Cost-Driven Selection of Parity Trees

Sobeeh Almukhaizim *
Electrical Engineering Dept.
Yale University
New Haven, CT 06520, USA

Petros Drineas
Computer Science Dept.
Rensselaer Polytechnic Institute
Troy, NY 12180, USA

Yiorgos Makris
Electrical Engineering Dept.
Yale University
New Haven, CT 06520, USA

Abstract

We discuss the problem of parity tree selection for lossless compaction of the output responses of a circuit. Earlier methods assume off-chip storage of the correct compacted responses and therefore minimize the number of necessary parity trees. In contrast, our method targets on-chip generation of the correct compacted responses and therefore minimizes the actual implementation cost of the corresponding parity prediction functions. We present a systematic search approach that exploits the correlation between the hardware cost of a function and its **entropy***, in order to select parity trees that minimize the incurred cost, while achieving lossless compaction. Experimental results demonstrate that our method achieves significant hardware reduction over methods that minimize the number of parity trees.*

1. Introduction

Parity trees are used extensively in test-related applications. For example, test response compaction methods [1, 2] employ parity trees in order to reduce the amount of test data. Since the correct compacted responses are explicitly stored off-chip, the optimization objective of such methods is the minimization of the number of parity trees and, by extension, the corresponding storage requirement. Similarly, Concurrent Error Detection (CED) methods [3, 4, 5] exploit the expected parity of one or several groups of output bits. In this case, however, the correct compacted responses are generated on-chip. Therefore, the optimization objective of such methods should be the minimization of the incurred hardware cost for predicting the correct compacted responses. Interestingly, selecting the minimum number of parity trees does not always result in the minimal parity predictor implementation. While the former problem has been studied extensively, little is known for the latter.

In this paper, we present a systematic method for selecting parity trees based on the cost of the corresponding parity prediction circuit. Based on the well-known correlation between the entropy of a function and its cost [6], our method uses an entropic metric to identify inexpensive solutions. We present an algorithm that searches for feasible solutions through Integer Linear Programming and Randomized Rounding, estimates their entropy through Monte Carlo sampling and approximates the optimal entropic potential function through a Metropolis random walk.

*The author is supported through a scholarship from Kuwait University.

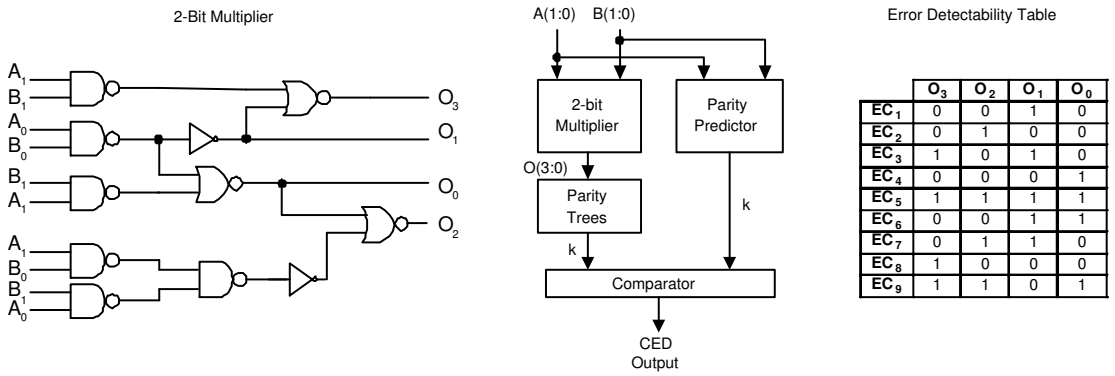
2. Motivation

Consider the gate-level implementation of a 2-bit Multiplier shown in Figure (1)(a) and assume that we want to perform CED using the method of Figure (1)(b). In this method, the outputs of the multiplier are compacted through k *Parity Trees*, and compared to the correct parity bits that are generated through an additional *Parity Predictor* [5]. In order to construct k parity trees that achieve lossless compaction, an *Error Detectability Table* such as the one shown in Figure (1)(c) is required. Columns on this table represent output bits and rows represent *Erroneous Cases (ECs)*. The (i, j) element of the table is "1" if and only if Erroneous Case i can be detected at output bit j.

