

# Synthesis of Low Power CED Circuits Based on Parity Codes

Shalini Ghosh<sup>1</sup>, Sugato Basu<sup>2</sup>, and Nur A. Touba<sup>1</sup>

<sup>1</sup>Dept. of Electrical and Computer Engineering,  
University of Texas, Austin, TX 78712  
{shalini,touba}@ece.utexas.edu

<sup>2</sup>Dept. of Computer Sciences,  
University of Texas, Austin, TX 78712  
sugato@cs.utexas.edu

## Abstract

*An automated design procedure is described for synthesizing circuits with low power concurrent error detection. It is based on pre-synthesis selection of a parity-check code followed by structure constrained logic optimization that produces a circuit in which all single point faults are guaranteed to be detected. Two new contributions over previous work include (1) the use of a k-way partitioning algorithm combined with local search to select a parity-check code, and (2) a methodology for minimizing power consumption in the CED circuitry. Results indicate significant reductions in area overhead due to the new code selection procedure as well as the ability to find low power implementations for use in power conscious applications.*

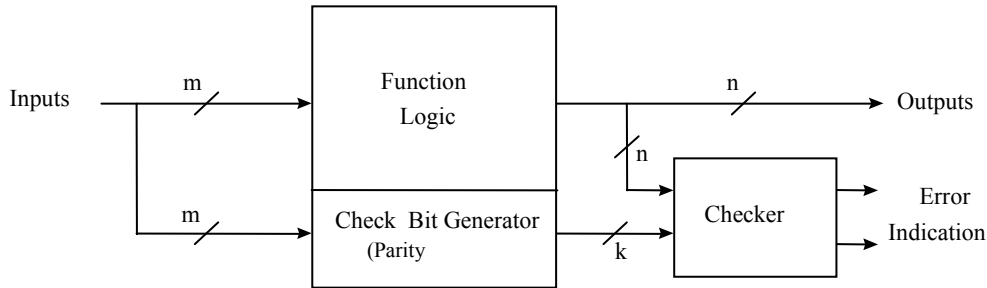
## 1. Introduction

Concurrent error detection (CED) involves detecting errors at the output of a circuit while it operates. As technology continues to scale with smaller features sizes, lower power supply voltages, and higher operating frequencies, the soft error rate in logic circuits is rapidly increasing [Shivakumar 02]. CED provides a means to detect soft errors quickly before they have a chance to propagate and compromise the data integrity of a system. CED is widely used in many applications to improve reliability.

One way to implement CED is to encode the outputs of a circuit with an error detecting code and have a checker that monitors the outputs and gives an error indication if a non-codeword occurs (as illustrated in Fig. 1). The check-bit generator is the parity prediction circuit that calculates the parity bits directly from the circuit inputs. The parity checker is self-checking so that any error that occurs in the checker itself is detected. One commonly used error detecting code is a parity-check code. A parity-check code is a linear code in which each parity check bit checks the parity over a group of output bits. Two special cases of a parity-check code are single-bit parity where there is a single parity bit checking all the outputs of the functional logic and duplication where there is a parity bit checking each output of the functional logic.

A number of techniques have been proposed for automated design of circuits with CED based on parity-check codes. There are two basic approaches. One is to first synthesize the functional logic and then select the parity-check code (post-synthesis code selection). The other is to select the parity-check code and then synthesize the functional logic with structural constraints to ensure high coverage (pre-synthesis code selection). For post-synthesis methods, the goal is to select a parity-check code that provides high coverage while minimizing the complexity of the parity prediction logic (i.e., check bit generator). Since the functional logic circuit is known up front, the code can be selected so that it detects all the output error combinations that can arise due to a specified fault class [Sogomonjan 93], [Goessel 93]. Recent work in [Almukhaizim 04] has investigated the use of fast entropy estimation techniques to find parity-check codes with less complex parity prediction logic. If 100% coverage is not necessary, then the complexity of the parity prediction logic can be reduced. This can be accomplished by using a self-dual complement [Saposhnikov 96] or disabling the parity check for some input combinations [Mohanram 03].

