An Efficient Multiple-Parity Generator Design for On-Line Testing on FPGA

Abstract

We propose a method to efficiently design a "parity generator", which is a stand-alone block producing multiple parity bits of a given circuit. The parity generator is designed by duplicating the original circuit, by XOR-ing given groups of its outputs and resynthesizing the whole circuit. The resulting circuitry is smaller than the original circuit in most of cases. The major task to be solved is to properly select the groups of outputs to be XORed to obtain multiple parity bits and maximally reduce the generator size. A method based on principles of the FC-Min minimizer is proposed in this paper. The parity generator can be exploited in on-line diagnostics, to design self-checking circuits. In our solution the self-checking circuits are basic blocks of the modified duplex system architecture used for increasing dependability parameters of a reliable design based on FPGAs. The method is tested on standard MCNC benchmark circuits and its efficiency is evaluated.

Track area: Fault Tolerance in Digital System Design

Topic: on-line BIST

Keywords: BIST, parity generator, on-line testing, Boolean minimization

1. Introduction

Systems realized by Field Programmable Gate Arrays (FPGAs) are more and more popular and widely used in more and more applications due to several advantages, like their high flexibility in achieving multiple requirements such as cost, performance and turnaround time and the possibility of reconfiguration and later changes of the implemented circuit, e.g., only via wireless connections.

The FPGA circuits can be used in mission critical applications such as aviation, medicine, space missions, and railway applications as well [1, 2, 3].

Many FPGAs are based on SRAM memories sensitive to Single Even Upsets (SEUs), therefore a simple usage of FPGA circuits in mission critical applications without using any method of error detection (and possibly correction) is impossible.

A change of one bit in the configuration memory leads to a change of a circuit function, often drastically. The Concurrent Error Detection (CED) techniques allow a faster detection of soft errors (errors which can be corrected by reconfiguration) caused by SEUs [4, 5, 6]. SEUs can change also the content of the embedded memory, Look-up Tables (LUTs) and other configuration bits. These changes are not detectable by off-line tests, therefore CED techniques have to be used. The probability of a SEU occurrence in the SRAM is described in [7].

The self-checking (SC) circuit (a method based on a CED technique) is used to detect an occurrence of a fault in the tested circuit. Only one copy of the SC circuit is not sufficient to increase dependability parameters. Thus, we use the Modified Duplex System (MDS) architecture [8].

This paper presents a parity generator design method based on parity bits grouping. The parity groups are generated from the original circuit's outputs. The self checking circuit quality is determined by an area overhead and the number of undetectable faults while keeping dependability parameters. The "dependability" is currently used to express the ability of a system or of its component to correctly perform its function, or "mission" over time [9].

Previously we have proposed an output grouping method based on evaluating a "similarity" of the functions [10]. Now we propose a method based on the principles of the FC-Min minimizer [11, 12, 13]. Here we exploit principles of sharing group implicants among two or more outputs of the function. The groups of outputs to be XORed are derived from the numbers of group implicants they share.

The paper is structured as follows: the principles of the parity generator and the dependable architecture based on a modified duplex system are described in Section 2. The dependability analysis is presented in Section 3. Then the FC-Min algorithm used to generate groups of parity bits is described in Section 4, the principles of the grouping of the outputs are stated in Section 5. Section 6 contains the experimental results and Section 7 concludes the paper.

2. The Parity Generator

The self-checking circuit is constructed by duplicating the original circuit and XORing the

outputs of the duplicate circuit, to obtain parity bits. The code words obtained by the original circuit and the parity generator are then compared in the Checker, see Fig. 1.

The number of used parity bits (check bits) significantly influences the area overhead, together with the dependability parameters [14]. Thus, proper number of parity bits has to be chosen, so that the overall logic would be minimized and the dependability of the circuit maximized.

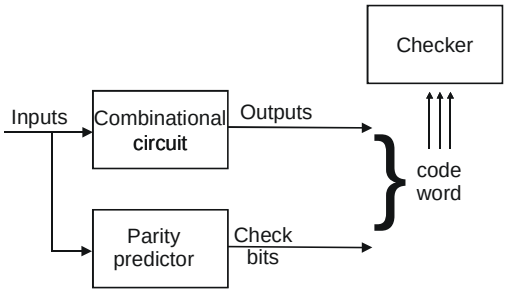

Figure 1: The self-checking circuit design

2.1. MDS architecture

When self-testing and self-checking parameters are satisfied to 100%, also the totally self checking (TSC) parameter is satisfied to 100% [9]. Our previously obtained results show that to fulfill the TSC property to reach 100% is difficult [15], so we are using a modified duplex system (MDS) architecture [9] based on two FPGAs, see Fig. 2.