Given a target error model, the set of Erroneous Cases of interest may be obtained by fault simulation of the errors in this model. For the purpose of this example, we fault-simulated all single stuck-at faults in the circuit for all possible input combinations and we obtained the 9 ECs shown in the table of Figure (1)(c). Essentially, this means that any error that behaves as a single stuck-at fault will result in a discrepancy between the good machine and the bad machine response in one of these 9 sets of output bits.

The problem now reduces to choosing p parity trees to compact the outputs, while preserving the detectability of all ECs. In this example, there are 15 possible parity trees, one for each element in the powerset of the output bits. An EC is detected by a parity tree if and only if it is detectable at an *odd* number of output bits included in the parity tree. Previous work [1, 2] has focused on selecting the *minimum number* of parity trees such that all ECs are detected. The problem has been shown to be NP-complete. An easy way to prove this is to expand the Error Detectability table such that a column is included for every possible parity tree, as shown in Figure (1)(d). In this case selection of the minimum number of parity trees becomes a MIN-COVER[7] problem. Explicitly building this expanded table, however, is both infeasible due to the exponential explosion and unnecessary, since all information regarding the detection of ECs by parity trees can be inferred from the basic table. Therefore, existing heuristics solve the problem without constructing the expanded table. For example, an Integer Program formulation of the problem was recently presented and a solution based on Linear Program Relaxation and Randomized Rounding was proposed [5]. On the multiplier, these methods yield the minimum number of parity trees that detect all ECs, $k = 2$. One such solution is $P_1 = O_1 \oplus O_0$ and $P_2 = O_3 \oplus O_2 \oplus O_1$.

Proceedings of the 22nd IEEE VLSI Test Symposium (VTS 2004)
1093-0167/04 $20.00 © 2004 IEEE



(a) (b)

	O_3	O_2	O_1	O_0
EC_1	0	0	1	0
EC_2	0	1	0	0
EC_3	1	0	1	0
EC_4	0	0	0	1
EC_5	1	1	1	1
EC_6	0	0	1	1
EC_7	0	1	1	0
EC_8	1	0	0	0
EC_9	1	1	0	1

(c)

	O_3	O_2	O_1	O_0	XOR O_3, O_2	XOR O_3, O_1	XOR O_3, O_0	XOR O_2, O_1	XOR O_2, O_0	XOR O_1, O_0	XOR O_3, O_2, O_1	XOR O_3, O_2, O_0	XOR O_3, O_1, O_0	XOR O_2, O_1, O_0	XOR O_3, O_2, O_1, O_0
EC_1	0	0	1	0	0	1	0	1	0	1	1	0	1	1	1
EC_2	0	1	0	0	1	0	0	1	1	0	1	1	0	1	1
EC_3	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0
EC_4	0	0	0	1	0	0	1	0	1	1	0	1	1	1	1
EC_5	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0
EC_6	0	0	1	1	0	1	1	1	1	0	1	1	0	0	0
EC_7	0	1	1	0	1	1	0	0	1	1	0	1	1	0	0
EC_8	1	0	0	0	1	1	1	0	0	0	1	1	1	0	1
EC_9	1	1	0	1	0	1	0	1	0	1	0	1	0	0	1

(d)

Figure 1. Motivation Example: Parity-Based CED on a 2-Bit Multiplier

This optimization objective is appropriate for methods that *store* the correct compacted responses off-chip [1, 2], since it minimizes the necessary storage. However, it is not the most appropriate optimization objective for methods that *generate* the correct compacted responses using on-chip hardware [5]. In the latter, parity tree selection should rather be driven by the *actual hardware cost* required to predict the parity tree outputs. When we synthesized the parity predictor for the two parity functions P_1 and P_2 chosen above, its cost was 22 gates. Surprisingly, these two functions are even more expensive than the circuit itself, which generates four functions and costs 11 gates. Yet other solutions exist that yield a cheaper parity prediction circuit. For example, another set of two parity trees, $P_3 = O_3 \oplus O_1$ and $P_4 = O_3 \oplus O_2 \oplus O_0$, will also detect all ECs and the corresponding parity predictor only costs 10 gates. Moreover, there exists a solution with $k = 3$ parity trees, $P_3 = O_3 \oplus O_1$, $P_5 = O_0$, and $P_6 = O_3 \oplus O_2$, which also detects all ECs and has a parity predictor that only costs 7 gates.

