

Seamless Integration of SER in Rewiring-Based Design Space Exploration

Sobeeh Almkhaizim* & Yiorgos Makris
Electrical Engineering Dept.
Yale University
New Haven, CT 06520, USA

Yu-Shen Yang & Andreas Veneris
Electrical and Computer Engineering Dept.
University of Toronto
Toronto, Ontario M5S 3G4, Canada

Abstract

Rewiring has been used extensively for optimizing the area, the power consumption, the delay, and the testability of a circuit. In this work, we demonstrate how rewiring can also be used for reducing the Soft Error Rate (SER). We employ an ATPG-based rewiring method to generate functionally-equivalent yet structurally-different implementations of a logic circuit based on simple transformation rules. This rewiring capability, along with an off-the-shelf method for assessing the SER of a circuit, enable the integration of the SER in a unified search algorithm that iteratively evolves the design in order to satisfy a given set of objectives. Experimental results on ISCAS'89 and ITC'99 benchmark circuits verify that rewiring can indeed be successfully used to reduce the SER of a circuit and, thus, it facilitates a design-space exploration framework for trading off area, power consumption, delay, testability, and SER.

1 Introduction

Soft errors are emerging as a serious reliability threat to the operation of logic circuits. When high-energy neutrons or alpha particles strike a sensitive region in a semiconductor device, they generate a Single Event Transient (SET) which may alter the state of the system, resulting in a soft error. Whereas soft errors have traditionally been of much greater concern in memories, smaller feature sizes, lower voltage levels, higher operating frequencies, and reduced logic depth are projected to cause a dramatic increase in soft error failure rate in core combinational logic in sub-100nm technologies [1]. Thus, designers are faced with the challenging task of implementing appropriate reliability mechanisms to shield electronic circuits against soft errors.

To this end, various methods have been proposed in the literature [2, 3, 4, 5, 6, 7] to reduce the Soft Error Rate (SER) of a circuit and, thus, improve its reliability. The idea behind most of these methods revolves around developing solutions at the physical level, wherein individual transistor characteristics are perturbed to reduce the sensitivity of logic gates to SETs. While these methods are particularly effective in

reducing the SER of a design, they are technology dependent, i.e. they rely on information available only during or after mapping of a circuit to a target technology. In contrast, in this work we are interested in investigating technology-independent methods, i.e. logic-level methods that select, among the many possible gate-level implementations of a circuit, the one that minimizes its SER. While such logic-level methods cannot benefit from the detailed information available at the physical level, and, thus, may not be able to provide comparable levels of SER reduction, they offer two unique advantages. First, they enable design modifications for SER reduction that are equally effective independent of the technology to which the circuit will be mapped. Second, they provide the ability to consider SER as a design objective much earlier in the design cycle. Moreover, the mechanisms through which soft errors can be averted at the logic level are typically orthogonal to those at the physical level; hence, *logic-level SER reduction methods do not intend to substitute their physical-level counterparts but, rather, to provide a better starting point.* Yet the literature lacks solutions for reducing the SER of a circuit at the logic level.

In this paper, we propose a systematic logic-level SER reduction method through the use of *rewiring*. Rewiring methods have been extensively used for transforming a logic circuit to meet design constraints such as minimizing area [8, 9, 10], reducing power consumption [11], satisfying timing requirements [12, 13], and improving testability [14]. Herein, we also demonstrate how rewiring can be used to minimize the soft error rate of a design. Thus, we advocate that rewiring can be used as the cornerstone of a common framework for exploring the trade-off space between area, power consumption, delay, testability, and SER. We start, in Section 2, by describing an ATPG-based rewiring method which we use to generate functionally-equivalent yet structurally-different gate-level circuit implementations through a set of simple transformation rules. We then illustrate using simple examples, in Section 3, how these transformation rules may reduce the soft error rate of a circuit. Then, in Section 4, we propose an algorithm which evolves a design through iterative selection of rewiring operations, in order to optimize a cost function reflecting both the soft error rate and the rest of the design parameters of a circuit. Finally, in Section 5, we evaluate the proposed method using ISCAS'89 and ITC'99 benchmarks.

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

2 ATPG-based Rewiring

The underlying principle of rewiring is the exploration of the space of functionally-equivalent but structurally-different implementations of a circuit, in order to optimize a given cost function. Typically, rewiring methods [8, 9, 10, 11, 12, 13, 14, 15, 16, 17] target a wire that violates some constraint(s), called the *target wire*, and delete it from the implementation. Subsequently, they apply the transformations necessary for correcting the functionality of the design.

For the purpose of the work described in this paper, we use as a starting point the ATPG-based rewiring method described in [18]. This method first introduces a design error based on a subset of the common design error models proposed in [19]. In particular, this rewiring tool supports the following design error models, which are illustrated in Fig. 1:

1. **Missing Input Wire:** This is the design error that is commonly performed by most rewiring tools. The error is introduced by removing the target wire.
2. **Incorrect Input Wire:** The target wire is replaced by another wire that has similar logic values (i.e. a wire that, with a probability of 0.75 or higher, obtains the same value as the target wire).
3. **Gate Replacement:** The type of the gate driven by the target wire is changed depending on the probability of the logic values of the target wire. If the probability of a Logic 0 (Logic 1) on the target wire is higher than 0.75, the gate is changed to (N)AND ((N)OR). This error model is novel to the work described herein.
4. **Extra Input Wire:** A wire with similar logic values to the target wire is added to a gate driven by the target wire. This error model is also novel to this work.

After the design error is introduced, the rewiring tool attempts to correct the design using a simulation-based *Design Error Diagnosis and Correction (DEDC)* algorithm, which returns a list of corrections that rectify the design. The corrections returned by the rewiring tool can only correct a design by performing a single correcting operation. Therefore, DEDC will fail to find corrections if:

- The target wire is a stem. In this case, the error models *Missing Input Wire*, *Gate Replacement*, and *Extra Input Wire* are not applicable since the design error will introduce multiple errors at the branches of the stem.
- The gate driven by the target wire is an inverter or a buffer. In this case, the error models *Gate Replacement* and *Extra Input Wire* cannot be applied. In both cases, additional information is required to complete the error injection. For example, changing a buffer to a NAND gate requires an additional input wire to be specified. Hence, the circuit will have two design errors.

