Microelectronics Reliability 51 (2011) 2388–2401

Contents lists available at ScienceDirect

Microelectronics Reliability

journal homepage: www.elsevier.com/locate/microrel

A fast and efficient technique to apply Selective TMR through optimization

Oscar Ruano, Juan Antonio Maestro *, Pedro Reviriego

Universidad Antonio de Nebrija, C/Pirineos, 55, E-28040 Madrid, Spain

ARTICLE INFO

Article history:
Received 10 December 2010
Received in revised form 6 July 2011
Accepted 6 July 2011
Available online 4 August 2011

ABSTRACT

Fault tolerance is an important factor for circuits in critical applications, especially those working in harsh environments. There are many techniques to increase reliability of circuits, being those based on redundancy very popular. In this way, Triple Modular Redundancy (TMR) is frequently used, but it usually incurs high area costs. That is why other alternative techniques, as Selective TMR, are used in order to reduce this cost. In this technique, only a subset of registers is tripled, those that are more sensitive and produce a higher error rate in the circuit. However, the problem of these methodologies is the complexity of finding the optimal set of registers to triple, what usually leads to very high computation times. In this paper, a novel solution that improves Selective TMR is presented, based on the automatic and fast calculation of an initial partition prior to the optimization process. The solution has been tested on a real communication circuit, a Feed-Forward Equalizer.

1. Introduction

Fault tolerance on semiconductor devices has been a relevant matter since upsets were first experienced in space applications many years ago [1]. Integrated circuits operating in the space environment [17] can be upset by charged particles that generate errors in the system. Therefore, the interest in studying fault-tolerant techniques to keep integrated circuits operational has increased [24]. In order to guarantee circuit reliability against Single Event Upsets (SEU) [2], some mitigation techniques have been proposed in the literature during the last decades [22,23,25,26]. The design-based techniques, also called architectural techniques, are divided into three groups:

- The first one refers to the standard techniques based on logic redundancy, such as Triple Modular Redundancy (TMR), that triplicates each register and adds a majority voter [3], or Error Detection and Correction coding (EDAC) like Hamming [4]. One advantage of these techniques when dealing with SEUs is that they are generic, and therefore they can be applied to most digital circuits in the same way without difficulty. However, this comes at a high cost in terms of circuit area and power [21].
- The second one corresponds to ad hoc techniques that take advantage of the so-called system knowledge. This kind of approaches consists in applying specific techniques that exploit the inherent redundancy of some circuits, like in [5,6]. The advantage of this is the generation of custom-tailored solutions for each circuit, with a good protection level and a more favorable implementation (what general techniques as TMR cannot achieve). However, they need a higher effort to be designed.
- The third option refers to a set of hybrid techniques based on TMR, known as Selective TMR [7,8,18,19], which are applied in an ad hoc way, tripling only the registers of the circuit that are more sensitive to SEUs. This approach is valid for those applications that do not require 100% reliability, i.e., they tolerate a limited amount of errors without affecting its functionality. Through this solution, the area cost of the circuit can be highly reduced compared to the standard TMR approach. These techniques can be applied manually or automatically. The main problem is that the detection of the minimal set of sensitive registers to be tripled is not straightforward and usually consumes too much time.

In Ref. [8], an automatic Selective TMR solution is implemented through an optimization process based on an iterative algorithm [9], which automatically determines the optimal set of registers that need to be triplicated. However, this methodology can spend too much computation time if the target design has many registers. There are several ways that can mitigate this problem, such as to utilize improved optimization algorithms or to use faster simulation processes.

In this paper, we propose an alternative solution to address this problem. It consists in automatically calculating an initial design, close enough to the optimal solution, so that the computation time needed by the optimization engine to reach the latter, starting with the former, is greatly reduced.

In order to test this methodology in a realistic scenario, an adaptive filter (a Feed Forward Equalizer) [10,11] has been chosen, as these circuits are widely used in communications receivers. The

* Corresponding author. Tel.: +34 914521100; fax: +34 914521110.
E-mail addresses: oruano@nebrija.es (O. Ruano), jmaestro@nebrija.es (J.A. Maestro), previrie@nebrija.es (P. Reviriego).

0026-2714/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.microrel.2011.07.020

results will also be put in perspective with a wide set of experiments.

This paper is organized as follows. Section 2 briefly introduces the selective TMR technique in terms of a classical optimization process. In Section 3, a model that accelerates the optimization process through the selection of a good initial partition is presented. Sections 4 and 5 develop the model, introducing a set of topological criteria that allow a quick evaluation of the process. Some experimental results and a case study based on a Feed-Forward Equalizer will be offered in Sections 6 and 7 respectively. Final remarks and future work are placed in Section 8, followed by the references.

2. Selective TMR as an optimization problem: performance issues

The goal of the Selective TMR methodology [7] is to optimize the high area cost necessary to protect the whole circuit through full TMR. It can only be applied when a certain error margin in the circuit behavior (Error Rate) is allowed. In other words, this methodology gets area savings at the expense of reducing the 100% protection against isolated errors achieved with full TMR.