As demonstrated through this toy example, output functions in practical circuits are typically structured and simple, resulting in inexpensive implementations. Yet when XOR-ed together they may result in complicated random functions that are expensive to implement. Therefore, minimizing the number of parity trees is not the best optimization objective for methods that generate the compacted responses through on-chip hardware. Rather, methodologies that select parity trees for lossless compaction based on the actual cost of the parity predictor are required.

3. Cost-Driven Parity Tree Selection

In this section, we provide a brief review of related research efforts and we introduce the proposed methodology.

3.1. Related Work

Previous research efforts have also pointed out the need to reduce the implementation cost of the parity prediction functions, especially within the area of CED. In [8], the authors propose replacing the parity function with a simpler function, the self-dual complement, that requires lower area overhead. More recently, a technique is proposed to disable CED for a pre-selected subset of the inputs of the circuit [9]. Masking the CED output introduces don't care conditions in the parity functions that allows further logic optimization of the parity predictor. Both of these techniques are lossy, i.e. they reduce the cost of the parity predictor at the expense of sacrificing fault coverage. Furthermore, they are applied *after* the parity trees are selected. In contrast, the method proposed herein considers the implementation cost of the corresponding parity predictor *during* the selection of parity trees. At the same time, it guarantees lossless compaction, which preserves fault coverage.

The only previous work that we are aware of, wherein parity tree selection is driven by the actual hardware cost of the implementation is [4]. In this work, the authors resynthesize the circuit to include CED based on multiple parity groups. They devise a cost function which reflects the total cost of the

modified original circuit and the corresponding parity prediction circuit. Subsequently, they guide the formation of parity groups towards minimization of this cost function through a greedy heuristic. The cost function is exact, since it is obtained through actual synthesis of potential solutions. Explicit synthesis of candidate solutions, however, is computationally expensive, thus limiting the search space that can be explored in practice. In contrast, the method proposed herein employs a statistical, yet highly accurate, cost function that can be computed rapidly, allowing exploration of a much larger search space. Furthermore, it is a post-synthesis method which does not modify the original circuit.

3.2. Proposed Method Overview

We emphasize the high complexity of the problem at hand, which arises from the two-fold objective of achieving lossless compaction while minimizing the hardware of the predictor. Although lossless compaction can be expressed as a linear optimization problem [5], hardware minimization leads to non-linearities that are much harder to handle. Therefore, in order to achieve our goal, we devise a combination of several advanced algorithmic methods. These include *randomized rounding* to approximately solve Integer Linear Programs, *Monte Carlo sampling techniques* to estimate the entropy of a function and, by extension, its cost, as well as the *Markov Chain Monte Carlo method* and the *Metropolis random walk* to approximately optimize an entropic potential function.

Before detailing the proposed algorithm in the following section, we briefly sketch our approach. Given a circuit and its Error Detectability Table, we start by finding **a specific feasible solution**, namely, a set of k parity trees for compacting the circuit outputs such that the detectability of all Erroneous Cases is preserved. Notice that the output of the k parity trees is a k-bit 0-1 string. Then, we **estimate the hardware cost of the predictor** associated with this specific set of parity trees by **approximating its entropy**. Again, notice that the predictor is a function whose domain is the set of all possible inputs to the circuit and the output is a k-bit string. We then examine a **neighboring feasible solution**, namely, a solution that preserves detection of all Erroneous Cases and is "close" to the previous solution, in some well-defined "proximity" metric. Subsequently, we estimate the cost of the new solution, once again, by estimating its entropy. We then **accept or reject** a move to this neighbor with a probability that depends on the estimated cost of the new solution. More specifically, "better" solutions are almost always accepted, while "worse" solutions are accepted with a probability that becomes exponentially smaller, as their quality decreases. This is essentially the classical Metropolis algorithm, which is the basis of Simulated Annealing techniques that have been very successful in tackling non-linear optimization problems in various contexts.

4. Proposed Algorithm

The proposed algorithm, which we describe at a high level in this section, is based on the following principles:

- Low entropy corresponds to non-random functions, and thus to cheaper hardware implementations.
- Estimating the entropy is feasible via uniform sampling.
- A *biased* random walk in the set of feasible solutions, using entropy as the potential function and a bias towards entropy minimization, converges to optimal feasible solutions.

