Fault Tolerant Design of Combinational and Sequential 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 synthesized circuit and the outputs of a constrained-sharing resynthesized 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

Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'03)
1063-6722/03 $17.00 © 2003 **IEEE**

IEEE COMPUTER SOCIETY

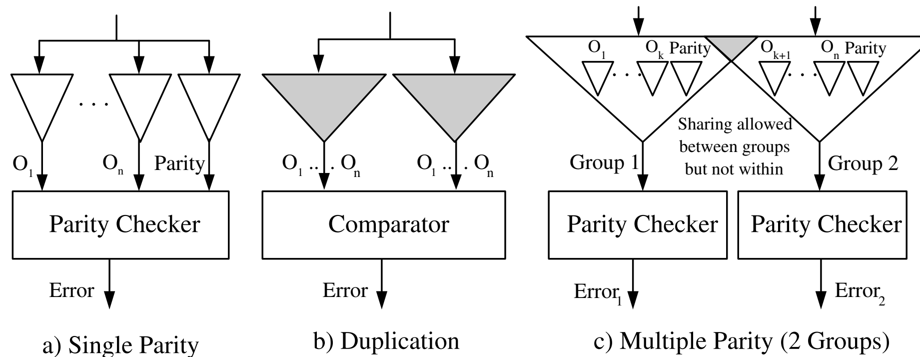


Figure 1. CED using Parity Check Codes Alternatives.

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* as illustrated in figures 1.a and 1.b, respectively. 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 and the parity checker reduces to a comparator. Nevertheless, both of these extremes may incur significant hardware overhead and may not lead to an acceptable solution.

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 illustrated in figure 1.c, the cost of the parity checker and parity prediction logic increases when the number of

Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'03)
1063-6722/03 $17.00 © 2003 **IEEE**
IEEE COMPUTER SOCIETY

groups increases 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. We denote the circuit generated with constraints on the sharing between the logic blocks as a *resynthesized circuit*. 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].

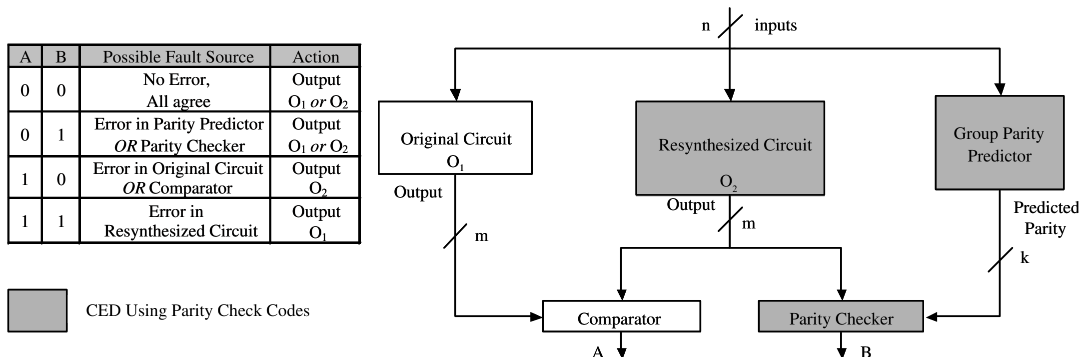
Zeng *et al.* [7] extended the parity check code method of [6] to design Finite State Machines (FSMs) with CED capabilities. Since states can be represented symbolically, CED circuitry can be added to the next state logic during state assignment, after the assignment but before logic optimization, or 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. 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 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 resynthesized circuit. Furthermore, independent of where the fault is, this mechanism should generate the correct result.

1063-6722/03 $17.00 © 2003 IEEE
IEEE COMPUTER SOCIETY



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 Resynthesized Circuit	Output O_1

Figure 2. Proposed Methodology for Combinational Circuits.