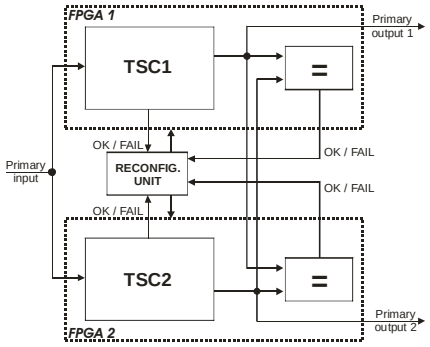

Figure 2. The MDS architecture

Each FPGA has its primary inputs, primary outputs and two pairs of checking signals OK/FAIL. The probability of the information correctness depends on the Fault Security (FS) property. When the FS property is satisfied only to 75%, the correctness of the checking information is also 75%. It means that the signal "OK" give a correct information for 75% of occurred errors (the same probabilities for both signals "OK" and "FAIL").

3. Dependability Analysis

To evaluate the influence of a sequence of the SEUs faults, a more precise definition of "a single fault" is needed. Availability computations for dependability analysis are used. In the following text we will assume that a "single data damaging" is defined as follows:

- It will occur at a single time event that is arbitrarily located at the time axis.
- The fault can change a data item located within the FPGA configuration memory. Both FPGAs can be affected with the same probability. We assume that a single fault changes only one bit of the FPGA configuration memory. Each bit in the FPGA configuration memory can be attacked with the same probability.
- The time between any two single faults is sufficient enough to enable a single fault to be successfully detected and corrected. If not, a multiple fault occurs.

Some basic rules are defined to calculate the availability parameters. We assume that:

- There is at least one input vector occurring between two SEUs which cause an output to differ from the normal operation.
- SEUs occurring in an unused logic do not change the function of the used part, therefore these faults are hidden.
- The comparator and the checker fully satisfy TSC property.
- The area overhead of the comparator and the checker is negligible.
- The reconfiguration unit loads correct configuration data after the fault being detected.
- The time needed to reconfigure the faulty part depends on the configuration data size.
- The fault occurred in the unused logic does not cause the damage of the whole FPGA.

The Markov model shown in Fig. 3 describes our architecture.

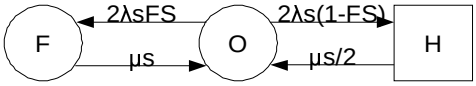

Figure 3. Model of our modified duplex system

There are three states (**O**, **F**, **H**). The **O** state (operational) represents the regular fault-free state of the system, where both FPGAs operate correctly. It

means that the malfunction function is signalized neither by the TSC circuit, nor by the comparator.

There is a transition from **O** to **F** state (one FPGA is faulty) corresponding to the situation when a fault occurs in one FPGA and this fault is detected by one of the TSC circuits. The system enters this state with a probability FS. λ is the failure rate for one bit of a configuration memory and s is the size of a configuration memory. The number 2 in the 2λsFS expression means that one of two FPGAs can be affected by SEUs. The reconfiguration process is initiated only for the faulty FPGA. The repair rate is represented by μ. The second FPGA is running correctly, and therefore performs the function of the system.

Some faults are not detected, when the output vector is an incorrect codeword. The probability that the occurred fault causes an incorrect codeword is equal to **1-*FS***. In this case, the system comes to the state **H**.