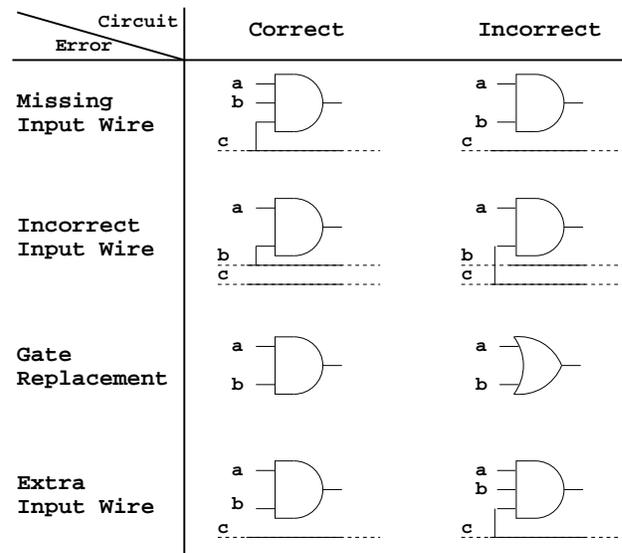


Figure 1. Supported Design Error Models

- The gate driven by the target wire is a 2-input gate. In this case, the *Missing Input Wire* error model cannot be applied since the circuit will also have two design errors, i.e. *Missing Input Wire* and *Gate Replacement*.

Finally, the corrections are verified using ATPG. Verification is necessary since the DEDC algorithm ensures the validity of a correction using a subset of the complete input vector space and, therefore, the corrections returned are only valid for this particular subset of vectors [21]. Details regarding the implementation of the ATPG-based verification step are beyond the scope of this paper and can be found in [18].

3 Impact of Rewiring on SER

The SER of a combinational circuit is proportional to three factors [1, 22]: i) the rate of occurrence of an SET at a gate (R_{SET}), ii) the probability of an SET reaching an output based on the current inputs to the circuit (P_{sens}), and iii) the probability that an SET is latched in a storage element (P_{latch}). Among these factors, rewiring primarily impacts P_{sens} . The probability of an SET reaching an output, P_{sens} , is measured by performing fault-simulation of the circuit for that SET and computing the percentage of times that the output responses were erroneous. Since rewiring changes the sensitization paths through which SETs may propagate to the outputs, the P_{sens} of any given SET may either increase or decrease, depending on the activation likelihood of its new sensitization paths. Thus, in order to assess the effectiveness of a rewiring operation, its impact on the P_{sens} of all SETs in the circuit should be taken into account [22, 23]. Throughout this work, computation of the SER is performed using SERA [23], which takes into account all the aforementioned factors.