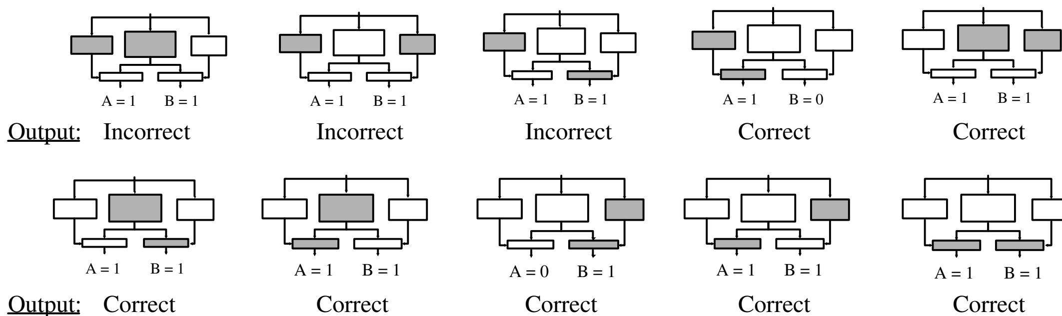


Figure 3. The behavior of two faulty components (non-Alias Cases).

3.1. Combinational Logic

The proposed fault tolerant design for combinational logic is illustrated in figure 2. The CED technique, as described in [6], consists of a resynthesized circuit, a group parity predictor, and a parity checker. Two components are added to extend the 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 faulty component. The parity checker and the comparator produce the control signals required to diagnose and correct a single faulty component.

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, the outputs of the two circuits are correct and a malfunction is detected either in the parity predict logic, the parity checker or both; single or multiple faults in these two components are tolerated in this case. If the comparator inputs disagree while the parity checker confirms the correctness of the group parity, a malfunction is detected either in the original circuit, the comparator or both; multiple faults in these two components are also tolerated in this case. When both the comparator and parity checker detect a discrepancy and under a *single faulty component* assumption, a fault is detected in the resynthesized circuit.

All the previous possible scenarios are restated in the table of figure 2. Similarly to TMR, the proposed method tolerates multiple faults in any single component. As discussed above, single and multiple faults in multiple components may or may not be tolerated. A TMR implementation, composed of the original circuit and two replicas, tolerates multiple faults in two components if the faulty response of the two faulty components is not the same, i.e. the majority for every output

IEEE COMPUTER SOCIETY

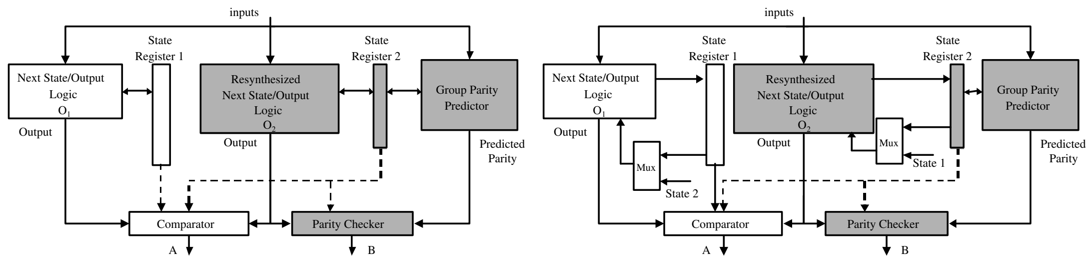


Figure 4. Proposed Methodology for Sequential Circuits.

bit over all the three components is correct. An output bit in the TMR implementation is faulty if two inputs to its majority voter are incorrect. We refer to such cases where the faulty responses are equivalent in some bit positions as *Alias Cases*. Similar to the TMR implementation, the proposed method has some Alias Cases where multiple faulty components can not be corrected. Nonetheless, the proposed method can tolerate faults in two components based on the effect of the faults on the control signals. Figure 3 illustrates all the possible scenarios for two faulty components, the associated control signals and correctness of operation. As seen from the figure, the proposed method can detect 6 out of 10 cases where two components are faulty. The correct output can be easily produced using a 2-to-1 multiplexer based on the aforementioned comparison results of the comparator and the parity checker. We should note at this point that the multiplexer is the Achilles heel of the proposed fault tolerant method, just as a majority voter is the weak point of TMR; a fault in both cases can be neither detected nor corrected.

