Design of Self Checking Circuits Based on FPGA

Pavel Kubalík, Hana Kubátová
Department of Computer Science and Engineering
Czech Technical University
Karlovo nam. 13, 121 35 Prague 2
e-mail: xkubalik@fel.cvut.cz, kubatova@fel.cvut.cz

Abstract

The paper focuses on error detection in circuits implemented in FPGAs using error detection codes (ED codes). The incorrect function of a given combinational circuit has to be detected and signalized at the time of its appearance and before its further distribution. It means that a safe operation is guaranteed. The ability to detect an error without stopping circuit function is called concurrent error detection (CED). We have used combinational circuits only to simplify testing process. A previous research was based on benchmarks described by tables. In some cases benchmarks with many inputs cannot be described by tables easily. The benchmarks used in our experiments to compute a quality of the code are described by equations. All of them will be implemented in XILINX FPGA circuits. Therefore the fault model considers the way of configuration data storage in memory. This work is a part of a more complex methodology of fault tolerant design based on FPGAs with a possibility to reconfigure the faulty part of the circuit.

1 Introduction

Nowadays when the circuit integration increases, the importance of radiation impact on integrated circuits grows even at the sea level. The mobile nets are becoming more important because of their radiation. They can affect any circuit used every day. Some machines such as the control units in cars can play an important role in places such as tunnels, where a car fault can endanger human lives. Other important areas are aviation, medicine or space missions. All of these applications and many others depend on a correct function of circuits and one wrong result can mean huge losses. The FPGA circuits are more and more often used to realize any function because of their prices and capabilities to upgrade the function when a bug is discovered. Another advantage is the possibility of dynamic reconfiguration when fault in the circuit is detected and localized [4].

Some techniques in [5] describe methods how to detect the faulty part of FPGA without stopping its function. The method tests unused part of the FPGA. When the test is done the tested part is exchanged with the used part and the test is started again for currently unused area.

This paper is organized as follows. Section 2 introduces the fault model and proposes general methodology for comparing quality of used error detection codes. Section 3 presents the experimental issue and solution of parity bits generation for every used test. Future work and conclusion are presented in the section 4.

2 Solution

With consideration of using FPGA circuits we have to take into account the property and representation of combinational part realized as memory (look up table - LUT) with several inputs and one output. After the synthesis the mapping is started and a set of gates is mapped on individual LUTs and connected by inner nets. The FPGA is based on the memory that contains configuration data realizing required function.

When the radiation impacts the chip the function of circuit may change due to toggling some bits in the configuration memory. The incorrect function is subsequently detected by an additional combinational circuit. It consists of gates generating parity bits (parity generators) and gates checking code words (checkers).

The parity generator is based on the duplicated combinational circuit. The part that generates parity bits is added according to the used detection code. Some of used codes have error-correction capabilities but we cannot correct outputs because a fault is propagated to outputs through a complex circuit. The error-correcting codes are often used for data paths. In the case of using error-correcting codes in the combinational circuits a single fault can influence multiple outputs and correction leads to a wrong data output.

The aim of this work is not to optimize the checking part but to find out appropriate safety codes. The synthesis process can only change the size of final circuit in this application, because the parity generator is not optimized in our work. To obtain only quality parameters of tested codes such optimization of the parity generator is not necessary.

The stuck-at fault model is used. The FPGA function is determined by the content of the configuration memory.

The fault is considered as a change of any bit in configuration memory. All combinational parts in FPGA are realized as many LUTs [4] connected to each other. Every LUT represents a part of the configuration memory. The connection net is realized by wires, multiplexers and switches and their settings depend on bits in configuration memory.

The proposed fault model is sufficient to compare quality of tested codes and to select the suitable one. The quality w is the ratio of the detected incorrect outputs x in the combinational circuit and all incorrect outputs y, $w = x/y$.

The creation of combinational circuits solving parity computation using CED techniques is based on the original circuit. The original circuit is duplicated and the parity generator is added. The synthesis process must be applied on each part separately due to the fact that the parity generator contains the same logic as the original circuit. Otherwise, the synthesis would optimize the circuit and remove redundancy.

