Fault Tolerant Design of Random Logic based on a Parity Check Code*

Sobeeh Almukhaizim and Yiorgos Makris
Electrical Engineering Department
Yale University
{sobeeh.almukhaizim, yiorgos.makris}@yale.edu

Abstract

We describe a method for designing fault tolerant circuits based on an extension of a Concurrent Error Detection (CED) technique. The proposed extension combines parity check codes and duplication in order to not only perform error detection but also provide diagnosis and correction capabilities. Informed selection among the outputs of the original circuit and the outputs of the duplicate with parity check codes renders a low-cost fault tolerant design. Experimental results confirm the efficacy of the proposed method as a general solution for designing fault tolerant circuits.

1. Introduction

Complex electronic circuits are currently utilized in safety critical applications, where reliability is of paramount importance. While manufacturing test typically identifies a large number of circuit defects, exhaustive testing is impractical and attaining complete fault coverage may not be feasible. Design-For-Testability (DFT) techniques are used to remedy low fault coverage [1, 2], albeit at the cost of additional hardware area. Yet undetected manufacturing defects, wear-and-tear faults, as well as transient errors still pose a threat to the reliable operation of the circuit. To shield against such faults, CED techniques [3, 4, 5, 6, 7, 8] are used to detect malfunctions during the lifetime of the circuit. While the cost of CED techniques is often twice the cost of the original design, such techniques can only report the occurrence of a fault and may not take any remedial action. Should the circuit need to remain operational in the presence of a fault, a fault tolerant design is required. Fault tolerant designs concurrently *detect*, *diagnose* and *correct* a fault effect, at the cost of either performance degradation or considerable hardware overhead. For example, Triple Modular Redundancy (TMR) [9], a standard fault tolerance method, comes at three times the cost of the original circuit.

The distinct objectives of CED and fault tolerance have resulted in quite different solutions to the two problems. In an effort to reduce this gap, we examine in this paper the extension of a standard CED technique into a method for designing fault tolerant circuits. The choice of the starting-point CED technique is based on its effectiveness in terms of fault model assumed, feasibility of implementation, performance overhead and diagnostic capabilities. The CED technique must be able to detect any single fault and, still, be efficiently implemented for large circuits. Moreover, the chosen CED technique must inherently provide some level of fault identification in order to assist the diagnosis and correction operation. Finally, the fault tolerance extension to the CED technique needs to derive adequate information for both detecting and correcting the fault on the fly, in order to attain the same performance as the original circuit.

Among the several CED techniques that have been proposed in the literature, we select one that meets most of the aforementioned requirements and we extend it with the minimum number of components necessary to perform fault diagnosis and correction. Section 2 describes the selected CED technique that fits our requirements and explains the underlying features that assist the construction of a fault tolerant circuit. The added components and the proposed extension for fault tolerance are outlined in section 3. Experimental results evaluating the proposed design are presented in section 4 and conclusions are drawn in section 5.

2. CED using Parity Check Codes

The idea of multiple parity bits was first introduced in [10]. A parity check code is a code in which the parity of multiple circuit outputs, forming a parity group, is checked against a predicted parity bit for that group. The objective is to classify the outputs in a minimal number of groups, such that any single fault in the circuit will affect the parity of at most one output bit in every parity group. To ensure that the fault effect will be detected, no sharing is allowed between the cones of logic of output bits belonging to the same parity group. The two extreme cases for the number of parity groups are *single-bit* parity and *duplication*. In single-bit parity, all output bits of the circuit form a single group and, consequently, no sharing between their cones of logic is allowed. In duplication, on the other hand, every output bit is considered a group by itself and, thus, there are no constraints on logic sharing. Nevertheless, both of these extremes may incur significant hardware overhead and may not lead to an acceptable solution.

*This work is partially supported through a fellowship from Kuwait University.

Predicting the value of the parity bit in the single-bit parity case is relatively inexpensive. However, the prohibition of sharing between the output cones of logic adds a significant overhead to the cost of the original circuit, which needs to be intrusively re-synthesized under this constraint. Duplication, on the other hand, leaves the original circuit intact, yet incurs the cost of an additional copy of the circuit to predict the parity of each group (which in this case is equal to the actual output of the circuit). The possibility of finding a more cost-effective solution in between the two extremes has motivated several research efforts. The overall goal is clear; minimize the area required to implement the parity check code. The area required by the parity check code is equal to the sum of the cost of the logic function, the parity prediction logic, and a parity checker. As the number of groups increases, the cost of the parity checker and parity prediction logic increase as well, while the cost of the original circuit decreases, as more logic sharing is permitted between the outputs. On the other hand, reducing the number of groups decreases the cost of the parity checker and the parity prediction logic, while increasing the cost of the original circuit due to reduced logic sharing.