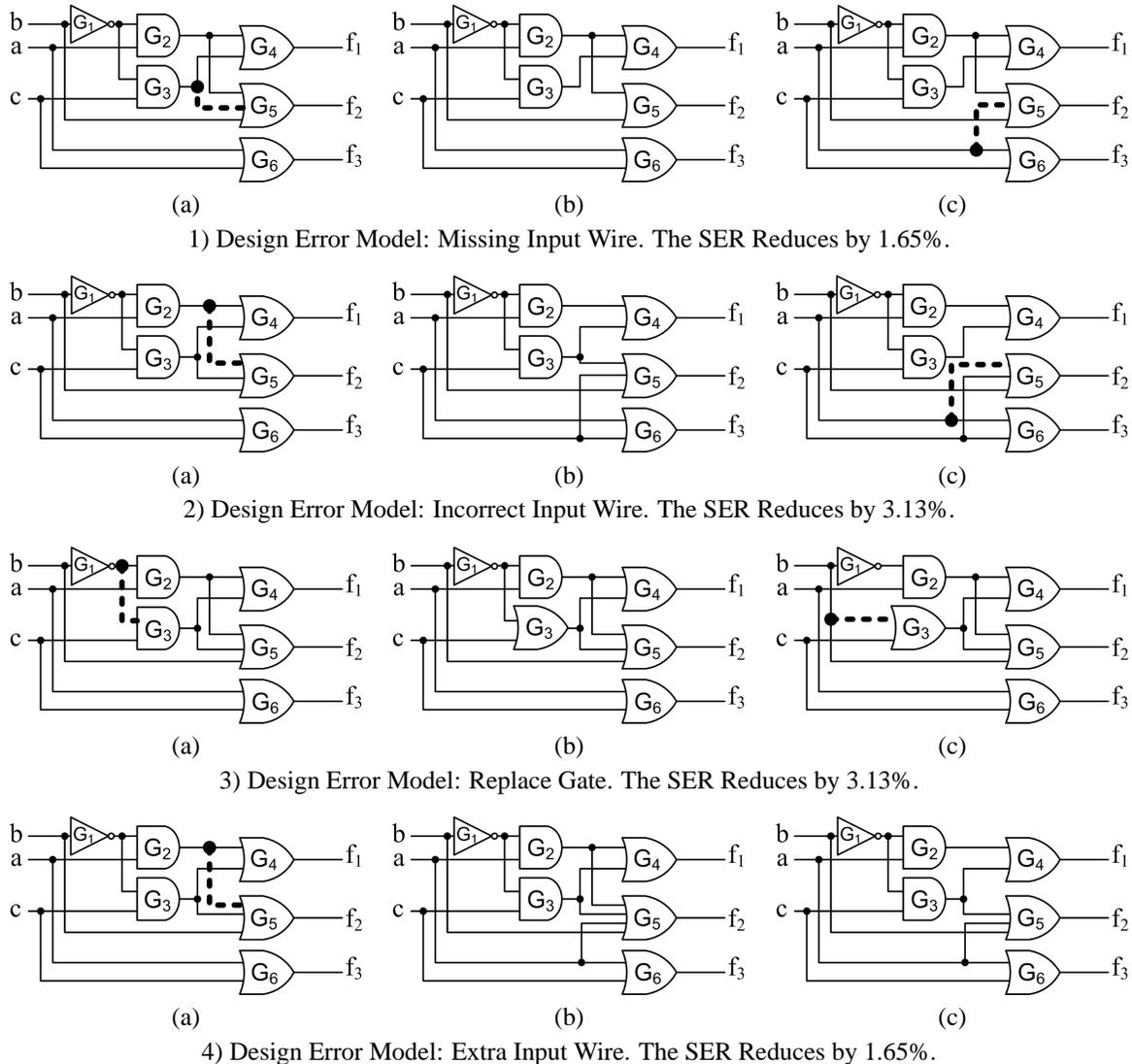


Figure 2. Examples of Rewiring Operations that Reduce the SER Using the 4 Design Error Models ((a) Target of Error, (b) Circuit After Error, and (c) Circuit After DEDC).

In the following, we provide simple examples to demonstrate that each of the four error models supported by the rewiring tool may result in corrections that reduce the SER.

3.1 Missing Input Wire Error Model

In the *missing input wire* error model, the target wire is removed from the circuit and the DEDC algorithm rectifies the design using a single correcting operation. The P_{sens} of any wire may either increase or decrease in the corrected circuit and, thus, the aggregate impact might be favorable, reducing the SER of the circuit. For example, in the circuit shown in Fig. 2.1.a, rewiring deletes $G_3 \rightarrow G_5$, which results in the incorrect implementation in Fig. 2.1.b. During the DEDC stage, $a \rightarrow G_5$ is identified as a possible correction and is added to the circuit, as illustrated in Fig. 2.1.c. The SER of

the corrected circuit in Fig. 2.1.c, reduces by 1.65% over the initial circuit in Fig. 2.1.a.

3.2 Incorrect Input Wire Error Model

In the *incorrect input wire* error model, the target wire is replaced by another wire in the circuit and the DEDC algorithm rectifies the design using a single correcting operation. Thus, the arbitrary impact on the P_{sens} of the wires in the corrected circuit may result in an overall reduction of the SER. For example, in the circuit shown in Fig. 2.2.a, rewiring substitutes $G_2 \rightarrow G_5$ with $c \rightarrow G_5$, resulting in the circuit shown in Fig. 2.2.b. DEDC identifies $a \rightarrow G_5$ as a possible correction and adds it to the circuit, as shown in Fig. 2.2.c. The SER of the corrected circuit in Fig. 2.2.c, reduces by 3.13% over the SER of the initial circuit in Fig. 2.2.a.

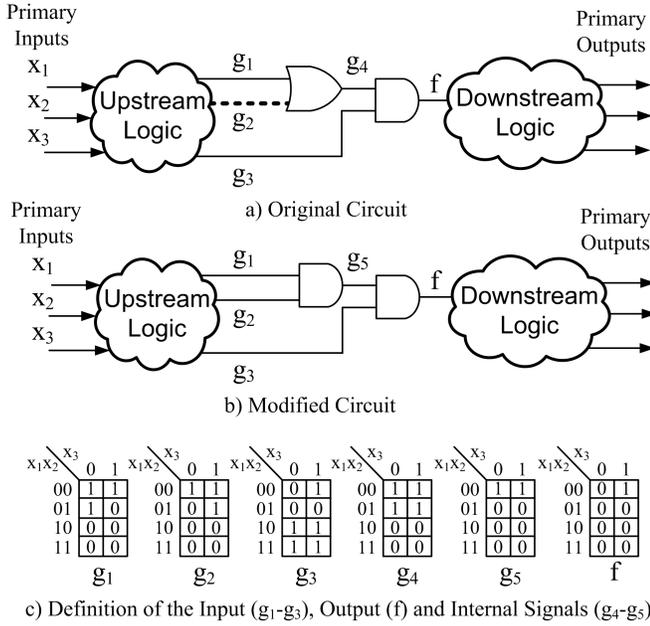


Figure 3. Example Illustrating the Intuition Behind the Gate Replacement Error Model.

3.3 Gate Replacement Error Model

The *gate replacement* error model, which was added to the rewiring tool implementation of [18] for the purpose of this work, changes the type of the gate driven by the target wire. This, in turn, reduces the P_{sens} of one of the two possible SETs (i.e. $0 \rightarrow 1$ and $1 \rightarrow 0$) that can affect the target wire for the following reason. If the target wire has a high probability of obtaining a non-controlling value of the gate that it drives, then an SET that flips the target wire to the controlling value of the gate has a high probability of propagating to its output. Consequently, P_{sens} will be high for that particular SET. By introducing an error that changes the type of the gate, the target wire will now have a high probability of obtaining a controlling value of this gate. Therefore, the same SET will now flip the target wire to the non-controlling value of the gate and, hence, has a low probability of propagating to its output. By extension, P_{sens} will now be reduced for this particular SET.