The building blocks of our approach are discussed below: We first formulate the problem of lossless compaction using a fixed number of parity trees as an *Integer Linear Program (ILP)* and we approximate it efficiently in polynomial time via *randomized rounding*, as explained in section 4.1. We then demonstrate how the hardware complexity of the parity predictor may be efficiently estimated by *approximating its entropy*, as detailed in section 4.2. Subsequently, in section 4.3 we discuss the use of a *Metropolis random walk* in order to select a parity predictor with minimum entropy. In section 4.4 we present the overall algorithm, which repeats the above procedure for all possible values of k from 1 up to n and selects the best solution. Finally, open issues and opportunities for further improvement of our algorithm are discussed in section 4.5.

4.1. Lossless Compaction via ILP

We demonstrate how to model parity tree selection for lossless compaction as a set of integral inequalities; we then use *randomized rounding* to identify *feasible solutions* - namely solutions satisfying the constraints. This formulation of the problem is described in detail in [5]. For the purpose of completeness, we briefly summarize its key points.

Assume that the circuit has m input bits $\{I_1 \dots I_m\}$ and n output bits $\{O_1 \dots O_n\}$. We are given a set of f erroneous cases $\mathcal{F} = \{EC_1 \dots EC_f\}$ and an $f \times n$ matrix V such that $V_{ij} = 1$ if and only if erroneous case EC_i is detected by the j-th output bit O_j; otherwise, $V_{ij} = 0$. This matrix is the Error Detectability Table that was described in the example of Figure (1)(c). Achieving lossless compaction via k parity trees is equivalent to the following statement:

Statement 1 *Given a positive integer k, find k n-dimensional binary vectors $\beta^{(1)}, \dots, \beta^{(k)}$ such that*

$$\sum_{i=1}^{k} \left(V \cdot \beta^{(i)} \bmod 2 \right) \quad \geq \quad \vec{\mathbf{1}}_{\mathbf{n}}$$

or report the lack thereof ($\vec{1}_n$ denotes an n-dimensional vector of ones).

Notice that the k vectors $\beta^{(i)}$ are n-dimensional 0-1 vectors; each of these vectors corresponds to a parity tree that includes the outputs of the circuit that are set to 1 in the vector. For example, if $\beta^{(1)}$ is [1 0 0 1 0 1], then the corresponding parity tree is $O_6 \oplus O_3 \oplus O_1$. Our goal is to find a *feasible solution*; namely, values for all $\beta^{(i)}$ such that statement 1 is satisfied. The above statement is equivalent to an Integer Linear Program, since we can remove the modulo operator at the expense of extra variables [5]. Identifying a feasible point for an integer program is NP-complete; we employ a technique called *randomized rounding* [10] to solve it. The idea of randomized rounding is simple: solve the *linear programming relaxation* of the integer program (which may be easily done in polynomial time using, for example, the Simplex algorithm) and round the resulting *real* values *probabilistically*, thus forcing them to integers. We note that the linear programming relaxation amounts to removing the integrality constraints and replacing them by continuous intervals. As proved in [10], if a feasible point for the integer program of statement 1 exists, randomized rounding identifies it with very high probability.

4.2. Hardware Estimation via Entropy

Recall from the example of Figure (1) that we are interested in minimizing the cost of the hardware implementation of the parity predictor. Thus, in order to avoid expensive feasible solutions during selection of parity trees for lossless compaction, we need to be able to *estimate* the implementation cost of the corresponding parity predictor functions efficiently. We use the *entropy* of the parity predictor function as our estimator. Cheng and Agrawal [6] pioneered the use of *entropy* for estimating the complexity of a multi-output function. Their observations have been followed by a body of work [11, 12, 13], establishing the correlation between the entropy of a function and the cost of its hardware implementation. Therefore, we refer the reader to these references and we do not elaborate further on the quality of entropy as an estimator for the implementation cost of a function.