Error-detecting circuits based on the duplication of original circuit can be predisposed to the genesis of Common-Mode Failures (CMF). Some solutions of this problem are presented in [1].

The methodology to design the combinational part for the parity generation does not depend on the original circuit and on the number of its inputs. Our experiments assume circuits with many inputs. Due to this fact the exhaustive test cannot be applied and so the minimal test must be generated. The test is inserted to the fault simulator and for every fault the output value is generated. After performing the simulation process, the checking part is computed by our software and compared with outputs generated by simulator.

3 Experiments

The Atalanta software was used to generate minimal test [6]. This tool allows to process ISCAS benchmarks based on equations. A tool for circuits optimization described by tables cannot be used in this case because the table creation is difficult due to many inputs.

The special software has been written in C language for automatic modification of the original circuit and adding the parity bits generator. The ISCAS benchmarks are loaded into inner form by this software. The form is composed of list of nets and gates. Some functions are written to modify this inner form and can add new gates, nets or can change name of all nets. The nets renaming allows to duplicate an original circuit. After the original circuit is loaded into the memory all nets are renamed and the original circuit is loaded again.

To add parity bits the original circuit is loaded and modified by corresponding function. Next all nets are renamed and original circuit is loaded again. By this procedure the original circuit with predicted parity bits has been obtained.

For every tested checking code the new functions modifying a circuit have been written. The even parity, double parity and multiple parity generated by a Hamming code are used in experiments described in this article. When the original circuit is modified, two methods of output form can be generated. The first form presents ISCAS benchmark used for simulation. The second form is VHDL source code allowing to synthesize modified circuit.

The variety of circuits can be created by this tool. Every part of the final circuit is created separately and combined together with the same tool. If the area occupied by circuit in FPGA has to be computed, then the VHDL output is selected and a synthesize tool is used. After that the original circuit and modified circuit can be compared. In our case the VHDL code is used to obtain the area overhead by every checking code. Two VHDL codes are generated: first for original circuit and second for circuit generating parity bits. For simulation the design contains both original circuit and parity bits generator (Figure 1).

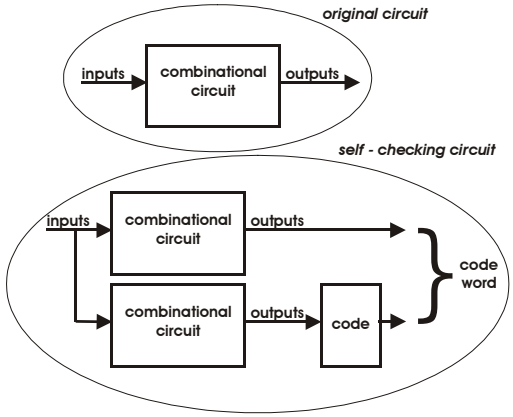

Figure 1. The circuit duplication and code word generation

The Atalanta software processes the modified circuit and generates the minimal fault test. Both files, minimal test and modified circuit, are put into the simulator which allows to compute efficiency of used checking code. The same tool that allows to modify original circuit is used for simulation. The same form of circuit stored in memory is used. The new functions of simulation are added to the source code of our tool (Figure 2).

By this modified tool we can simulate faults. Firstly, our tool simulates stuck-at-zero faults by consecutive fault insertion in every net. When fault is inserted the whole set of patterns from minimal test is applied to every net. The same steps are used for simulation of stuck-at-one. The simulator does not simulate faults on primary inputs and outputs. For every step of the test, the outputs of circuits

with and without inserted faults are processed. In a case when the outputs are incorrect the check of code word is performed.

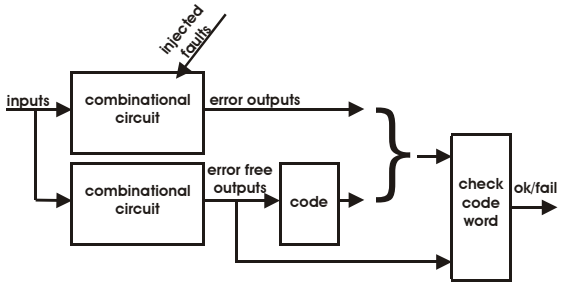

Figure 2. Automatic fault insertion and checking code word

We have chosen such codes, e.g. Hamming codes, even parity and double parity, for which the checking combinational part can be generated easily. Berger code is not used because it is difficult to generate checking combinational part.

