p_4) \rightarrow S_{error}
\end{aligned}
$$

$$Q = (Q, T, delta, q_0, F)$$

The second group of rules considers the sequences of control signals. For the Local Link protocol the following rules can be applied:

- Any of four control signals SOF_N, SOP_N, EOP_N, EOF_N can not be activated more then once without activation of any other control signal. The condition is valid only if signal SRC_RDY_N and DST_RDY_N are active.
- If the frame does not contain either header or footer then a EOF_N activation has to follow SOF_N activation. The condition is valid only if signal SRC_RDY_N and DST_RDY_N are active.

- If the frame contains header and footer then the control signals are activated in the following order: SOF_N, SOP_N, EOP_N, SOP_N, EOP_N, SOP_N, EOP_N, and EOF_N. The condition is valid only if signal SRC_RDY_N and DST_RDY_N are active.

$$
\begin{aligned}
p0 &= SRC\_RDY\_N = 0 \wedge DST\_RDY\_N = 0 \wedge \\
&\quad SOF\_N = 0 \\
p1 &= SRC\_RDY\_N = 0 \wedge DST\_RDY\_N = 0 \wedge \\
&\quad SOP\_N = 0 \\
p2 &= SRC\_RDY\_N = 0 \wedge DST\_RDY\_N = 0 \wedge \\
&\quad EOP\_N = 0 \\
p3 &= SRC\_RDY\_N = 0 \wedge DST\_RDY\_N = 0 \wedge \\
&\quad EOF\_N = 0
\end{aligned}
$$

$$
\begin{aligned}
&(S\_0, p\_0) \rightarrow S\_SOF,\ (S\_0, p\_1) \rightarrow S\_SOP, \\
&(S\_0, p\_2) \rightarrow S\_EOP,\ (S\_0, p\_3) \rightarrow S\_EOF, \\
&(S\_SOF, p\_1) \rightarrow S\_SOP \ldots \\
&\ldots (S\_SOF, p\_0) \rightarrow S\_error, \\
&(S\_SOP, p\_1) \rightarrow S\_error, \\
&(S\_EOP, p\_2) \rightarrow S\_error, \\
&(S\_EOF, p\_3) \rightarrow S\_error,
\end{aligned}
$$

The third type of rules check not only control signals but also control data which can be part of frame header or footer. Rules can for example check the first byte of Ethernet frame or any other byte in frame header which do not not have arbitrary value. For the LocalLink protocol, eight bytes of data were tested to evaluate the number of resources needed for checker unit implementation.

Rules	Slices
Signal Combination (1)	4
Sequence of signals (2)	7
Data checking (3)	16

Table 1. Resources usage in Slices

The synthesis to Virtex-II Pro FPGA was also performed to obtain basic parameters of generated circuit. For all generated circuits the maximal frequency was higher than 300 MHz and does not affect maximal frequency of IP cores. FPGA logic utilization was different for all types of rules. The results are summarized in table 1.

6 Conclusions and Future research

This paper proposes a technique for automatic design of checker component for communication protocol testing. Using a simple language it is possible to model the fault states at different levels of communication protocol. The complexity of the fault model or its input rules directly impacts the amount of resources needed for component implementation. The results achieved on Local Link protocol show that relatively small number of resources is needed for the detection of possible faults. The presented approach can be further utilized for example in systems with two module redundancy or as a trigger for activating a partial reconfiguration of corrupted components.

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] C. Galke, M. Grabow, and H. T. Vierhaus. Perspectives of combining on-line and off-line test technology for dependable systems on a chip. *IOLTS*, 00:183, 2003.

[2] 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.

[3] 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.

[4] S. Mitra, W.-J. Huang, N. R. Saxena, S.-Y. Yu, and E. J. McCluskey. Reconfigurable architecture for autonomous self-repair. *IEEE Design and Test of Computers*, 21(3):228–240, 2004.

[5] 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.

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