The intuition behind the gate replacement error model is illustrated using the example in Fig. 3, which shows part of a logic circuit. Let f be the output function of this sub-circuit and g_1 , g_2 and g_3 be the input functions, expressed in terms of the primary inputs x_1 , x_2 and x_3 , as illustrated in the Karnaugh maps of Fig. 3.c, and assume that g_2 is the target wire for rewiring. Since g_2 has a high probability of obtaining a logic value of 0 (a non-controlling value of the OR gate that it drives), a $0 \rightarrow 1$ SET on g_2 has a high probability of propagating to the output of the OR gate. Conversely, a $1 \rightarrow 0$ SET on g_2 has a low probability of propagating to the output of the

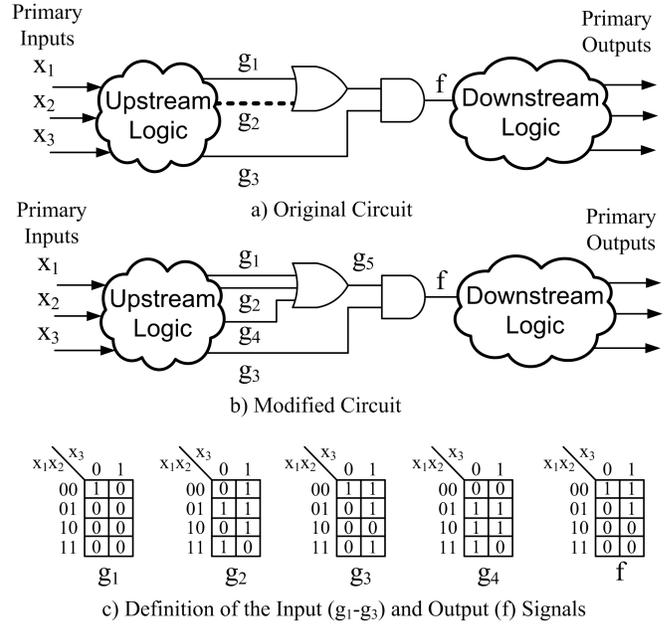


Figure 4. Example Illustrating the Intuition Behind the Extra Input Wire Error Model.

OR gate. When the error is introduced by changing the gate type to an AND gate, and after correcting the design error in the modified circuit of Fig. 3.b¹, the probability of a $0 \rightarrow 1$ SET on g_2 propagating to the output of the AND gate reduces, while the probability of a $1 \rightarrow 0$ SET propagating to the output increases. Yet, it is possible that the aggregate impact will be favorable, reducing the SER of the circuit. For example, consider the circuit shown in Fig. 2.3.a, wherein the AND gate G_3 is replaced by an OR gate, resulting in the incorrect circuit of Fig. 2.3.b. Then, the DEDC algorithm corrects the design by replacing $\bar{b} \rightarrow G_3$ with $b \rightarrow G_3$, as shown in Fig. 2.3.c and the SER of the corrected circuit reduces by 3.13% over the SER of the initial circuit.

3.4 Extra Input Wire Error Model

The *extra input wire* error model adds a similar wire to the gate driven by the target wire. This, in turn, reduces the P_{sens} of one of the two possible SETs on the target wire for the following reason. If the target wire has a high probability of obtaining a controlling value of the gate, an SET that flips the target wire to the opposite (non-controlling) value will propagate to the output of the gate, unless another input of the gate also has a controlling value. Thus, by adding a similar wire as an extra input to the gate, we increase the probability that such a controlling value will exist and, by extension, the probability that the SET will be masked.

¹The output of the circuit in the example is not affected by the change in the gate type and, thus, no correction is necessary.

The intuition behind the extra input wire error model is illustrated using the example in Fig. 4. Let g_2 be the target wire for rewiring and let $g_1 - g_4$ be defined as illustrated in Fig. 4.c. The functionality of g_4 is similar to g_2 (identical for 75% of the possible input combinations) and is added as an input to the OR gate in the modified circuit in Fig. 4.b. Since g_2 has a high probability of obtaining a logic value of 1 (a controlling value of the OR gate that it drives), a $1 \rightarrow 0$ SET on g_2 has a high probability of propagating to the output of the OR gate. When g_4 is added in the modified circuit, however, the probability of a $1 \rightarrow 0$ SET propagating to the output of the OR gate is reduced since g_4 will have a controlling value of the OR gate with high probability. While the addition of g_4 introduces a new location where SETs might appear, the $1 \rightarrow 0$ SET on g_4 will often be masked at the output of the OR gate as g_2 will also have the controlling value of the gate with high probability. On the other hand, a $0 \rightarrow 1$ SET on g_4 might propagate to the output of the OR gate with high probability. Yet, it is possible that the overall effect will be a reduction in the SER of the circuit. For example, consider the target wire $G_2 \rightarrow G_5$ in the circuit of Fig. 2.4.a. Wire $a \rightarrow G_5$, which is similar to $G_2 \rightarrow G_5$ since it obtains the same value in 75% of the input combinations, is added in the implementation of Fig. 2.4.b. The DEDC algorithm corrects the circuit by deleting $G_2 \rightarrow G_5$, as shown in Fig. 2.4.c, thus reducing the SER by 1.65%.

4 Rewiring-Based Optimization Algorithm

As demonstrated in the previous section, rewiring operations may, indeed, reduce the SER of a circuit. Furthermore, rewiring operations have previously also been shown to significantly improve area, power consumption, delay, and testability [9, 10, 11, 12, 13]. Based on these observations, in this section we devise an algorithm that iteratively selects effective rewiring operations and evolves the circuit implementation in order to optimize a cost function reflecting a given set of design objectives.

The selection of an optimal set of rewiring operations is NP-complete and, thus, computationally infeasible. In the proposed algorithm, we follow a simple greedy heuristic, wherein, at each step, rewiring is attempted for the wire with the highest P_{sens} in the circuit that has not been tried so far. In order to identify the most susceptible wire, we employ fault simulation of random patterns and compute the P_{sens} for each wire by taking the ratio of the number of times that faults on the wire are sensitized to an output over the number of simulated input patterns. Then, the list of wires is sorted ($SortWires()$) in decreasing order of their P_{sens} and the first wire in the list is selected as a target wire ($TargetWire$). Once the target wire is selected, we perform rewiring using the four design error models of Fig. 1 and the DEDC algorithm generates a list of k_i corrections to fix the design. For each candidate correction j , we construct the corrected cir-