The **H** state (hazard) means that the system is in the hazard state. The hazard state is detected (e.g., by the comparators), because the output vectors are not identical. Both FPGAs have to be reconfigured in this case. The repair rate is equal to $\mu/2$, because each FPGA is being reconfigured separately. If it is possible to reconfigure both FPGAs at the same time, the availability parameters will increase.

$$\begin{aligned} 2s\lambda\, p_O - \mu s\, p_F - \frac{\mu s\, p_H}{2} &= 0 \\ \mu s\, p_F - 2s\lambda FS\, p_O &= 0 \\ \frac{\mu s\, p_H}{2} - 2s\lambda(1-FS)\, p_O &= 0 \\ p_O + p_F + p_H &= 1 \end{aligned} \qquad (1)$$

The described model introduces four parameters: the failure rate (λ), the repair rate (μ), the fault security (***FS***) and the configuration memory size (s). These parameters are discussed in the next section. Now let us transform the Markov model into a system of equations describing the steady state probabilities of each of the states (Equations 1). The system of equations is completed with a normalisation condition.

$$A_{SS} = p_O + p_F \qquad (2)$$

The value of the steady-state availability A_{SS} is a sum of probabilities for all working states (Equation 2).

4. FC-Min

The output grouping method is based on the FC-Min minimizer principles [11, 12]. FC-Min has been developed to efficiently handle functions with a large number of output variables. The minimization is being conducted in a reverse way than the standard minimizers do. First, the group cover of the on-set of all functions is found, independently on the source implicants. After that the minimized implicants are produced by processing the source implicants, in order to satisfy (and validate) the cover. Thus, group implicants are generated directly, not like in other minimization methods by reducing prime implicants of single functions.

This approach makes FC-Min a very fast two-level group minimizer, since only implicants that will be a part of the final solution are produced.

The minimization process consists of two processes: the *Find Coverage* algorithm and *Implicants Generation*.

4.1. The Find Coverage Algorithm

The Find Coverage algorithm is the essential phase of FC-Min. The whole cover of the on-set of the multi-output function is found, using the output part of the source function only. The algorithm tries to find a cover of the on-set by finding a rectangle cover [16] of all the "1" values in the output matrix (description of the function's on-set), and then it generates implicants having the properties given by this cover.

An example of such a cover is shown in Fig. 3. There is shown a 5-input and 5-output function defined by 10 terms, in a form of a truth table. The rest out of the total 32 terms is assigned as don't cares. The result of the Find Coverage algorithm is a cover consisting of six *coverage elements*, $t_1 - t_6$. A coverage element is a Cartesian product of two sets, the *coverage set* $C(t_i)$ and the *coverage mask* $M(t_i)$. The coverage set describes the rows that are covered by t_i, the coverage mask gives the output variables covered by t_i. Our example coverage elements are shown in Tab. 1.

Each coverage element describes a potential implicant. For example, the *group* term (implicant) t_1 covers "1"s of the fourth and fifth output variable (y_3 and y_4) in vectors 4, 6 and 8. Let us note that the structure of the terms is not known yet; only the set of covered "1"s is known. Now it is apparent, that if we succeed in finding the implicants having the properties of $t_1 - t_6$ (i.e., the terms cover the appropriate "1"s), the solution will consist of six implicants. To solve the coverage finding problem we use a greedy heuristic, since it is NP-hard, see [12] for details.

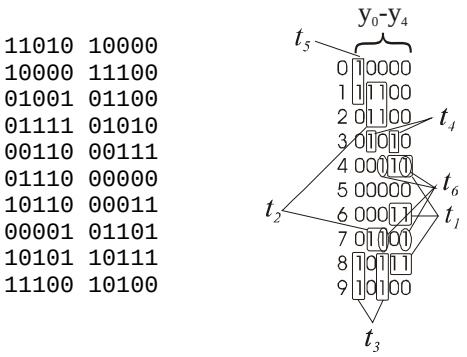

Figure 4: Cover of the output matrix

Table 1: Coverage elements from Fig. 4

Implicant	$C(t_i)$	$M(t_i)$
t_1	{4, 6, 8}	$\{y_3, y_4\} \equiv 00011$
t_2	{1, 2, 7}	$\{y_1, y_2\} \equiv 01100$
t_3	{8, 9}	$\{y_0, y_2\} \equiv 10100$
t_4	{3}	$\{y_1, y_3\} \equiv 01010$
t_5	{0, 1}	$\{y_0, y_1\} \equiv 10000$
t_6	{4, 7}	$\{y_2, y_4\} \equiv 00101$

4.2. Implicant Generation

After each coverage element is produced, it has to be validated, i.e., we must verify, whether there exist an implicant covering the "1"s in $C(t_i) \times M(t_i)$. This is done by directly generating the respective implicant. If this process fails, the coverage element is discarded and another one is computed.