For pre-synthesis methods, the selection of the parity-check code places constraints on the structure of the functional logic. The goal is to find a parity-check code that minimizes the overall area considering the functional logic, check-bit generator, and parity checker. Techniques in [De 94] and [Touba 97] have been proposed to constrain the structure of the functional logic so that a simpler parity-check code can be utilized to provide 100% coverage of single-point faults. By trading off structural constraints on the functional logic (which may result in it being larger) to get a simpler parity-check code (which reduces the size of the parity prediction logic), the overall size of the circuit with CED can be reduced. The key issue here is how to best select the parity-check code that optimizes this tradeoff. The approach in [Touba 97] uses a simply greedy algorithm that starts with the duplication code and iteratively merges parity groups as long as the merging causes the overall area to decrease. The number of parity groups is determined automatically in this greedy algorithm by the convergence criterion of merging until no more area reduction is possible. The drawback of this approach is



**Figure 1.** Concurrent error detection using parity-check code

that it can easily get caught in local minima since no look-ahead information is used during each merge.

While previous work in automated design of circuits with CED based on parity-check codes has focused on minimizing area, this paper investigates minimizing power dissipation. In many low power applications, including hand-held devices, mobile computing, laptop computers, etc., minimizing power is a first-order design issue. This paper presents a new approach for selecting a parity-check code that provides the best tradeoff between structural constraints on the functional logic and complexity of the parity prediction logic to reduce the overall power of the circuit with CED. The problem is reformulated as a  $k$ -way partitioning problem that overcomes the limitations of earlier approaches to identify more optimal parity-check codes. Experiments on benchmark circuits demonstrate that the proposed approach is able to reduce the power dissipation of the CED circuit as well as its area compared to previous techniques.

The paper is organized as follows: Sec. 2 gives an overview of the overall scheme, Sec. 3 gives details of the proposed 2-phase algorithm, Sec. 4 outlines the results of experiments on benchmark circuits, and finally Sec. 5 concludes the paper.

## 2. Overview of Proposed Technique

In the proposed technique, a parity-check code is used to detect all single point faults in the circuit. To ensure this, one has to be careful about logic sharing while synthesizing the logic circuit so that single point faults in the circuit cannot cause errors that get masked and go undetected. We use the structure constrained logic synthesis algorithm of [Touba 97] for synthesizing the logic circuit. The basic idea of this method is that two outputs assigned to the same parity group should not share any logic, because that may allow a single error in the shared logic block to be propagated to both the outputs, resulting in a two-bit error that would not be detected by the parity code. The structure constrained logic optimization algorithm essentially enforces constraints on logic sharing that are necessary to ensure

that no undetectable errors are caused by single point faults.

The power consumption in each of the CED circuit components depends on the type of parity check code used for concurrent error detection. If the duplication code is used (i.e., each output is put in its own parity group), then there are  $n$  parity bits for  $n$  logic outputs. In that case, the parity prediction circuit is a duplicate of the original logic circuit and can have significant power overhead. However, since there are no logic-sharing constraints between outputs in each parity group, the structure constrained logic optimization can share a lot of logic to reduce the power dissipation in the function logic. On the other hand, if all the outputs were put into one parity group, then the parity prediction logic would in general be simpler and have less power dissipation. But then the structure constrained logic optimization algorithm would not be able to share a lot of logic when synthesizing the function logic, which could result in increased power dissipation in the function logic. Thus, we see that there is a tradeoff between the power dissipation of the parity prediction circuit and the functional logic circuit, for which the proper choice of the parity check code is essential to minimize the overall power of the circuit with CED.

The problem of finding an optimal parity check code is equivalent to finding the optimal grouping of the outputs of the function logic such that the power of the circuit with CED is minimized. This suggests a formulation of the problem as a  $k$ -way partitioning of the logic outputs into  $k$  groups so that the power reduction from merging the outputs in each group is maximized.