We now define the entropy of the parity predictor function. Recall that the domain of this function is $\{0,1\}^m$ (all possible inputs combinations) and its range is $\{0,1\}^k$, where k is the number of parity trees. Let y denote the output of the predictor, which is a k-bit string of zeros and ones. Then, given a set of parity trees $\mathcal{B} = \beta^{(1)} \dots \beta^{(k)}$, the entropy of the corresponding parity predictor function is

$$H(\mathcal{B}) = - \sum_{x \in \{0,1\}^k} \mathbf{Pr}(y = x) \cdot \ln \mathbf{Pr}(y = x)$$

Essentially, the $\mathbf{Pr}(y = x)$ is the number of times that the output of the parity predictor (y) is equal to the specific string x over all possible 2^m inputs of the parity predictor, divided by 2^m. The entropy always ranges between 0 and $log(k)$; smaller values imply that the function is less random and, hence, its implementation cost is expected to be smaller, while values of entropy close to $log(k)$ imply that the function is quite random and, hence, its implementation cost is expected to be high. Notice that exactly computing the entropy takes time exponential in m; fortunately, we can approximate the entropy by *sampling* over all possible 2^m inputs and *estimating* the probabilities $\mathbf{Pr}(y = x)$ in the above formula.

4.3. Cost-Driven Selection via Random Walk

Let Ω denote the set of all possible feasible solutions to the ILP of statement 1; each feasible solution is a set of vectors $\mathcal{B} = \beta^{(1)} \dots \beta^{(k)}$. Let each element of Ω be a vertex of a graph G; we will perform a *biased random walk* on G. However, first we need to define the *edges* of G.

An *edge* in a graph usually denotes some "similarity" between the corresponding vertices. In our case, two vertices (feasible solutions) will be "similar" if at most t elements of their corresponding set of vectors (parity trees) are different, where $t < k \cdot n$ is a small, constant value. Intuitively, this notion of similarity means that the two feasible solutions corresponding to the vertices of G have almost the same parity trees, with at most t changes. Edges on G exist between all "similar" vertices, under the above "similarity" definition. Notice that the number of neighbors of a vertex is exactly equal to $MAX = \sum_{j=1}^{t} \binom{kn}{j}$. We now construct a Markov chain in the form of a biased random walk in G as follows:

1. Find a feasible solution, by solving the ILP of statement 1 via randomized rounding; this solution corresponds to a vertex v of G.
2. Find the set of all the neighbors $\Gamma(v)$ of v; this amounts to constructing all MAX neighbors and testing whether they correspond to feasible solutions. Let $d(v) = |\Gamma(v)|$ denote the *degree* of v.
3. With probability 1/2 stay at v.
4. Otherwise, with probability $1 - d(v)/MAX$ stay at v; otherwise, select one of the $d(v)$ neighbors of v with probability $1/MAX$.
5. Let w denote the selected neighbor; move to w with probability
$$\min\{1, 2^{H(\mathcal{B}) - H(\tilde{\mathcal{B}})}\},$$
where $\mathcal{B}$ is the set of parity trees corresponding to the feasible solution of vertex v and $\tilde{\mathcal{B}}$ is the set of parity trees corresponding to the feasible solution of vertex w. Here, we do not exactly compute the entropy; instead we approximate it by uniform sampling.
6. Set $v = w$, go to step 2 and repeat for $M = 10^3$ steps.

The above random walk essentially corresponds to the Metropolis random walk [14]. The constant 2 in step 5 is rather arbitrary; it could be replaced by any constant $\alpha \geq 1$. Notice that if $H(\mathcal{B})$ is larger than $H(\tilde{\mathcal{B}})$ (the solution at w is better than the solution at v), then the move happens with probability 1; otherwise, the probability of moving drops rapidly. Usually, values of α close to one correspond to random walks that explore a large fraction of Ω (thus, they are slow); on the other hand large values of α (e.g. $\alpha \rightarrow \infty$) correspond to myopic, greedy approaches that are fast, but could get stuck at local optima. As a result, picking a value of α usually emerges from fine tuning the experimental setting.

Finally, the choice of t is certainly a critical issue in our approach. Small values of t (e.g. $t = 1$) result in graphs with a small number of edges, and, usually, with many independent connected components. Obviously, a random walk is unable to explore such a space efficiently. In our experiments with benchmark circuits, we observed that small perturbations of feasible solutions resulted to new feasible solutions; thus, we experimented with small values of t which were chosen adaptively. However, larger, more complex circuits will probably require larger values of t, which will directly impact the complexity of our algorithm.