Considering the conditions described above, particularly the definition of the rows each cover element should cover ($C(t_i)$), a simple rule the implicants have to satisfy can be derived: the *minimum implicant* satisfying the particular cover can be constructed as a *minimum supercube* of all the input vectors corresponding to the rows of the cover of t_i. Moreover, this supercube must not intersect any term that is not included in the particular cover $C(t_i)$, since it would cover some zeros then. In our example, a minimum implicant t_1 would be `(-01--)`, because of:

```
00110
10110
10101
-01--
```

5. The Output Grouping

The idea of grouping the multiple-output function's outputs to form multiple parity bits is straightforward: we try to group together outputs having many *common group implicants*. Such outputs will more likely share some terms, thus grouping them together would be advantageous for a two-level minimization of the source multi-output function. We have found experimentally that the same effect can be observed for a multi-level synthesis as well; the outputs sharing many group implicants share a lot of logics in the multi-level implementation of the function as well [13]. When these outputs are connected by a XOR gate to form a parity bit, the overall logic could be furthermore reduced, since only one output needs to be produced then.

The main output grouping idea is simple: first, we perform a two-level minimization of the unmodified multi-output function. Then we identify the output variables to be grouped together by evaluating the numbers of group implicants common to the outputs, the outputs in each group of outputs are joined into one XOR gate and the whole circuit is resynthesized by SIS [17] to obtain a multilevel network (or LUTs) or by ESPRESSO [18] for a two-level implementation of the parity generator.

5.1. Grouping Matrix

As it was stated before, the grouping of the outputs is derived from the valid coverage of the on-set. Since there are often big numbers of possible group implicants (coverage elements) and output variables, it is not easy to combine the influences of the implicants. We have found that an efficient way to estimate the grouping of the outputs is by constructing a *grouping matrix G*. It is a symmetric matrix of dimensions [*m*, *m*], where *m* is the number of output variables. The value G[*i*, *j*] defines the "binding strength" of the two output variables *i* and *j*.

The G matrix is being constructed during the coverage generation process. Firstly, the matrix is filled with zeros. After each valid coverage element is produced, the values in all the positions in G corresponding to all the couples of variables in $M(t_i)$ are increased by one. In our example (Fig. 4), after t_1 is found, the cells G[3, 4] and G[4, 3] are set to one. This describes an increased likelihood that the outputs y_3 and y_4 will be grouped together. The whole G-matrix computation process is shown in Fig. 5.

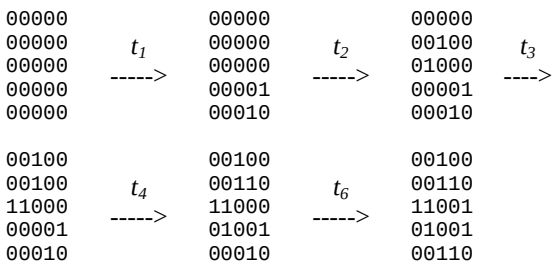

Figure 5: G-matrix construction

It is a very simple example, however, in practice the G-matrix mostly contains values greater than 1. Greater values indicate that the respective two variables have more than one common implicants in the solution.

5.2. Deriving the Output Grouping

There have been no assumptions or requirements for the number of parity bits (i.e., groups of outputs) until now. The G-matrix just describes the binding strengths of every two function's outputs. Now the distribution of the function's outputs among the groups has to be found. Let us note that *any* number of groups (parity bits) can be generated by this method, according the designer's needs.

We use a simple greedy algorithm. First, we compute the *nominal group size N*, by dividing the number of function's outputs by the number of required parity bits. This would be the average number of outputs forming one parity bit. Then the algorithm proceeds as follows: first, we find the *maximum* value in the G-matrix, let it be G[*i, j*]. When there are more possibilities for a choice, one is selected at random. Both the respective output variables (*i, j*) are assigned to the first group. After that we look for the next highest value in the *i*-th and *j*-th G-matrix rows, thus we find the output that "suits most" to one of the two selected ones. This new output is added to the group under construction. This process is repeated until *N* outputs are assigned to the group. Then we repeat the process from the beginning, to generate all the groups.

6. Experimental Results

6.1. The Overall Synthesis Process

The overall synthesis process, i.e., the way how all the tests have been performed will be described in this subsection.