3.1 Even parity

The even parity, that is the simplest checking code, was used for the first experiment. Results, presented in Table 1, show that one parity bit cannot cover all faults inserted into the tested circuit. The circuit c17 is not used for our experiments because of its simplicity.

3.2 Double parity

The double parity was used to generate checking bits as a second experiment. Even and odd bits of outputs are coded separately by even parity. The results of this experiment are presented in Table 2.

Both previous experiments did not reach 100% fault coverage of tested circuits.

Next possibility how to generate checking bits is using the Hamming code, which enables adding more checking bits with keeping quality of the Hamming code.

The Hamming code is defined by its generating matrix. For simplicity we use the matrix containing the unity submatrix at the right side. The generating matrix of Hamming code (15,11) is shown in Figure 3. The values a_{ij} have to be defined.

$$G = \left(\begin{array}{cccc|cccc} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & 1 & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{11,1} & a_{11,2} & a_{11,3} & a_{11,4} & 0 & 0 & \cdots & 1 \end{array}\right)$$

Figure 3. Generating matrix for Hamming code (15,11)

When a more complex Hamming code is used, more values have to be defined. The number of outputs o_i used for checking bits determines the appropriate code. For example the circuit c432 that has 7 outputs requires at least Hamming code (15,11). In this case we use 7 data bits and 4 checking bits. The definition of values a_{ik} is also important.

Now we present a method how to generate values a_{ik}. Let us mention the Hamming code (15,11) that has 4 checking bits. We generate a set of all 4bit vectors. From all these vectors we remove vectors containing less then 2 binary ones. The resulting subset is relatively regular - there are many zeros at the upper left side and many ones at the lower left side of the subset (see the left matrix in Figure 4). This regularity must be removed. If not, some parity bits would lose a capability to detect a fault. To eliminate this phenomenon every even row from the beginning of the set is mutually exchanged with a corresponding even row from the end (see Figure 4).

$$\begin{pmatrix} 0&0&1&1 \\ 0&1&0&1 \\ 0&1&1&0 \\ 0&1&1&1 \\ 1&0&0&1 \\ 1&0&1&0 \\ 1&0&1&1 \\ 1&1&0&0 \\ 1&1&0&1 \\ 1&1&1&0 \\ 1&1&1&1 \end{pmatrix} \Rightarrow \begin{pmatrix} 0&0&1&1 \\ 1&1&1&0 \\ 0&1&1&0 \\ 1&1&0&0 \\ 1&0&0&1 \\ 1&0&1&0 \\ 1&0&1&1 \\ 0&1&1&1 \\ 1&1&0&1 \\ 0&1&0&1 \\ 1&1&1&1 \end{pmatrix}$$

Figure 4. Generating left side of matrix

The number of vectors in the set is the same as the number of rows in the appropriate Hamming matrix. Then we generate circuits for checking bits x_k (Equation 1).

$$x_k = a_{1k}o_1 \oplus a_{2k}\, o_2 \oplus \ldots \oplus a_{mk}o_m, \qquad (1)$$

where $o_1 \ldots o_m$ are outputs of the circuit.

3.3 Hamming code (63,57)

The third experiment is based on the Hamming code (63,57), where the maximum number of data bits is 57 and the number of checking bits is 6. Experimental results are shown in Table 3.

The fault coverage for c499 and c1355 benchmarks is 100%. It means that the Hamming code (63,57) is appropriate. We must mention that fault coverage depends on generated minimal test. If the minimal test created by Atalanta does not cover all faults, we cannot say that simulated circuits are 100% fault covered. In other words some faults cannot be detected because minimal test set does not cover all faults. This Hamming code cannot be used for benchmark c2670 because the number of its outputs is bigger than the Hamming code can cover.

3.4 Hamming code (255,247)

The fourth experiment is based on the Hamming code (255,247). The maximum number of data bits is 247 and the number of checking bits is 8. In our case only 7 outputs are used. The experimental results are shown in Table 4.