One of the first efforts in finding an optimal point between the two extreme cases was developed by De *et al.* [5]. The circuit outputs are partitioned to form logic blocks where no logic sharing is allowed between blocks but sharing is maximized within the block. A cost function that is based on logic sharing in the optimized circuit implementation guides the partitioning process. The number of parity groups depends on the logic block with the highest number of outputs. Every group contains no more than a single output from every logic block. Any single fault will affect outputs of a single block and, as logic block outputs are in different parity groups, the fault effect will not be masked.

The synthesis method proposed in [5] depends on the number of output partitions; a user supplied parameter. The greedy cost function considers only the area required by the original function. Touba *et al.* [6] enhanced the efficiency of the solution by employing a cost function that takes into account the cost necessary for the logic of the original function, the parity prediction, and the parity checker. The proposed solution automates parity check code selection and allows sharing between outputs in different logic blocks, as long as the outputs belong to different parity groups, which was a restriction in the method proposed in [5].

Zeng *et al.* [7] extended the parity check code method of [6] to design Finite State Machines (FSMs) with CED capabilities. Since states can be represented symbolically, CED circuitry can be added to the next state logic during state assignment, after the assignment but before logic optimization, or after logic optimization. A new state encoding technique that encodes the states with the objective of reducing the area overhead of the self-checking FSM is introduced.

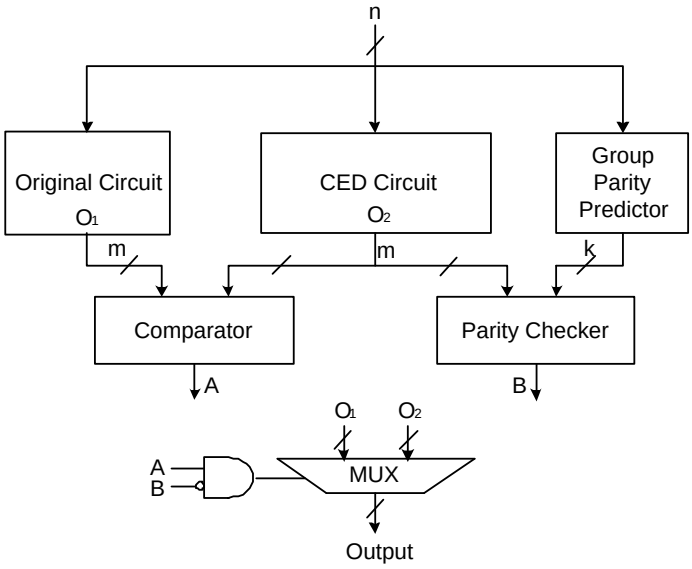

Figure 1. Proposed Methodology.

The next state parity check is done one clock cycle later to also detect faults in the bistable elements. The output logic and next state logic belong to different parity groups, and hence, logic sharing between them is allowed.

Parity check codes appear to be the most appropriate choice for our purpose of extending a CED method to perform fault diagnosis and fault correction. In accordance with the requirements set forth in the introduction, parity check codes allow the detection of all single faults, which constitutes a reasonable fault model. Moreover, they incur tolerable area overhead and are feasible to implement for large circuits [6]. In addition, parity check codes reveal some fault identification information that, when used properly, may assist with fault diagnosis and fault correction. In the next section, we show how this information may be utilized in order to design fault tolerant circuits.

3. Proposed Extension for Fault Tolerance

The result of the comparison between the predicted group parity and the actual parity indicates the presence of a fault, yet it provides us with no information regarding the fault source. While a fault in the circuitry implementing the desired logic function leads to faulty circuit results, potential faults in the parity prediction logic will not affect correctness of the output. Yet, the CED method will still indicate detection of a fault, rendering the circuit results unusable. Therefore, we need a mechanism to distinguish between faults in the parity prediction logic and faults in the original circuit. Furthermore, independent of where the fault is, this mechanism should generate the correct result.