4.4. Overall Algorithm

Based on the above methods, the overall algorithm is now simple to describe: for every possible value of k (where k is the number of parity trees and ranges from 1 up to n), run the Metropolis random walk on the graph G generated by the set of feasible solutions to the ILP of statement 1. Obtain the best solution for each k, and divide its entropy by $log(k)$ for normalization purposes. At the end, pick the solution with the lowest normalized entropy. Notice that larger values of k might return smaller hardware costs, thus necessitating the execution of the random walk for all possible values of k.

4.5. Open Issues

Our methodology takes an important first step towards tackling the question of simultaneously *achieving lossless compaction* and *minimizing the hardware overhead* of the parity predictor function in non-trivial ways. Yet, several important and difficult to answer questions remain open, presenting opportunities for further research. More specifically:

- Is there a better estimator than entropy for the hardware cost of the parity prediction functions?
- What would be a good choice for t in practical circuits?
- What would be a good choice for the parameters of the Metropolis random walk (α and M)?
- Would it be advantageous to use a time inhomogeneous random walk by varying α over time? (This is the simulated annealing approach described in [15]).

5. Experimental Results

In this section, we compare our cost-driven parity tree selection method to the method that selects the minimum number of parity trees. We implemented both methods and applied them on several MCNC benchmarks. The circuits are synthesized using the *rugged* script of SIS [16] and mapped to a standard library containing only 2-input gates. In order to obtain a set of realistic Erroneous Cases of interest for the experiments, we construct the Error Detectability Table based on the single stuck-at fault model. This is performed through internally developed software that employs fault simulation [17] of all single stuck-at faults for all input combinations and identifies all error-free and erroneous responses. Once the Error Detectability Table is constructed, we perform lossless compaction via ILP to compute the minimum number of parity trees required, k_{min}, as described in [5]. Subsequently, the proposed method is applied and the set of k parity trees that detect all the Erroneous Cases and have the minimum normalized entropy is selected for k ranging between k_{min} and n, as described in section 4.4.

The results are summarized in Table 1. Under the first major heading, we provide details about the circuits that were used: name, number of primary inputs, number of primary outputs, gate count and the hardware cost reported by SIS after synthesis. Under the second major heading, we provide the results of selecting the minimum number of parity trees for lossless compaction, k_{min}, the gate count for the corresponding parity predictor, the hardware cost reported by SIS after synthesis, and the entropy of the selected solution. Under the third major heading, the same information is reported for the proposed cost-driven selection method.

The rightmost column indicates the hardware reduction that cost-driven selection of parity trees achieves over selection of the minimum number of parity trees. The average reduction over all benchmark circuits is 21.85%. On several benchmarks, the proposed method yields a set of $k = k_{min}$ parity trees. Yet the selected solution is picked based on the lowest entropy and therefore results in a less expensive hardware implementation. For example, this is the case for circuits tav, $s27$, $dk16$, $s1$, pma and $s386$. In some cases, such as $s27$ and $s1$, the cost of the solutions is reduced by more than 50%.

Additionally, the proposed method selects a set of k parity trees, with $k > k_{min}$, if the entropy of the k-bit parity predictor is lower than the entropy of k_{min}-bit parity predictor. For example, on circuit $dk512$ the proposed method yields a solution with $k = 6$ parity trees and a normalized entropy of 0.818, while the method targeting the minimum number of parity trees yields a solution with $k_{min} = 4$ parity trees and a normalized entropy of 0.973. As may be observed, selecting the solution with the higher number of parity trees, yet with a lower entropy, results in a less expensive parity prediction circuit. Similar observations hold for circuit $s1a$.