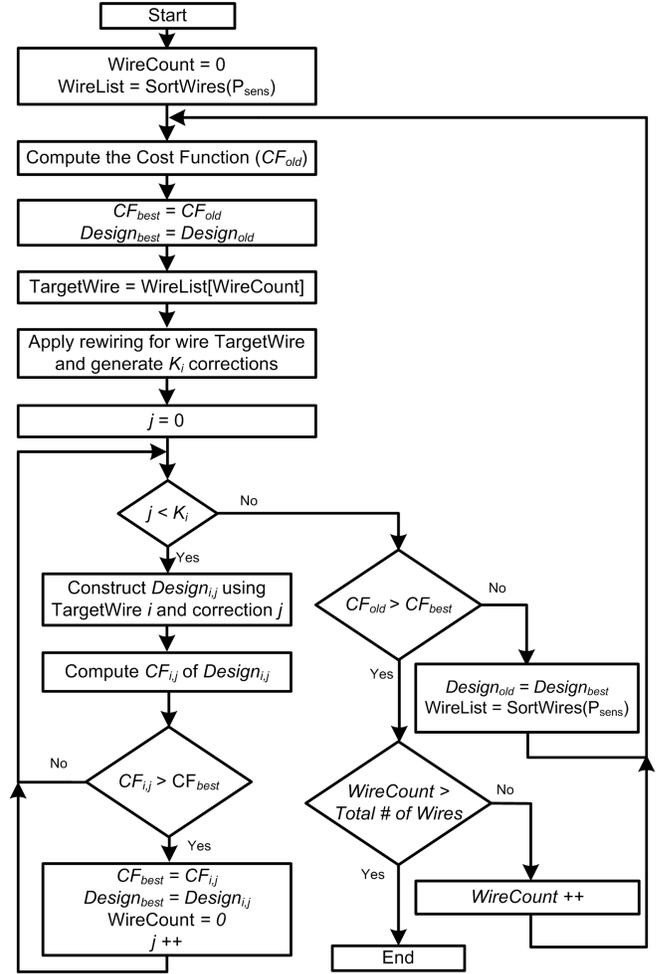


Figure 5. Flowchart of the Proposed Algorithm

cuit ($Design_{i,j}$), evaluate the cost function ($CF_{i,j}$), and keep the design ($Design_{best}$) that has the best improvement to the cost function (CF_{best}) over the previous design ($Design_{old}$). The process is iteratively repeated until all the wires have been tried without improvement to the cost function. The algorithm is summarized in Fig. 5.

5 Experimental Results

In this section, we evaluate experimentally the SER reduction and associated overheads for the proposed rewiring-based design space exploration method. First, in section 5.1, we describe the setup of the experiments. Next, in section 5.2, we discuss four cost functions that we used to drive the optimization algorithm in our experiments. Then, in section 5.3, we present, analyze, and compare the results for these cost functions. Finally, in section 5.4, we discuss the shortcomings of existing SER estimation tools, eluding to the fact that the effectiveness and scalability of rewiring-based SER reduction will drastically improve as these tools mature.

5.1 Experimental Setup

We experiment with a set of ISCAS'89 and ITC'99 benchmark circuits. The SER is computed using SERA [23], which accounts for the rewiring effect on R_{SET} , P_{sens} and P_{latch} , and reports the SER for each output of the circuit. The area overhead is computed based on transistor counts of the original and final circuits. Power and delay overhead computation is performed using SIS [24]. The internal BDD-based power simulator in SIS is used to compute the power overhead assuming a zero-delay model. The circuit is, then, mapped to the standard *lib2.genlib* library, and the delay of the most critical path is used for computing the delay overhead. Finally, ATALANTA [25] is used to perform Automatic Test Pattern Generation (ATPG) and compute any loss in fault coverage during production testing.

5.2 Optimization Cost Functions

Rewiring has already been shown to be effective in optimizing area, power, delay and testability [9, 10, 11, 12, 13]. Therefore, we mainly focus on cost functions that reduce the SER while varying the constraints placed on the other design parameters. Thus, the first cost function we consider aims at minimizing the soft error rate, regardless of the impact of rewiring on the other design parameters such as area, delay, power and testability. Let $improv(x)$ be a function that returns the ratio between parameter x of the initial circuit over the same parameter of the circuit after the rewiring operation. Then, the first cost function, called *OnlySER*, can be expressed as:

$$OnlySER = \max\{improv(SER)\} \quad (1)$$

The second cost function, called *SERandTest*, aims at reducing the SER of the circuit as long as no additional untestable faults are introduced in the circuit after the rewiring operation. Hence, the *SERandTest* cost function can be represented as:

$$\begin{aligned} SERandTest &= \max\{improv(SER)\}, \\ \text{Subject to:} & \quad improv(Testability) \geq 0 \end{aligned} \quad (2)$$

The third cost function, called *SERandAll*, reduces the SER as long as all the design parameters of the modified circuit after the rewiring operation are better than or equal to the design parameters of the initial circuit. The *SERandAll* cost function is defined as:

$$\begin{aligned} SERandAll &= \max\{improv(SER)\}, \\ \text{Subject to:} & \quad improv(Area) \geq 0 \\ & \quad improv(Delay) \geq 0 \\ & \quad improv(Power) \geq 0 \\ & \quad improv(Testability) \geq 0 \end{aligned} \quad (3)$$

Finally, the function *JointOptimization* enables the joint optimization of all the design parameters, with the ability to prioritize the various optimization objectives using different weights for the corresponding design parameters. Thus, it enables the designer to optimize the overhead of the design based on the target application of the product. The *JointOptimization* cost function is defined as:

$$\begin{aligned} JointOptimization &= \max\{w_1 \cdot improv(SER) \\ & \quad + w_2 \cdot improv(Area) \\ & \quad + w_3 \cdot improv(Delay) \\ & \quad + w_4 \cdot improv(Power) \\ & \quad + w_5 \cdot improv(Testability)\} \end{aligned} \quad (4)$$

where $0 \leq w_i \leq 1$ and $\sum_{i=1}^{i=5} w_i = 1$. The weights were set in these experiments to $w_1 = 0.5$, $w_2 = w_3 = w_4 = 0.1$, and $w_5 = 0.2$, giving higher priority to the SER reduction and the improvement in testability of a circuit implementation over the reduction in area, delay and power consumption. While we only present results using the above four cost functions, any other cost function can be used to drive the search algorithm reflecting the constraints placed by the designer on the overhead of the final design.