## 3. Proposed Algorithm

In this work, we propose a 2-phase non-greedy algorithm of  $k$ -partitioning and local search. This avoids the problem of a greedy algorithm getting stuck in bad local optima. The algorithm uses a power-based cost function that estimates the power reduction due to merging two parity groups at a time.

### 3.1. Power-based Cost Function

Merging two parity groups affects the power dissipation in each component of the CED circuit. The reduction in power of the checker and parity prediction circuit can be calculated by the difference in the power dissipated before and after the merging. However, merging of the parity groups also reduces the amount of logic sharing in the functional logic during structure-constrained logic synthesis, which results in more power dissipation. The overall power cost function is the difference of these two components:

*Effective Power Reduction = (Power reduction in checker circuit and parity prediction circuit) – (Power increase from decreased logic-sharing in logic circuit)*

During logic synthesis, the circuit is represented by a Boolean network [Sentovich 92] in which each node represents a two-level logic function and an edge exists from node  $A$  to node  $B$  if node  $A$  is a fanin to node  $B$ . To estimate the effective power reduction, we consider the Boolean network of the parity prediction circuit for the parity code before merging. The power dissipated in each node of the Boolean network is estimated by decomposing the node into 2-input gates, assuming equiprobable input values to the primary inputs of the Boolean network, and calculating the resulting switching probability at each gate output weighted by its capacitive load. We merge the two parity groups under consideration by removing the two corresponding nodes from the Boolean network for the parity prediction circuit and adding a new node obtained by taking an XOR of the logic functions of those nodes. The difference in load-weighted switching probabilities between the new node and the two original nodes gives us an estimate of the power reduction due to merging.

To estimate the second component of the power cost function, we calculate the total power dissipated in the function logic nodes that are reachable from the outputs of both the parity groups. This gives an estimate of the power increase due to decreased sharing in the logic circuit, since the nodes that are reachable from both the parity groups cannot be shared when the functional logic is synthesized with constraints. Since that logic cannot be shared, the total power consumption will increase by that amount.

The difference between the first and the second components gives us the power cost function, i.e., the effective power reduction due to merging two parity

groups. For larger circuits, we can use sampling instead of exact power estimation for computational efficiency.

Note that the actual power dissipation in the parity prediction circuit depends on which cell-library is used to map the logic gates to cells. As an approximation, the proposed procedure decomposes the parity prediction circuit into 2-input gates when estimating power. We also did not consider changing the ordering of the operations in the parity tree to reduce the power consumed in this circuit component – schemes such as the one described in [Mohanram 02] could be used to reorder the parity operations and further reduce power dissipation.

### 3.2. $k$ -partitioning and Local Search

The proposed algorithm partitions the logic outputs into parity groups corresponding to the lowest-power parity check code. It finds the best number of parity groups  $k$  by searching all values of  $k$  between 1 and  $n$ , where  $n$  is the total number of logic outputs. For each value of  $k$ , a 2-phase technique is used for partitioning the logic outputs into parity groups. The details of the algorithm are given in Fig. 2.

In the first phase, a fast  $k$ -partitioning technique is used to create  $k$  good initial groups. Given a particular value of  $k$ , the algorithm chooses  $k$  outputs that have the minimum power reduction due to mutual pairwise merging and initializes the  $k$  partitions with these outputs. These initial outputs are found using the *farthest-first* algorithm [Hochbaum 85], where the basic idea is to get  $k$  points out of  $n$  that are mutually “far” from each other. In farthest first traversal, an initial point is chosen at random. The next point is selected to be farthest from it using a particular distance measure and is added to the traversed set. The remaining points are selected to have maximum distance from the traversed set, where we use the standard notion of distance of a point  $x$  from a set  $S$ :  $d(x,S) = \min_{z \in S} d(x,z)$ . In this case, the distance between two logic outputs  $a$  and  $b$  was set to the inverse power reduction  $1/\text{power\_reduction}(a,b)$  in merging the outputs, and farthest first traversal was performed on the logic outputs using this distance measure. So, when we find the  $k$  initial outputs by farthest first traversal, they are put in different partitions since there will not be a substantial power reduction by merging any two of these outputs. The other outputs are then sequentially merged with their “nearest” partition using a *nearest-neighbor* assignment scheme, which effectively gives the maximum power reduction for each assignment of an output to a group.