The idea consists in providing an automatic way to determine the minimal set of nodes (being a node each of the registers of a circuit, once it has been represented as a graph) that need to be tripled (minimal area), so that the imposed error rate constraints are met. This technique is usually based on complex combinatorial optimization problems driven by iterative algorithms [9]. More formally, let us define a circuit system characterized by the following inputs:

- $\{x_i\}$ is the whole set of registers and,
- $[r_s]$ is the reliability constraints imposed to each output, expressed in terms of the Error Rate (ER).

Then, the goal is to find a solution characterized by the following two partitions,

- $\{t_j\}$ is the set of nodes to be tripled and,
- $\{n_k\}$ is the set of nodes to remain unprotected.

so that the following is met:

- $\{t_j\} \cup \{n_k\} = \{x_i\}$,
- $\{t_j\}$ area cost is minimal,
- $[r_s]$ is met.

This can be represented as a classical optimization process (Fig. 1).

As can be seen, the optimization process starts with a default initial partition $\{t'_j\}, \{n'_k\}$. Then, it is evaluated using the estimator, the cost function and the reliability constraints $[r_s]$ imposed to each output. Finally, the optimization engine tries to find a better solution that minimizes the area while still meeting the constraints. This process is repeated until the optimal design is found (or the algorithm is trapped in a local minimum).

The different modules depicted in Fig. 1 are detailed in terms of the previous formulation and the implementation used in [8]:

(1) *Inputs*: $\{x_i\}$, $[r_s]$ and an initial default partition.
(2) *Output*: $\{t_j\} \cup \{n_K\} = \{x_i\}$, that meets $[r_s]$, with a minimal area cost of $\{t_j\}$.
(3) *Optimization modules:*
   (a) *Engine*: it is based on an iterative algorithm that drives the optimization process towards the optimal solution. Details about how these algorithms work can be found in Ref. [12]. Basically, what the engine does is to sequentially explore all the adjacent solutions to the initial point in order to determine which one is closer to the optimal while meeting the reliability constraints. This is achieved by characterizing each solution with its cost function (see below) and choosing the best one. Once this selection has been made, the algorithm moves to it and repeats the process with the new adjacent positions. The process ends when no better solutions are found, assuming that the current position is the optimal. Of course, local minima can be reached, and to avoid these most of the algorithms have certain backtracking abilities. Anyway, no optimization algorithm based on heuristics can guarantee that the optimal solution will be reached at the end.
   (b) *Estimator*: it is in charge of measuring the error rate of each partial design $\{t_j\}$, $\{n_K\}$, in order to verify if the reliability constraints are met. In Ref. [8], a simulation tool designed by ESA, called SST [13] is used.
   (c) *Cost function:* it is the function that the optimization engine tries to minimize. The 'solution quality' is measured, taking into account the two factors of the optimization problem: area and reliability:

$$F = \#\text{registers} + \alpha_1 \cdot \max(\text{ER}_1 - r_1, 0) + \alpha_2 \cdot \max(\text{ER}_2 - r_2, 0) + \cdots + \alpha_n \cdot \max(\text{ER}_n - r_n, 0) \quad (1)$$

- *Area cost:* represented by the number of registers of the system, #registers = $(3t_j + n_k)$.
- *ER of the outputs:* $ER_s - [r_s]$ represents the difference between the Error Rate and the specified reliability constraint for each output.

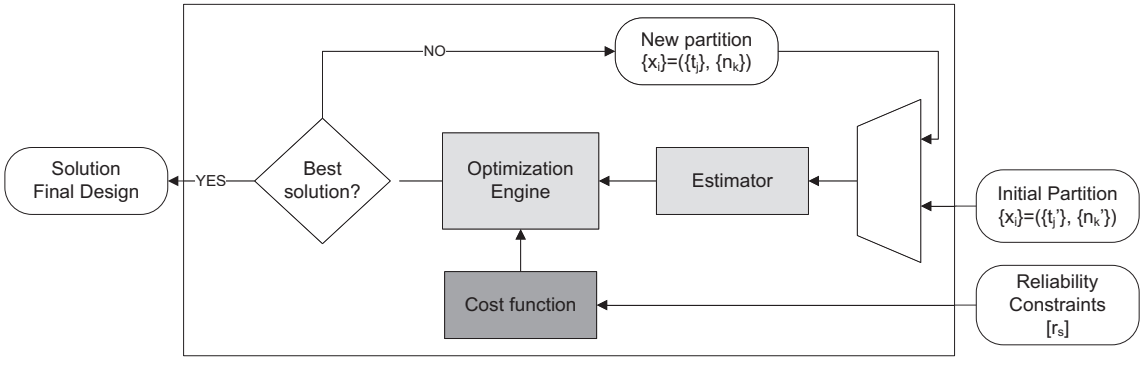

Fig. 1. Classical optimization flow chart.

- The α_s coefficients are large enough predefined values, so that, if one or more outputs do not meet the ER (i.e., ER_s $[r_s] > 0$), this factor dominates over the area cost and it is then minimized by the algorithm. When this constraint is met, then this factor is 0 and it does not contribute to the equation. Only when all the outputs meet $[r_s]$, then F = #registers, and the optimization goal consists in minimizing the cost.