The source functions for our experiments were the MCNC [19] benchmark circuits. The parity generator design process has been held in the following steps:

1. First, the MCNC benchmark described as a PLA structure has to be pre-processed, in order to generate the function's on-set and off-set, which is needed for FC-Min. This is done by ESPRESSO [18].
2. The circuit is then processed by FC-Min, to generate its group implicants.
3. The grouping matrix and, subsequently, the grouping of the outputs, is derived from the group implicants.
4. The obtained groups of outputs are XORed, to obtain the parity bits. This is done by converting the original circuit's PLA into a BLIF [17] file by SIS [17] and appending the XOR gates to the outputs.
5. The obtained parity generator is resynthesized by SIS, in order to obtain its PLA description (by collapsing the network), or to decompose it into LUTs. The number of literals in the SOP (sum-of-products) form for a PLA and the number of LUTs are counted then, to make an estimation of the size of the parity generator. Since the design is targeted to FPGAs, only 4-LUTs will be considered from now on.

6.2. The Efficiency of the Method

In order to evaluate the efficiency of the proposed method, we have compared the FC-Min based parity bits grouping with a purely random grouping. We have tested the method on standard MCNC benchmark circuits [19]. We have varied the number of parity bits from one to the number of the circuit's outputs. One limit, the 1-parity bit case, involves XORing all of the circuit's outputs, thus any "smart" output grouping method cannot come into effect. The second limit case, i.e., the number of parity bits equal to the number of outputs, corresponds to the original circuit (no XORs).

For each benchmark circuit and a given number of parity bits, 500 random and 500 FC-Min based output groupings have been generated and the average of each was taken.

A typical growth of the number of look-up tables (LUTs) for the FPGA realization obtained by SIS [17] is shown in Fig. 6 for a sqr6 MCNC [19] benchmark circuit. The size of the circuit grows with the number of parity bits here. It can be concluded that by XORing the circuit's output its size is reduced after resynthesis; producing the parity bits only is advantageous, with respect to the total area.

On the other hand, Fig. 7 shows the alu2 benchmark results. Here the number of LUTs increases with decreasing the number of the parity bits, thus adding XORs to the circuit inputs involves the circuit size growth, even after the resynthesis. Such benchmarks typically are hard-to-synthesize functions with many XOR gates, like ALUs. Adding XOR gates to their output just increases their complexity and, moreover, standard synthesis tools, like SIS [17] and ESPRESSO [18] are not able to handle such circuits efficiently [20]. Fortunately, such cases are quite rare, see Tab. 2.

Two curves are shown in figures 6 and 7. One curve corresponds to the FC-Min based output grouping, one to a random grouping. We can see that the FC-Min grouping always produced a circuit having fewer literals. Of course, the curves meet at the two limit

cases (no parity and 1-parity bit), since no grouping is involved in these cases.

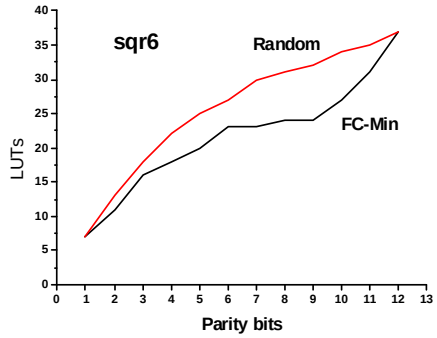

Figure 6: The sqr6 MCNC example

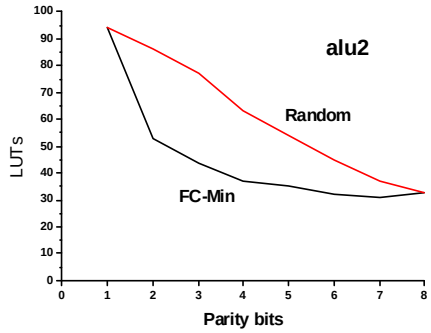

Figure 7: The alu2 MCNC example

The summary results obtained from several MCNC benchmarks [19] are shown in Tab. 2. The "*Bench*" column shows the benchmark name, the number of its outputs follows (*m*). The ratio of the size of a 1-parity generator to the original circuit is shown in the next column ("*Ratio*"). 100% means no difference, circuits having the ratio less than 100% correspond to the Fig. 6 case (i.e., the area is reduced by XORing the circuit's outputs), ratios higher than 100% correspond to the Fig. 7 case. It can be seen that the area of some benchmarks is rapidly reduced after XORing the outputs (newbyte, p82, t3, tms), however, for some benchmarks, the area is drastically increased (for alu1 almost 40x!). As we have stated before, this is due to a fact that the added XOR gate complicates the logic after resynthesis to such extend, so that standard synthesis tools are not able to handle these functions efficiently [see 20].