In the second phase, these initial partitions are further refined by using a *local-search* refinement technique. Local search is a method of perturbing a solution so as to help it go out of potential local minima. We use a variant of local search where a series of local refinements are performed by considering each output in turn. The local refinement considers removing each output from its current partition and placing it in a new partition, and the resulting power reduction is calculated. For a given output, it finds the movement that gives the maximum power reduction, and the output is removed from the old partition and merged with the new partition. Every output is considered in turn for this local refinement step, and the process is continued iteratively until no more movements give further power reduction. In the end, the outputs in every partition form a parity group.

Since both these phases are computationally efficient, they can be applied to do a non-greedy search over all possible values of  $k$  from 1 to  $n$ , thereby exploring the exponential-size space of all possible parity groupings more effectively. The value of  $k$  that gives the best overall reduction in power and area is used as the number of parity groups, and the corresponding  $k$ -partitioning is used to synthesize the parity prediction and function logic circuits.

#### 4. Experimental Results

For our experiments, the proposed algorithm was run on combinational circuits from the *MCNC* benchmark suite. The  $k$ -way partitioning, local search, and structure-constrained logic optimization algorithms were implemented in *SIS-1.2* [Sentovich 92]. The internal BDD power simulator in *SIS-1.2* was used to do power simulation using a zero-delay model. Power was measured by the weighted switching probabilities of all the nodes of the circuit with CED, where the weight for the switching probability of a node corresponds to the capacitive load of that node. We used two area estimates – the total number of factored form literal counts and the cell area of the circuit with CED after structure-constrained logic synthesis. The mapping was performed using 2-input cells from the *mcnc.genlib* library and the cell area numbers are given in units of  $1000\lambda^2$ , where  $\lambda$  is the minimum size in the technology.

In the first experiment, we used the 2-phase algorithm for parity code selection with the proposed cost function that considers power reduction. Table 1 shows that our proposed 2-phase scheme of  $k$ -partitioning and local search refinement reduces power by as much as 34% on the benchmark circuits. As seen in Table 2, running the 2-phase algorithm with the power cost function also reduces area for all the benchmark circuits by as much as 35% as both area and power are correlated to some degree. The results in Table 1 also show that for some of the benchmark circuits the number of parity groups selected

**Input:** Logic circuit with  $n$  outputs.

**Output:** Number of parity groups and parity grouping of the logic outputs corresponding to the optimal-power parity check code.

**Algorithm:**

1. Initialize
  - $best\_power = 0$
  - $best\_num\_groups = 0$
  - $best\_grouping = NIL$
2. For  $k = 1$  to  $n$
3. Initialize  $k$  partitions using *farthest\_first\_init(k)*
4. Assign outputs to the initial partitions using *nearest\_neighbor\_assign(k)*
5. Refine the partitioning obtained using *local\_search\_refine(k)*, store resulting partitioning in *current\_grouping*
6. If total power reduction *power\_reduced* in steps 3 and 4 is more than *best\_power*, update
  - $best\_power = power\_reduced$
  - $best\_num\_groups = k$
  - $best\_grouping = current\_grouping$
7. Return *best\_num\_groups* and *best\_grouping*

**Subroutines:**

*farthest\_first\_init(k)*

1. Initialize  $k$  partitions with  $k$  logic outputs chosen using the *farthest-first* heuristic, using  $1/power\_reduced(a,b)$  as measure of distance between  $a$  and  $b$

*nearest\_neighbor\_assign(k)*

1. for  $i = 1$  to  $n$
2. If output  $i$  is not already assigned to one of the  $k$  initial partitions, then assign output  $i$  to the partition that is nearest to it, i.e., has the maximum power reduction on merging