5.3 Analysis and Comparison

The results are presented in Table 1 for the *OnlySER* and *SERandTest* cost functions, and in Table 2 for the *SERandAll* and *JointOptimization* cost functions, respectively. Under the first major heading, we provide details about the circuits that were used: name, number of primary inputs, number of primary outputs, and gate count. Under the next two major columns, we report the percentile SER reduction², area overhead, power overhead, delay overhead, and fault coverage loss for the *OnlySER* and *SERandTest* cost functions (*SERandAll* and *JointOptimization* cost functions) in Table 1 (Table 2), respectively. The key points revealed by these results are summarized below:

- The results for *OnlySER* indicate that rewiring can reduce substantially the SER of the circuit. For example, the SER of *b01*, *b06* and *s510* is reduced by more than 25%. However, when the search is driven by the sole objective of reducing SER, the impact of rewiring on other design parameters such as area, power, delay, and testability can be significant. For example, the delay of *b06* and *s510* increases by more than 40%.

²We note that the computation of SER reduction takes into account all possible SET locations, including the ones in the additional area incurred by the rewiring method, if any.

Original Circuit				<i>OnlySER</i>					<i>SERandTest</i>				
Name	PI	PO	Gates	SER	Area	Delay	Power	F. C. Loss	SER	Area	Delay	Power	F. C. Loss
b01	7	7	51	26.62%	18.29%	4.92%	-5.92%	4.35%	17.74%	7.32%	-21.78%	-3.27%	0.00%
b02	5	5	27	5.63%	9.30%	29.53%	-6.52%	3.57%	1.66%	6.98%	0.00%	-2.69%	0.00%
b03	34	34	153	0.19%	0.40%	3.02%	0.05%	0.00%	0.19%	0.40%	3.02%	0.05%	0.00%
b06	11	15	55	26.79%	22.35%	50.60%	-11.93%	9.77%	19.84%	10.59%	-16.77%	-28.70%	0.00%
b08	30	25	171	9.94%	20.21%	62.99%	10.94%	10.25%	2.01%	0.34%	3.61%	-7.68%	0.00%
b09	29	29	160	11.47%	2.59%	14.86%	-1.63%	2.8%	10.76%	-1.48%	-2.24%	-8.89%	0.00%
b10	28	23	180	21.42%	11.24%	48.01%	5.07%	7.13%	11.44%	5.92%	22.23%	-1.67%	0.00%
s298	17	20	119	20.41%	-3.69%	16.13%	6.78%	11.81%	14.91%	-10.39%	-10.99%	-24.31%	0.00%
s382	24	27	158	19.53%	8.52%	3.60%	-9.42%	7.71%	14.77%	0.00%	-12.89%	-17.06%	0.00%
s344	24	26	160	7.14%	2.60%	10.62%	29.51%	6.02%	3.12%	1.86%	-0.98%	0.10%	0.00%
s349	26	26	161	7.17%	6.96%	-0.51%	5.07%	7.21%	5.81%	0.00%	2.02%	5.36%	0.00%
s526	24	27	173	11.71%	1.57%	-18.49%	-18.07%	3.10%	8.22%	-3.52%	0.74%	-22.29%	0.00%
s444	24	27	181	14.52%	1.99%	-9.85%	-10.18%	6.29%	12.91%	-0.28%	-0.13%	-11.79%	0.00%
s510	25	13	211	25.39%	12.74%	48.01%	5.07%	7.13%	15.03%	2.80%	18.20%	1.42%	0.00%

Table 1. Experimental Results on ISCAS89 & ITC99 Benchmark Circuits (*OnlySER* and *SERandTest*)

Original Circuit				<i>SERandAll</i>					<i>JointOptimization</i>				
Name	PI	PO	Gates	SER	Area	Delay	Power	F. C. Loss	SER	Area	Delay	Power	F. C. Loss
b01	7	7	51	15.65%	0.00%	-18.85%	-6.72%	0.00%	9.93%	1.22%	-4.40%	-7.22%	0.00%
b02	5	5	27	0.00%	0.00%	0.00%	0.00%	0.00%	1.44%	0.00%	-9.17%	-12.63%	0.00%
b03	34	34	153	0.17%	0.00%	-2.53%	-1.32%	0.00%	3.84%	0.79%	1.81%	-0.08%	0.00%
b06	11	15	55	15.15%	-1.18%	-7.93%	-13.40%	0.00%	9.90%	0.00%	-7.49%	-12.35%	0.00%
b08	30	25	171	3.76%	0.00%	-4.46%	-9.33%	0.00%	2.68%	1.03%	-25.25%	-11.84%	0.92%
b09	29	29	160	6.46%	-0.74%	-30.51%	-2.16%	0.00%	11.28%	6.54%	-26.43%	-12.94%	0.94%
b10	28	23	180	11.32%	-0.59%	-7.11%	-3.09%	0.00%	7.88%	3.25%	-8.48%	1.38%	0.84%
s298	17	20	119	12.15%	-5.77%	-18.27%	-26.36%	0.00%	7.17%	-6.15%	-30.17%	-19.87%	0.96%
s382	24	27	158	17.05%	-0.63%	-4.03%	-12.24%	0.00%	11.45%	-0.32%	-3.31%	-9.73%	0.00%
s344	24	26	160	1.07%	0.00%	-7.84%	-1.65%	0.00%	1.49%	1.49%	-0.65%	2.64%	0.29%
s349	26	26	161	3.12%	0.00%	-2.53%	-1.64%	0.00%	2.79%	1.47%	0.00%	-6.69%	0.87%
s526	24	27	173	6.35%	-1.86%	-4.02%	-14.59%	0.00%	5.52%	2.86%	-13.97%	-7.65%	0.00%
s444	24	27	181	7.62%	-4.68%	-7.48%	-10.18%	0.00%	9.05%	1.65%	-6.84%	-2.68%	0.40%
s510	25	13	211	8.98%	-0.70%	-4.60%	-3.71%	0.00%	10.37%	4.21%	-25.72%	-00.89%	2.18%