The average and maximum improvement obtained by our output grouping method, with respect to the random grouping is shown in the next two columns, in terms of the number of LUTs, obtained by SIS [17]. The number of parity bits, where the maximum improvement was reached is indicated in the parentheses in the "*Max. impr.*" column. Measurements equal to these made to obtain figures 6 and 7 have been performed for all the benchmark circuits. All the respective dependency curves were similar to these shown in figures 6 and 7, the FC-Min based grouping method always gave better or approximately equal results than the random based approach.

It can be seen that the FC-Min based output grouping method yields a substantial improvement with respect to the random method, so we can say it is efficient. However, for some benchmark circuits, the average improvement is negligible or even negative (e.g., ex1010, f51m, inc, in0, newpla, t3). These are probably mostly symmetric functions, where any "smart" output grouping method cannot help.

Table 2: Output grouping results

Bench	*m*	*Ratio*	*Avg. impr.*	*Max. impr.*
alu1	8	3950.0%	11.7%	37.9% (4)
alu2	8	284.8%	25.3%	42.9% (3)
alu3	8	356.7%	3.5%	8.9% (4)
apla	12	32.2%	13.8%	28.0% (4)
b10	11	23.7%	4.7%	12.9% (8)
b12	9	255.2%	12.7%	38.0% (2)
bc0	11	21.8%	11.0%	24.3% (8)
br1	8	35.9%	7.4%	18.9% (5)
br2	8	15.7%	13.4%	36.4% (4)
dk17	11	38.2%	7.9%	14.7% (9)
dk27	9	50.0%	4.6%	14.3% (2)
dk48	17	100.0%	2.4%	9.1% (5)
ex1010	10	14.3%	0.1%	3.0% (7)
exp	18	16.4%	10.2%	19.1% (12)
f51m	8	51.4%	-2.1%	0.0% (0)
gary	11	21.2%	7.9%	8.4% (8)
in0	11	21.1%	0.8%	9.0% (9)
in2	10	33.3%	6.1%	21.3% (7)
in5	14	77.4%	6.1%	21.3% (3)
in7	10	86.2%	30.4%	51.5% (2)
inc	9	16.7%	-1.0%	9.1% (6)
m1	12	11.1%	6.8%	33.3% (4)
m2	16	13.3%	15.6%	32.4% (4)
m3	16	13.9%	7.1%	24.5% (4)
m4	16	16.8%	12.6%	28.9% (5)
mlp4	8	20.0%	6.6%	26.9% (6)
mp2d	14	151.4%	25.1%	42.0% (3)
newapla	10	14.3%	0.1%	3.0% (7)
newbyte	8	6.3%	11.7%	37.9% (4)
newcpla	16	52.4%	17.4%	28.2% (6)

Bench	*m*	*Ratio*	*Avg. impr.*	*Max. impr.*
newcpla2	10	19.4%	29.9%	53.3% (3)
p82	14	8.6%	4.3%	20.0% (10)
sex	14	47.4%	23.4%	38.1% (9)
sqr6	12	18.9%	13.9%	25.0% (9)
t2	16	102.9%	17.4%	43.8% (2)
t3	8	2.7%	-0.9%	0.0% (0)
t4	8	192.9%	6.8%	28.6% (4)
tms	16	8.3%	10.1%	27.3% (10)

6.3. The Dependability Parameters

The parity net grouping methodology is used to increase the dependability of the system based on the MDS architecture. The availability computations were used to compare our modified duplex system with a standard duplex system and with TMR (Triple Modular Redundancy) system. Availability is a function of a time, A(*t*), defined as the probability that a system is operating correctly and is available to perform its functions at an instant of a time *t*. This section follows the Section 3 describing our modified duplex system with the Markov model and with dependability equations.