Original Circuit					Minimum # of Parity Trees				Cost-Driven Selection				
Circuit Name	PI	PO	Gates	Cost	# of Trees (k_{min})	Gates	Cost	Entropy	# of Trees (k)	Gates	Cost	Entropy	Savings
cse	11	11	196	256128	5	131	171680	0.641	5	97	127136	0.598	25.95%
donfile	7	6	97	128064	4	57	74704	0.716	4	56	75632	0.692	-1.24%
dk16	7	8	240	316448	6	323	428736	0.953	6	294	389760	0.883	9.09%
dk512	5	7	74	96048	4	79	104400	0.973	6	61	80272	0.818	23.11%
keyb	12	7	228	298352	5	82	107648	0.512	5	67	86304	0.439	19.83%
pma	13	13	347	453792	6	186	243136	0.247	6	138	180496	0.210	25.76%
sse	11	11	131	178640	5	80	104864	0.771	5	59	77488	0.753	26.12%
styr	14	15	413	547056	8	217	287216	0.463	8	162	213904	0.394	25.53%
s1	13	11	167	217616	5	121	156832	0.680	5	61	77024	0.512	50.89%
s1a	13	11	153	199056	6	96	124816	0.533	7	61	77024	0.314	38.23%
s27	7	4	20	25056	3	15	18096	0.952	3	7	8352	0.813	53.85%
s386	13	13	123	158688	4	83	105328	0.420	4	72	92336	0.400	12.35%
tav	6	6	28	34336	4	31	39440	0.723	4	26	33408	0.685	15.22%
tbk	11	8	146	190240	5	160	207872	0.683	5	151	198592	0.647	4.46%
tma	12	11	219	285824	5	130	169824	0.258	7	131	172144	0.224	-1.37%

Table 1. Experimental Results on MCNC Benchmark Circuits

Occasionally, the proposed method will fail to provide hardware reduction over the solution with the minimum number of trees. This is for example the case for circuit tma, where the entropy of the proposed solution with $k = 7$ parity trees is slightly less than the entropy of the solution with $k_{min} = 5$ parity trees, yet the implementation of the latter is slightly less expensive. This may be attributed to the heuristics employed for reducing the search space of the problem. Additionally, the entropy is a statistical and not an exact metric. Significant entropy differences provide a good indication of the relative circuit complexity. However, the comparison resolution may degrade as the absolute value of the difference between the entropy of two circuits becomes smaller.

6. Conclusion

Parity trees that achieve lossless compaction while incurring inexpensive parity prediction functions are necessary for test methods that generate the correct compacted responses on-chip. Towards this end, we presented a method that utilizes the entropy of the prediction circuit as a potential function for guiding our search algorithm and selecting appropriate parity trees. We also discussed the limitations of the proposed methodology and pointed towards directions for further improvement. Experimental data on several benchmark circuits indicated that the proposed methodology yields significant cost reduction for the parity predictor of the selected parity trees, as compared to methods that select the minimum number of parity trees.

References

[1] K. Chakrabarty and J. P. Hayes, "Test response compaction using multiplexed parity trees," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 15, no. 11, pp. 1399–1408, 1996.

[2] O. Sinanoglu and A. Orailoglu, "Space and time compaction schemes for embedded cores," in *International Test Conference*, 2001, pp. 521–529.

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

[4] N. A. Touba and E. J. McCluskey, "Logic synthesis of multilevel circuits with concurrent error detection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 7, pp. 783–789, 1997.

[5] S. Almukhaizim, P. Drineas, and Y. Makris, "On concurrent error detection with bounded latency in FSMs," in *Design Automation and Test in Europe Conference*, 2004.

[6] K-T. Cheng and V. D. Agrawal, "An entropy measure for the complexity of multi-output boolean functions," in *Design Automation Conference*, 1990, pp. 302–305.

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[8] Vl. V. Saposhnikov et al., "Self-dual parity checking - a new method for on-line testing," in *VLSI Test Symposium*, 1996, pp. 162–168.

[9] K. Mohanram et al., "Synthesis of low-cost parity-based partially self-checking circuits," in *International On-Line Test Symposium*, 2003, pp. 35–40.

[10] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 3rd edition, 1995.

[11] S. Rajgopal, "Spatial Entropy - A Unified Attribute to Model Dynamic Communication in VLSI Circuits," Tech. Rep. UIUC TR92-041, 1, 1992.

[12] N. Shanbhag, "A mathematical basis for power-reduction in digital VLSI systems," *IEEE Transactions on Circuits and Systems*, vol. 44, pp. 935–951, 1997.

[13] M. Nemani and F. N. Najm, "Delay Estimation VLSI Circuits from a High-Level View," in *Design Automation Conference*, 1998, pp. 591–594.

[14] N. Metropolis et al., "Equation of state calculation by fast computing machines," *Journal of Chemical Physics*, 1953.

[15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science, Number 4598, 13 May 1983*, pp. 671–680.

[16] 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.

[17] H. K. Lee and D. S. Ha, "HOPE: An efficient parallel fault simulator for synchronous sequential circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1048–1058, 1996.