Table 2. Experimental Results on ISCAS89 & ITC99 Benchmark Circuits (*SERandAll* and *JointOptimization*)

- By monitoring the impact on other design parameters during the search algorithm, we can moderate its effect. The results for *SERandTest* indicate that when no fault coverage loss is allowed during rewiring, the SER of the circuit can still be significantly reduced. For example, this is the case for *b06* and *s298*. While this reduction is smaller than in the *OnlySER* case, the impact on the remaining design parameters, i.e. area, delay, and power, is also less severe.
- Even when the search algorithm is constrained to only allow rewiring operations that do not cause a negative impact to *any* of the design parameters, the SER of the circuit is still reduced, as indicated by the results for the *SERandAll* cost function. As expected, however, the additional constraints placed on the search algorithm diminish the attained SER reduction. Nevertheless, this reduction is overhead-free and, as such, highly desirable. Moreover, this constrained design-space exploration often results in significant reduction in one or more of these design parameters. For example, the delay of *b09* is reduced by more than 20%, and the area overhead of *b06* is reduced by 10%.
- When design parameters are used as an inherent part of the cost function, rather than constraints, logic rewiring enables a more efficient exploration of the design space of the circuit under optimization. This is demonstrated through the results for *JointOptimization*, where SER is given the highest weight during the search, testability is given the next highest weight, and area, power, and delay are given the smallest weight. As a result, the final rewired circuit exhibits high SER reduction and minimal loss in fault coverage, while moderate impact -either positive or negative- is effected on area, power, and delay. For example, the SER of circuits *b09*, *s382* and *s510* is reduced by more than 10% while the overhead of other design parameters is also reduced.

5.4 Discussion

The above results demonstrate that rewiring can indeed be used to reduce the SER of a logic circuit and to facilitate a common optimization framework for logic-level design-space exploration. While the attained SER reduction is significant, we would like to point out that it is a very conservative and pessimistic indication of what rewiring can achieve. The underlying reason for this has to do with limitations related to SERA, the SER assessment tool that we used in our experiments. To the best of our knowledge, SERA is the only public-domain SER estimation tool, which is the reason for using it. SERA supports a generic $0.18\mu\text{m}$ CMOS library composed of standard gates that have a maximum of *three* inputs. Therefore, in order to assess the SER of a circuit that utilizes gates with more than three inputs, we need to split these gates into an equivalent structure that only uses gates with a maximum of three inputs. Such decomposition, however, has a very negative effect. First, the number of locations where SETs may occur in the circuit increases. Second, a gate with k inputs has, in general, a lower probability of masking SETs on its inputs than the same type of gate with n inputs, where $n > k$. Thus, gate decomposition increases the P_{sens} and, by extension, the SER of a circuit.

To make things even worse, the limited number of gate inputs results in fewer rewiring opportunities. This happens because the number of inputs to a gate after rewiring cannot exceed three, therefore preventing a large number of potential corrections from being considered. Overall, the potential of rewiring in reducing P_{sens} is precluded by the input width of the available gates. Hence, the results reported herein reflect very conservatively the SER reduction that rewiring would achieve on the benchmark circuits, should a library of gates with more than three inputs be supported by SERA.

Our conjecture is that the SER reduction achieved by rewiring on a logic circuit constructed out of gates with up to n inputs is higher than the SER reduction of rewiring on a logic circuit constructed out of gates with up to k inputs, where $k < n$. To support this claim, we plot in Fig. 6 the SER reduction obtained when the library is restricted to 2-input gates only, along with the SER reduction when the library is restricted to all of the supported gates in the library of SERA (i.e. both 2-input and 3-input gates), for several benchmark circuits. As can be seen from the figure, the SER reduction obtained using 2-input gates only is, on average, 56.46% of the SER reduction obtained using 2-input and 3-input gates. This result corroborates the conjecture that rewiring-based SER reduction is expected to increase if the supported library contains wider gates.

As a final note, we would like to comment on the accuracy and scalability of our method to address the concerns of the observant reader who may have noticed that results for the larger of the ISCAS'89 and ITC'99 benchmarks are not included in the experiments. The optimization algorithm relies

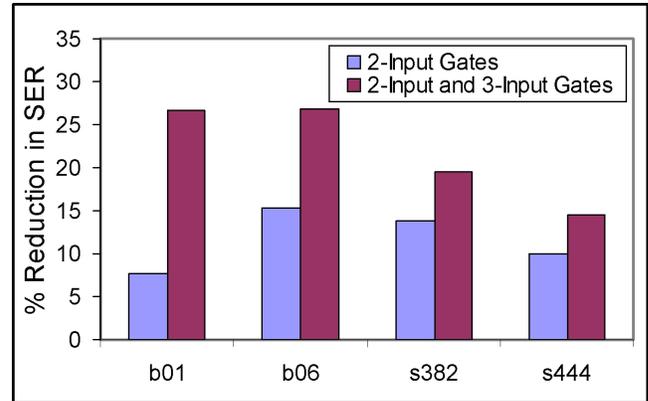


Figure 6. Comparison Between the SER Reduction Achieved by Rewiring Using 2-input Gates Only v.s. Using 2-Input and 3-Input Gates.

on two distinct tools, a rewiring tool and an SER estimation tool. In its present form, the algorithm employs the ATPG-based method described in [18] for rewiring and SERA [23] for SER estimation. The accuracy of the SER reduction results depends on the approximation and estimation methods of the underlying SER estimation tool. In terms of scalability, ATPG-based rewiring has been shown to require less than a second to perform rewiring for circuits with more than $3K$ gates [18], so scalability is not a concern. SERA, on the other hand, requires a significant amount of time for larger circuits, hence the lack of results for such benchmarks. However, development of SER estimation tools has been a very active research area in recent years [3, 4, 5, 6]. As these tools mature and become more efficient, methods employing them, such as the rewiring-based SER reduction described in this paper, will also be positively affected.