Figure 1 illustrates the proposed fault tolerant design. The group-parity CED technique, as described in [6], consists of the CED circuitry, the group parity predictor, and

A	B	Possible Fault Source	Action
0	0	No Error, All agree	Output O_1 *or* O_2
0	1	Error in Parity Predictor *OR* Parity Checker	Output O_1 *or* O_2
1	0	Error in Original Circuit *OR* Comparator	Output O_2
1	1	Error in CED Circuit	Output O_1

Figure 2. Diagnosis and Correction Actions.

a parity checker. Two components are added to extend the group-parity CED technique for fault tolerance: the original area-optimized circuit and an output comparator. The two copies of the circuit, along with the CED capability, provide adequate information for generating correct results, even in the presence of a fault. The parity checker and the comparator produce the control signals required to diagnose and correct a fault.

During fault-free operation, the two inputs of the comparator will always agree. Similarly, the two inputs of the parity checker will also agree. In case of a discrepancy in the parity checker but not the comparator, a fault is detected in the parity predict logic or in the parity checker, respectively. If the comparator inputs disagree, while the parity checker confirms the correctness of the group parity, a fault is present either in the original circuit or in the comparator. When both the comparator and parity checker detect a discrepancy and under a single fault assumption, a fault is detected in the CED circuit. The comparison results of the parity checker and the comparator are used to diagnose the fault source and select the correct output. All the previous possible scenarios are restated in the table of figure 2. The correct output can be easily produced using the circuitry shown at the bottom of figure 1. We should note at this point that this small circuit is the Achilles heel of the proposed fault tolerant method, just as a majority voter is the weak point of TMR. In both cases, faults in this logic can be neither detected nor corrected.

The proposed technique may be easily extended to sequential logic, by extending the method proposed in [7] for CED in FSMs based on parity check codes. To make the circuit fault tolerant, the original area-optimized FSM implementation and an output comparator are added to detect faults in the next state and output logic. In [7], the parity check bits are stored in bistable elements to also detect faults in the state register; however, this is not possible in a fault tolerant implementation as the check is performed a cycle later, while the next state logic is evaluating the next state using an erroneous present state. Moreover, although the fault is detected it may not be corrected without performing a re-computation that will slow down the circuit operation. If the detection is performed before the next state is stored in the bistable elements, then correction can be added in the same cycle. Unfortunately, this will not allow us to detect any faults in the bistable elements.

In summary, the proposed method requires three additional hardware components in order to make a combinational or sequential circuit fault tolerant: a replica of the circuit synthesized with CED based on group parity, a comparator, and the small output selection hardware. In comparison to TMR, the proposed method provides a substantial hardware reduction, as indicated through the experimental results provided below.

4. Experimental Results

The generation of CED circuits using parity check codes requires modification of the synthesis tool to prevent logic sharing between outputs belonging to the same parity group. We are currently implementing the required modifications in the SIS [11] synthesis tool, in order to implement the code selection and restructuring algorithms described in [6]. In the meantime, and for the purpose of estimating the hardware overhead of the proposed method, we use the results presented in [6] for MCNC combinational benchmarks. Final results confirming the current estimates will be provided in the final manuscript.

The literal count of the original circuit, the fault tolerant circuit with TMR, and the proposed fault tolerant method are provided in the table of figure 3. The first major heading in the table describes each benchmark circuit considered: number of primary inputs, number of primary outputs and literal count. The second major heading summarizes the TMR implementation: the number of parity bits used, the literal count for the three copies of the circuit, the literal count for the checker, and the total literal count. As no output grouping is performed in the TMR version, the number of parity bits equals the number of primary outputs. The third heading summarizes the proposed method: the number of parity bits, the literal count for the circuit (original circuit, CED version, and group parity predictor), the literal count for the checker (parity checker and comparator), and the total literal count. The last major heading in the table of figure 3 provides the area overhead reduction of the proposed scheme as compared to TMR, indicating that the proposed scheme provides an average saving of 14.62%.