3.2. Sequential Logic

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, as illustrated in figure 4. The parity checker and the comparator produce the same control signals of the previous section to diagnose and correct a fault.

The parity check bits are stored in bistable elements to also detect faults in the state register in [7]; 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, as shown in the left diagram of figure 4. Unfortunately, this will not allow us to correct any faults in the bistable elements. An alternative implementation that resolves this problem is illustrated in the left diagram of figure 4. In this case, we multiplex the correct state based on the value of the control signals A and B so that the correct previous state is used in the calculation of the next state. However, the clock cycle time must be increased to account for the delay it takes to compute A and B.

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 resynthesized 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.

IEEE COMPUTER SOCIETY

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 5. Experimental Results on the MCNC Combinational Benchmarks.

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 use the results presented in [6] for MCNC combinational benchmarks and [7] for MCNC sequential benchmarks in order to estimate the hardware overhead. Both methods have been implemented using SIS [11] synthesis tool to perform the code selection and restructuring algorithms necessary to prevent fault masking.

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 5 for combinational circuits. 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, resynthesized circuit, and group parity predictor), the literal count for the checker (parity checker and comparator), and the total literal count. The bold entries in the total count column correspond to the benchmarks were the proposed method performed better than TMR. The last major heading in the table of figure 5 provides the area overhead reduction of the proposed scheme as compared to TMR. An average saving of 18.42% is seen for all the benchmarks where the proposed method performs better.

IEEE COMPUTER SOCIETY

Circuit Name	Original Lits. Count	TMR Lits. Count	Proposed Lits. Count	Hardware Reduction
cse	183	675	**614**	9.04 %
dk14	101	387	**351**	9.30 %
dk15	69	277	**236**	14.80 %
dk16	239	801	**692**	13.61 %
dk17	57	227	**195**	14.10 %
dk27	24	114	**95**	16.67 %
dk512	56	238	**192**	19.33 %
ex1	221	**971**	1035	-6.59 %
ex2	74	278	**231**	16.91 %
ex3	25	117	**94**	19.66 %
ex4	70	364	**295**	18.96 %
ex5	12	64	**52**	18.75 %
ex6	78	**360**	380	-5.56 %
ex7	22	94	**76**	19.15 %
kirkman	186	**684**	706	-3.22 %
mark1	80	492	**378**	23.17 %
mc	23	139	**110**	20.86 %
opus	77	343	**291**	15.16 %
planet	213	2991	**2231**	25.41 %
pma	237	697	**665**	4.59 %
s27	36	136	**122**	10.29 %
sand	476	1596	**1537**	3.70 %
scf	763	3143	**2314**	26.38 %
sse	120	521	**472**	9.40 %
styr	183	675	**614**	9.04 %

Figure 6. Experimental Results on the MCNC Sequential Benchmarks.

The table of figure 6 illustrates the results of the proposed method and TMR on sequential circuits. The first column provides the literal count of the original circuit. The second and third columns summarize the literal count for the TMR and the proposed method. As seen from the last column, the proposed method provides an average saving of 16.23% for cases where the proposed method performs better than a TMR implementation and savings in excess of 25% for some combinational and sequential circuits. The results of the proposed method exploit the advantages of the CED method described in [6], i.e. the hardware reduction of the proposed method is proportional to the hardware reduction of [6].

5. Conclusions

In conclusion, we presented a method to construct fault tolerant designs for combinational and sequential 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 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.

1063-6722/03 $17.00 © 2003 **IEEE**

IEEE COMPUTER SOCIETY

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 multi-phase 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, N. Saxena, 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.

1063-6722/03 $17.00 © 2003 **IEEE**

IEEE COMPUTER SOCIETY