6 Conclusion

In addition to the various design parameters that rewiring has been shown to improve in the past, this work demonstrated that rewiring can also be used to reduce the SER of a circuit. Thus, rewiring provides an excellent basis for constructing a unified optimization framework for exploring the trade-offs between area, power consumption, delay, testability, and SER. To this end, we described an ATPG-based rewiring method that generates functionally-equivalent yet structurally-different implementations of a logic circuit using a set of simple transformation rules. We demonstrated how these transformations result in circuit implementations with reduced SER and we presented a search algorithm that iteratively evolves a design in order to satisfy a given set of design objectives. Experimental results on ISCAS'89 and ITC'99 benchmark circuits verify that SER reduction can be seamlessly and effectively integrated in the list of objectives supported by rewiring-based design-space exploration.

References

- [1] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *International Conference on Dependable Systems and Networks*, 2002, pp. 389–398.
- [2] M. Okabe, M. Tatsuki, Y. Arima, T. Hirao, and Y. Kuramitsu, "Design for reducing alpha-particle-induced soft errors in ECL logic circuitry," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 5, pp. 1397–1403, 1989.
- [3] Y. S. Dhillon, A. U. Diril, A. Chatterjee, and A. D. Singh, "Sizing CMOS circuits for increased transient error tolerance," in *IEEE International On-Line Testing Symposium*, 2004, pp. 11–16.
- [4] H. Deogun, D. Sylvester, and D. Blaauw, "Gate-level mitigation techniques for neutron-induced soft error rate," in *ACM/IEEE International Symposium on Quality Electronic Design*, 2005, pp. 175–180.
- [5] S. Krishnamohan and N. R. Mahapatra, "Combining error masking and error detection plus recovery to combat soft errors in static CMOS circuits," in *International Conference on Dependable Systems and Networks*, 2005, pp. 40–49.
- [6] Q. Zhou and K. Mohanram, "Cost-effective radiation hardening technique for logic circuits," in *IEEE/ACM International Conference on Computer-Aided Design*, 2004, pp. 100–106.
- [7] S. Mitra, M. Zhang, T.M. Mak, N. Seifert, V. Zia, and K.S. Kim, "Logic soft errors: A major barrier to robust platform design," in *International Test Conference*, 2005, pp. 687–698.
- [8] S. C. Chang, K. T. Cheng, N. S. Woo, and M. Marek-Sadowska, "Postlayout logic restructuring using alternative wires," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 6, pp. 587–596, 1997.
- [9] L. A. Entrena and K. T. Cheng, "Combinational and sequential logic optimization by redundancy addition and removal," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 7, pp. 909–916, 1995.
- [10] W. Kunz, D. Stoffel, and P. R. Menon, "Logic optimization and equivalence checking by implication analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 3, pp. 266–281, 1997.
- [11] B. Rohfleisch, A. Kolbl, and B. Wurth, "Reducing power dissipation after technology mapping by structural analysis," in *Design Automation Conference*, 1996, pp. 789–794.
- [12] Y. M. Jiang, A. Krstic, K. T. Cheng, and M. Marek-Sadowska, "Postlayout logic restructuring for performance optimization," in *Design Automation Conference*, 1997, pp. 662–665.
- [13] G. Stenz, B. M. Riess, B. Rohfleisch, and F. M. Johannes, "Performance optimization by interacting netlist transformations and placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 3, pp. 350–358, 2000.
- [14] M. Chatterjee, D. Pradhan, and W. Kunz, "LOT: Logic optimization with testability - new transformations using recursive learning," in *IEEE/ACM International Conference on Computer-Aided Design*, 1995, pp. 115–118.
- [15] S. Yamashita, H. Sawada, and A. Nagoya, "SPFD: A new method to express functional flexibility," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 8, pp. 840–849, 2000.
- [16] S. P. Khatri, S. Sinha, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SPFD-based wire removal in standard-cell and network-of-PLA circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 7, pp. 1020–1030, 2004.
- [17] J. Cong, J. Y. Lin, and W. Long, "A new enhanced SPFD rewiring algorithm," in *IEEE/ACM International Conference on Computer-Aided Design*, 2002, pp. 672–678.
- [18] A. Veneris and M. Abadir, "Design Rewiring Using ATPG," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1469–1479, 2002.
- [19] M. S. Abadir, J. Ferguson, and T. E. Kirkland, "Logic Verification via Test Generation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 1, pp. 138–148, 1988.
- [20] C. W. Chang and M. Marek-Sadowska, "Single-pass redundancy addition and removal," in *IEEE/ACM International Conference on Computer-Aided Design*, 2001, pp. 606–609.
- [21] A. Veneris and I. N. Hajj, "Design Error Diagnosis and Correction Via Test Vector Simulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 12, pp. 1803–1816, 1999.
- [22] K. Mohanram and N. A. Touba, "Cost-effective approach for reducing soft error failure rate in logic circuits," in *International Test Conference*, 2003, pp. 893–901.
- [23] M. Zhang and N. R. Shanbhag, "A soft error rate analysis (SERA) methodology," in *IEEE/ACM International Conference on Computer-Aided Design*, 2004, pp. 111–118.
- [24] 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.
- [25] H. K. Lee and D. S. Ha, "Atalanta: an efficient ATPG for combinational circuits," Technical Report, 93-12, Department of Electrical Engineering, Virginia Polytechnic Institute and State University, 1993.