*local\_search\_refine(k)*

1. for  $i = 1$  to  $n$
2. Initialize  $best\_reduction = 0$
3. for  $j = 1$  to  $k$
4.  $current\_reduction =$  power reduced by moving output  $i$  from its current partition to the new partition  $j$
5. If  $current\_reduction > best\_reduction$ 
  - $best\_reduction = current\_reduction$
  - $best\_new\_partition = j$
6. Move output  $i$  to partition  $best\_new\_partition$
7. Return if  $best\_reduction = 0$  for all outputs, else repeat steps 1-6

**Figure 2.** 2-phase algorithm for parity code selection

by our algorithm is different from that chosen by the greedy scheme in [Touba 97], demonstrating that the 2-phase algorithm is able to reach a better parity check code by a more effective search of the space of all possible parity groups.

We performed another experiment to explore in detail the effectiveness of each phase in the proposed 2-phase algorithm. As shown by the results in Table 3, the first

phase (i.e.,  $k$ -way partitioning only) alone gave some reductions in power, but using the 2-phase algorithm (i.e.,  $k$ -way partitioning and local search) increased the amount of power reduction. This demonstrates that each of the 2 phases in the algorithm plays a significant role in selecting a low-power parity code.

**Table 1.** Comparison of power reduction of proposed 2-phase algorithm with greedy algorithm in [Touba 97]

Circuit Information			Greedy Algorithm [Touba 97]		Proposed Algorithm		Comparison
Name	PIs	POs	Number of Parity Groups	Power Dissipated	Number of Parity Groups	Power Dissipated	Reduction in Power (%)
misex1	8	7	3	463	3	305	34.1
wim	4	7	2	211	2	159	24.6
rd53	5	3	1	247	1	171	30.6
squar5	5	8	3	513	1	360	29.8
dcl	4	7	3	396	2	285	28.0
adr2	4	3	2	193	2	140	27.7
b12	15	9	4	589	2	521	11.6
rd73	7	3	1	716	1	518	27.7
misex2	25	18	3	624	4	516	17.4
bw	5	28	9	1375	2	1086	20.9
alu2	10	6	5	1527	4	1482	3.0
inc	7	9	2	740	2	573	22.5
5xp1	7	10	3	921	1	827	10.2

**Table 2.** Comparison of area reduction of proposed 2-phase algorithm with greedy algorithm in [Touba 97]

Circuit Information			Greedy Algorithm [Touba 97]		Proposed Algorithm		Comparison	
Name	PIs	POs	Number of Literals	Cell Area	Number of Literals	Cell Area	Reduction in Literals (%)	Reduction in Cell Area (%)
misex1	8	7	138	156	91	105	30.1	32.7
wim	4	7	69	82	50	57	23.2	30.5
rd53	5	3	55	73	35	47	27.4	35.6
squar5	5	8	139	174	93	120	26.4	31.0
dcl	4	7	101	117	65	78	30.8	33.3
adr2	4	3	48	56	30	37	32.1	33.9
b12	15	9	174	209	153	159	10.0	23.9
rd73	7	3	158	196	121	135	18.9	31.1
misex2	25	18	296	355	255	319	11.5	10.1
bw	5	28	442	548	362	426	14.6	22.3
alu2	10	6	417	512	409	469	1.6	8.4
inc	7	9	215	253	169	197	18.2	22.1
5xp1	7	10	247	289	211	240	12.5	17.0