Firstly, the model parameters are discussed. The failure rate (λ) depends on the probability that the impacting SEUs will change a bit in the FPGA configuration memory. The effect of the SEUs impacting on random access memory RAM is described in [7]. In this article authors tested many systems with different size and type of memory and calculated SEU failure rate. In our calculation we have taken into account results presented in [7] and we set the "failure rate" parameter to:

$$\lambda = 1.8\,e^{-5}\,[h^{-1}] \qquad (3)$$

We assume more than one device with embedded RAM, therefore the failure rate parameter was increased.

The repair rate ($\boldsymbol{\mu}$) depends on the time needed for the reconfiguration of an FPGA. The clock frequency was set to 25 MHz. The configuration memory size ***s*** (needed for each benchmark) was calculated as a product of the configuration memory size for AT94K40 ATMEL FPSLIC and the circuit area overhead (*AO*[%]).

$$s = 233k \cdot AO\,[bits] \qquad (4)$$

Dependability calculations are processed firstly for a single parity and then for a multiple parity. In the multiple parity case 2 or 3 parity nets were selected.

Our results of improved availability parameters are shown in Tab. 3. Here "*Bench*" is the name of the benchmark circuit, "*AO*" is the area overhead, "*FS*" is the probability that a fault is detected by a code word, "*ASS*" is the steady-state availability and "*Impr. ASS*" indicates the improvement of *ASS* against single parity when multiple parity is used.

The availability of the original duplex system is 0,999978249. The availability parameter is the same as for the triplex system in the case when FS property is 100%.

Our results show that area overhead is higher in a case when we use multiple parities. Due to more parity nets increase observability of the tested benchmark the FS parameter is higher. The value of the FS parameter depends on the used algorithm to create parity nets.

7. Conclusions

We have proposed an efficient method to design a multiple parity generator for on-line BIST. The method is based on properly choosing the original circuit's outputs to be XORed to obtain respective parity bits. The choice is being done by determining outputs that share many group implicants in the two-level representation of the multi-output function. These outputs share a lot of combinational logic and, most likely, the amount of the overall logic would be decreased when these outputs would be joined together by a XOR gate.

The availability parameters of the MDS architecture based on self-checking circuits have been calculated. The results show that using the multiple parity bits increase the availability parameters at the price of a higher area overhead, with respect to the single parity case.

The efficiency of the method has been approved by an experimental evaluation on standard MCNC benchmark circuits.

References

[1] ----------, "FPGA Based Design of Raiway's Interlocking Equipment", In Proc. of EUROMICRO Symposium on Digital System Design, Rennes (FR), 31.8. - 3.9. 2004, pp 467-473.

[2] D. Ratter, " FPGAs on Mars", www.xilinx.com,Xcell Journal Online, 2004.

[3] Actel Corporation.:"Historic Phoenix Mars Mission Flies Actel RTAX-S Devices", www.actel.com, 2007.

[4] L. Sterpone and M. Violante, "A design flow for protecting FPGA-based systems against single event upsets ", DFT2005, 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 436 – 444.

[5] QuickLogic Corporation.: Single Event Upsets in FPGAs, 2003, www.quicklogic.com

[6] M. Bellato, P. Bernardi, D. Bortalato, et al., "Evaluating the effects of SEUs affecting the configuration memory of an

SRAM-based FPGA", Design Automation Event for Electronic System in Europe 2004, pp. 584-589.

[7] E. Normand, "Single Event Upset at Ground Level," IEEE Transactions on Nuclear Science, vol. 43, 1996, pp. 2742-2750.

[8] ----------,, "Dependable Design for FPGA based on Duplex System and Reconfiguration", In Proc. of 9th Euromicro Conference on Digital System Design, Los Alamitos: IEEE Computer Society, 2006, pp. 139-145.

[9] D.K. Pradhan, "Fault-Tolerant Computer System Design", Prentice-Hall, Inc., New Jersey, 1996.

[10] ----------,, "Output Grouping Method Based on a Similarity of Boolean Functions", Proc. 7th Int. Workshop on Boolean Problems (IWSBP'06), Freiberg, Germany, 21.-22.9.2006, pp. 107-113

[11] ----------,. FC-Min: A Fast Multi-Output Boolean Minimizer, Proc. 29th Euromicro Symposium on Digital Systems Design (DSD'03), Antalya (TR), 1.-6.9.2003, pp. 451-454.

[12] ----------,, "Boolean Minimizer FC-Min: Coverage Finding Process", Proc. 30th Euromicro Symposium on Digital Systems Design (DSD'04), Rennes (FR), 31.8. - 3.9.04, pp. 152-159.