4 Conclusions

This work is the part of methodology of automatic design of concurrent error detection CED circuits based on FPGA with possibility of the reconfiguration part that generates incorrect outputs. The reliability is increased by reconfiguration. The most important part is a speed of the fault detection and the safety of all circuit towards the surrounding environment. We can summarize, all of our experiments say that 100% fault coverage can be reached using more redundancy outputs generated by special codes. The Hamming code can be used as a suitable code to generate parity bits. Its type depends on the number of outputs and on the complexity of the original circuit. More complex circuits need more parity outputs.

Acknowledgement

This research has been in part supported by the GA102/03/0672 grant and MSM 212300014, 1999 – 2003.

References

[1] Mitra, S. and E. J. McCluskey, “Which Concurrent Error Detection Scheme To Choose?” Proc. International Test Conf., pp. 985-994, 2000.

[2] C. Bolchini, F. Salice,D. Sciuto, “ Designing Self-Checking FPGAs through Error Detection Codes” 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'02), November 06 - 08, 2002 , Canada,

[3] K. Elshafey, J. Hlavička, “ On-Line Detection and Location of Faulty CLBs in FPGA- Based Systems”, IEEE DDECS Workshop, Brno, Czech republic, April 17-19, 2002, pp. 183-190.

[4] XAPP 151 (v1.5), Virtex Series Configuration Architecture User Guide

[5] M. Abramovici, C. Stroud, C. Hamiliton, S. Wijesuriya, V. Verma, “Using Roving STARs for On-Line Testing and Diagnosis of FPGAs in Fault-Tolerant Applications. Proceeding IEEE International Test Conference, pp. 973-982, 1999

[6] H.K. Lee and D.S. Ha, "Atalanta: an Efficient ATPG for Combinational Circuits,", Technical Report, 93-12, Dep't of Electrical Eng., Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1993

Circuit	Inputs	Outputs	Redundancy parity bits	All tested faults	Detected faults	Detected faults[%]	Area occupation[LUT]	
							Redundancy part	Data part
c432	36	7	1	5536	4626	83,56	72	69
c499	41	32	1	15150	14628	96,55	87	88
c880	60	26	1	24567	23998	97,68	117	112
c1355	41	32	1	64165	62472	97,36	92	87
c1908	33	25	1	134012	119280	89,01	125	120
c2670	233	140	1	105532	84840	80,39	166	175

Table 1. Application of even parity code

Circuit	Inputs	Outputs	Redundancy parity bits	All tested faults	Detected faults	Detected faults[%]	Area occupation[LUT]	
							Redundancy part	Data part
c432	36	7	2	5433	5012	92,25	73	69
c499	41	32	2	13984	13750	98,33	99	88
c880	60	26	2	27495	27206	98,95	120	112
c1355	41	32	2	62834	62012	98,69	90	87
c1908	33	25	2	130140	124958	96,02	124	120
c2670	233	140	2	116220	111270	95,74	219	175

Table 2. Application of double even parity code

Circuit	Inputs	Outputs	Redundancy parity bits	All tested faults	Detected faults	Detected faults[%]	Area occupation[LUT]	
							Redundancy part	Data part
c432	36	7	6	5569	5544	99,55	77	69
c499	41	32	6	17791	17791	100,00	116	88
c880	60	26	6	27109	27106	99,99	140	112
c1355	41	32	6	68647	68647	100,00	117	87
c1908	33	25	6	123651	123376	99,78	145	120

Table 3. Application of Hamming code (63,57)

Circuit	Inputs	Outputs	Redundancy parity bits	All tested faults	Detected faults	Detected faults[%]	Area occupation[LUT]	
							Redundancy part	Data part
c432	36	7	7	5694	5602	98,38	74	69
c499	41	32	7	18003	18003	100,00	111	88
c880	60	26	7	30277	30277	100,00	134	112
c1355	41	32	7	69634	69634	100,00	104	87
c1908	33	25	7	135402	134600	99,41	138	120
c2670	233	140	7	160092	160061	99,98	314	175

Table 4. Application of Hamming code (255,247)