**Table 3.** Breakdown on power reduction for each phase of proposed 2-phase algorithm

Circuit Information			Greedy Algorithm [Touba 97]	After phase 1	After phase 2	Comparison	
Name	PIs	POs	Power Dissipated	Power Dissipated	Power Dissipated	Power Reduction after phase 1 (%)	Power Reduction after phase 2 (%)
misex1	8	7	463	326	305	29.6	34.1
wim	4	7	211	166	159	21.3	24.6
rd53	5	3	247	187	171	24.2	30.6
squar5	5	8	513	399	360	22.1	29.8
dc1	4	7	396	298	285	24.9	28.0
adr2	4	3	193	140	140	27.7	27.7
b12	15	9	589	551	521	6.5	11.6
rd73	7	3	716	564	518	21.2	27.7
misex2	25	18	624	546	516	12.5	17.4
bw	5	28	1375	1108	1086	19.4	20.9
alu2	10	6	1527	1489	1482	2.4	3.0
inc	7	9	740	577	573	22.1	22.5
5xp1	7	10	921	882	827	4.3	10.2

## 5. Conclusions

This paper presents a new 2-phase algorithm for synthesis of low-power concurrent error-detecting circuits based on parity codes, which outperforms a previously known parity-code selection technique. Along with reducing the power of benchmark circuits with CED by as much as 34%, the 2-phase algorithm is also able to simultaneously reduce their area by as much as 35%.

## Acknowledgements

The authors would like to thank Subhasish Mitra, Sandip Ray and Arindam Banerjee for helpful discussions and suggestions. This research was supported in part by the National Science Foundation under Grant No. CCF-0426608.

## References

- [Almukhaizim 04] Almukhaizim, S., P. Drineas, and Y. Makris, "Cost-Driven Selection of Parity Trees", *Proc. of the IEEE VLSI Test Symposium (VTS)*, pp. 319-324, 2004.
- [Bolchini 97] Bolchini, C., F. Salice, and D. Sciuto, "A novel methodology for designing TSC networks based on the parity bit code", *Proc. of the European Design and Test Conference (ED&TC-97)*, pp. 440 – 444, 1997.
- [De 94] De, K., C. Natarajan, D. Nair, and P. Banerjee, "RSYN: A system for automated synthesis of reliable multilevel circuits," *IEEE Trans. on VLSI Systems*, vol. 2, pp. 186 – 195, June 1994.
- [Goessel 93] Goessel, M., and S. Graf, *Error Detection Circuits*, London, NY: McGraw-Hill, 1993.
- [Hochbaum 85] Hochbaum, D., and D. Shmoys, "A best possible heuristic for the  $k$ -center problem", *Mathematics of Operations Research*, Vol. 10, No. 2, pp. 180 – 184, 1985.
- [Mohanram 02] Mohanram, K., and N.A. Touba, "Input Ordering in Concurrent Checkers to Reduce Power Consumption", *Proc. of IEEE Symposium on Defect and Fault Tolerance*, pp. 87 – 95, 2002.
- [Mohanram 03] Mohanram, K., and N.A. Touba, "Partial Error Masking to Reduce Soft Error Failure Rate in Logic Circuits", *Proc. of IEEE Symposium on Defect and Fault Tolerance*, pp. 433 – 440, 2003.
- [Saposhnikov 96] Saposhnikov, V.I., A. Dmitriev, M. Goessel, V.V. Saposhnikov, "Self-dual parity checking - a new method for on-line testing", *Proc. of the IEEE VLSI Test Symposium (VTS)*, pp. 162 – 168, 1996.
- [Sentovich 92] Sentovich, E.M., et al., SIS: A System for Sequential Circuit Synthesis," TRM No. UCB/ERL M92/41, University of California, Berkeley, 1992.
- [Shivakumar 02] Shivakumar, P., M. Kistler, S.W. Keckler, D. Burger, and L. Alvisi, "Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic", *Proc. of the International Conference on Dependable Systems and Networks*, pp. 389 – 398, 2002.
- [Sogomonjan 93] Sogomonjan, E.S., and M. Goessel, "Design of self-parity combinational circuits for self-testing and on-line detection", *Proc. of the IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, pp. 239 – 246, 1993.
- [Touba 97] Touba, N.A., and E.J. McCluskey, "Logic Synthesis of Multilevel Circuits with Concurrent Error Detection", *IEEE Trans. on Computer-Aided Design*, Vol. 16, No. 7, pp. 783 – 789, 1997.