[13] ----------,, "Output Grouping-Based Decomposition of Logic Functions":, Proc. 8th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop 2005 (DDECS'05), Sopron, HU, 13.-16.4.2005, pp. 137-144.

[14] ----------,, "Fault Tolerant System Design Method Based on Self-Checking Circuits", Proc. 12th International On-Line Testing Symposium 2006 (IOLTS'06), Lake of Como, Italy, July 10-12, 2006.

[15] ----------,, "Minimization of the Hamming Code Generator in Self Checking Circuits", Proceedings of the International Workshop on Discrete-Event System Design (DESDes'04). Zielona Gora: University of Zielona Gora, 2004, s. 161-166.

[16] S. Hassoun and T. Sasao, „Logic Synthesis and Verification", Boston, MA, Kluwer Academic Publishers, 2002, 454 pp.

[17] E.M. Sentovich et al. "SIS: A System for Sequential Circuit Synthesis", Electronics Research Laboratory Memorandum No. UCB/ERL M92/41, University of California, Berkeley, CA 94720, 1992.

[18] R.K. Brayton, et al. „Logic Minimization Algorithms for VLSI Synthesis", Boston, MA, Kluwer Academic Publishers, 1984.

[19] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide", Technical Report 1991-IWLS-UG-Saeyang, MCNC, Research Triangle Park, NC, January 1991

[20] J. Cong and K. Minkovich, "Optimality Study of Logic Synthesis for LUT-Based FPGAs", Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on CAD, Vol. 26, Issue 2, Feb. 2007, pp. 230 – 239.

Table 3: Improved availability parameters

Bench.	*Single parity*			*Multiple parity*			*Impr. ASS*
	AO	*FS*	*ASS*	*AO*	*FS*	*ASS*	
alu1	3337.5%	100.0%	1	275.00%	100.00%	1	0.0%
apla	40.5%	74.3%	0.999988965	76.19%	87.21%	0.999991356	18.2%
b10	26,7%	92,6%	0,999997416	44,40%	95,83%	0,999998095	3,4%
b12	95.8%	95.9%	0.999996581	179.17%	98.08%	0.999996779	1.1%
dk17	41.9%	84.9%	0.999993387	100.00%	95.23%	0.999995824	13.9%
dk48	96.7%	88.7%	0.99999049	103.33%	91.91%	0.999992718	15.4%
ex1010	7.3%	81.7%	0.999995417	19.47%	89.59%	0.99999677	7.3%
ex7	246,2%	97,6%	0,999993744	328,21%	98,84%	0,999995215	8,7%
f51m	50.0%	87.2%	0.999993736	72.22%	88.48%	0.999992583	-7.4%
gary	25.3%	90.6%	0.99999679	54.36%	94.94%	0.999997356	3.0%
inc	15,9%	86,2%	0,999995968	43,18%	93,05%	0,999996922	5,1%
m1	9.7%	84.0%	0.999995812	29.03%	97.44%	0.999999059	15.6%
m3	33,3%	93,7%	0,999997565	60,32%	97,41%	0,999998547	4,8%
mp2d	61.3%	88.2%	0.999993322	87.10%	92.96%	0.99999467	8.2%
mlp4	17,8%	94,5%	0,99999834	49,50%	97,64%	0,999998833	2,4%
newapla	43.8%	85.3%	0.999993388	75.00%	92.81%	0.999995204	10.7%
newbyte	11.1%	100.0%	1	33.33%	100.00%	1	0.0%
newcpla1	54.3%	90.3%	0.999994977	47.83%	94.94%	0.999997577	13.5%
newcpla2	25.0%	75.7%	0.999991741	58.33%	86.96%	0.999992914	8.0%
p82	14.7%	85.3%	0.999995793	20.59%	90.33%	0.999996931	6.1%
sex	57.9%	83.6%	0.999991106	84.21%	92.35%	0.999994391	20.4%
sqr6	14.3%	94.9%	0.999998551	45.24%	96.87%	0.999998578	0.1%
t2	56,3%	91,1%	0,999995271	81,25%	92,69%	0,999994781	-2,9%
tms	8,1%	84,9%	0,999996162	23,23%	91,32%	0,999997128	5,1%