5. Conclusion

In conclusion, we presented a method to construct fault tolerant designs for random logic. The proposed methodology extends a parity check code based CED method by appending the required components to perform fault diagnosis and fault correction. CED using parity check codes is a key component of the proposed method, as it inherently

Original Circuit				TMR FT Circuit				Proposed FT Circuit				
Name	PI	PO	Lits. Count	Parity bits	Lits. Count	Check. Lits.	Total Count	Parity bits	Lits. Count	Check.. Count	Total Count	Hardware Reduction
apla	10	12	312	12	936	176	1112	3	628	140	768	32.91
br1	12	8	196	8	588	112	700	2	439	88	527	25.34
bw	5	28	178	28	534	432	966	8	480	352	832	10.11
chkn	29	7	398	7	1194	96	1290	3	1104	80	1184	7.54
dc1	4	7	45	7	135	96	231	3	107	80	187	20.74
dc2	8	7	162	7	486	96	582	2	350	76	426	27.98
exp	8	18	435	18	1305	272	1577	5	961	220	1181	26.36
luc	8	27	211	27	633	416	1049	6	546	232	778	13.74
p82	5	14	127	14	371	208	579	3	289	164	453	22.10
signet	39	8	335	8	1005	112	1117	5	645	100	745	35.82
wim	4	7	60	7	180	96	276	2	121	76	197	32.78
5xpl	7	10	134	10	402	144	546	3	351	116	467	12.69
alu4	14	8	800	8	2400	112	2512	8	1600	112	1712	33.33
b12	15	9	87	9	261	128	389	4	237	108	345	9.20
cmb	16	4	52	4	156	48	204	2	116	40	156	25.64
cu	14	11	53	11	159	160	319	1	136	120	256	14.47
f51ml	8	8	130	8	390	112	502	2	348	88	436	10.77
misex1	8	7	54	7	162	96	258	3	150	80	230	7.41
misex2	25	18	104	18	312	272	584	2	306	208	514	1.92
pc1e	19	9	69	9	207	128	335	3	225	104	329	-8.70
term1	34	10	149	10	447	144	591	7	505	132	637	-12.98
ttt2	24	21	191	21	573	320	893	9	565	272	837	1.40
x2	10	7	51	7	153	96	249	2	130	76	206	15.03

Figure 3. Experimental Results on the MCNC Benchmarks.

discloses information that assist the diagnosis and correction phases in the proposed technique and reduce the corresponding hardware overhead. The proposed method can be easily extended to handle fault tolerant design of sequential logic. Experimental results confirm the overhead reduction of the proposed technique, as compared to the traditional TMR fault tolerance approach.

References

[1] M. Geuzebroek, J. van der Linden, and A. van de Goor, "Test point insertion for compact test sets," in *Proceedings of the IEEE International Test Conference*, 2000, pp. 292–301.

[2] N. Tamarapalli and J. Rajski, "Constructive multiphase test point insertion for scan-based bist," in *Proceedings of the IEEE International Test Conference*, 1996, pp. 649–658.

[3] M. Gossel and S. Graf, *Error Detection Circuits*, McGraw-Hill, 1993.

[4] S. J. Piestrak, "Self-checking design in eastern europe," *IEEE Design and Test of Computers*, vol. 13, pp. 16–25, 1996.

[5] K. De, C. Natarajan, and P. Banerjee, "Rsyn: A system for automated synthesis of reliable multilevel circuits," in *Proceedings of the IEEE Transactions on Very Large Scale Integration Systems*, 1994, vol. 2, pp. 186–195.

[6] N. Touba and E. McCluskey, "Logic synthesis of multilevel circuits with concurrent error detection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 783–789, 1997.

[7] C. Zeng and E. McCluskey, "Finite state machine synthesis with concurrent error detection," in *Proceedings of the IEEE International Test Conference*, 1999, pp. 672–679.

[8] S. Mitra and E. McCluskey, "Which concurrent error detection scheme to choose?," in *Proceedings of the IEEE International Test Conference*, 2000, pp. 985–994.

[9] R. E. Lyions and W. Vanderkulk, "The use of triple modular redundancy to improve computer reliability," Tech. Rep., IBM J. Res. Develop., 1962.

[10] E. Sogomonyan, "Design of built-in self-checking monitoring circuits for combinational devices," *Automation and Remote Control*, vol. 35, no. 2, pp. 280–289, 1974.

[11] E. M. Sentovich et al., "SIS: a system for sequential circuit synthesis," ERL MEMO. No. UCB/ERL M92/41, EECS UC Berkeley CA 94720, 1992.