However, this kind of optimization systems usually presents a common problem: both the complexity of the target design in terms of number of registers and the time consumed by the estimator can negatively affect the methodology performance, making the computation time unfeasible. Although in recent works improvements on the estimator performance have been undertaken through several methods, like the use of checkpoints and parallelism [14], the fact is that these efforts are insufficient.

For that reason, a novel solution is presented in this paper, which consists in reducing as much as possible the number of iterations required by the optimization engine. This is achieved through the fast selection of a good quality initial partition, $\{t'_j\}$ and $\{n'_k\}$, close enough to the optimal solution $\{t_j\}$ and $\{n_k\}$, so that the optimization algorithm needs to perform a low number of movements.

3. Selection of a quality initial partition

In order to select an appropriate initial partition, a model will be proposed that determines the probability that each register x_i is tripled in the optimal solution. If the model is accurate, few differences should be expected with the optimal design, thus needing little time to reach it. The methodology can be divided into two main steps:

- Calculation of the TMR Probability Index (TPI). This index indicates, for each register, the probability that it is tripled in the optimal solution. The TPI index will be calculated upon a set of topological criteria of the circuit, e.g., the distance of each node to the outputs or the number of loops. An important point is that these criteria are static, which means that they can be calculated in advance (before the optimization process begins) with a quick analysis of the circuit graph. This is fundamental in order to keep the complexity of the algorithm as low as possible.
- Calculation of the TMR Threshold. The previously calculated TPI will order the list of nodes $\{x_i\}$, with those that are most likely to be tripled at the top of it. Now, it has to be determined how many of these nodes will be actually tripled in $\{t_j\}$ and which will remain in $\{n_j\}$. In other words, a threshold has to be calculated that will split the ordered list in the two subsets of registers (tripled and non-tripled). This threshold will depend on the overall reliability demanded to the circuit. A reliability constraint close to 100% will force most of the nodes to be tripled, while a lower reliability will allow more nodes to remain non-tripled.

In the following sections, a model to order $\{x_i\}$ and determine the threshold that would create $\{t_j\}$ and $\{n_j\}$ will be explained. The presented model and the associated topological criteria have been derived based on the study of a high number of circuits. The results obtained on some of these circuits will be offered in further sections.

4. The TMR probability index

In order to calculate the TMR Probability Index (TPI) for each node, the following factors have to be taken into account:

- Trn_i (Transmission): This represents how likely it is that an SEU that has hit node i affects a primary output of the circuit.
- Msk_i (Masking): This indicates how likely it is that a SEU that has hit node i is masked before reaching an output.
- ro_i (Reliability output): This represents the reliability constraint of the output affected by node i. The more critical an output is, the more likely that the nodes in its critical path are triplicated. If a node affects several outputs, the reliability of the most critical is used.

The first and third factors will increase the probability of triplicating node i, while the second one would decrease it. Based on the conducted experiments, the following simple relation is proposed:

$$TPI_i = \frac{Trn_i}{Msk_i} \cdot ro_i \tag{2}$$

In the following, the two main factors to determine the TPI, Trn_i and Msk_i, will be explained. For the sake of simplicity, Msk_i will be addressed first.

4.1. Masking factor (Msk_i)

This factor models how likely it is that the circuit masks an SEU in node i before it affects one of the outputs. The main contributing factor for masking is combinational logic. If an SEU hits a node very far from an output, it will probably have to traverse several combinational stages. The probability that this SEU is masked in this situation is greater than in the case in which the SEU affects a node very close to an output. Another criterion that determines the degree of masking is the combinational logic itself. For example, in this way, a sea of AND gates will mask a much greater percentage of SEUs than XOR gates.

In the following, the set of topological criteria that favor error masking will be proposed. These criteria will be later combined to model error masking.

4.1.1. Topological criteria that favor error masking

- *C1* (*Proximity to the outputs*): This parameter refers to the proximity of the evaluated node to the circuit outputs. Depending on the distance to the output, an SEU can have a lower or higher probability to be propagated due to the masking logic [16].
- *C2* (*Masking logic*): This parameter evaluates the combinational logic between nodes and the capability of masking [20]. It is calculated as the inverse of the transmission probability of the logic, which depends on the inherent behavior of the combinational gates and modules. For example, an AND gate would have a 0.50 probability of transmitting an error in one of its inputs, depending on the value of the other input (a '0' would mask the error and a '1' would transmit it). In this case, the $C2$ parameter for the AND gate would be 1/0.50 = 2, which is the masking capability. On the other hand, an XOR gate will transmit any error in one of its inputs (100% probability), and therefore, the masking factor would be 1. The more likely that a gate or module filters errors, the higher the masking probability. In Fig. 2, an example is depicted, in which the propagation probability is shown for a simple circuit. In Table 1, the masking probabilities for the most common types of gates are shown.
- *C3* (*Number of inputs of successors*): A high number of inputs to the successors of a given node would indicate a high combinational activity (i.e. a lot of gates before the node), which would tend to mask more errors traversing through it. Therefore, this criterion can be considered as a measure of the error masking probability between a node and its successors.

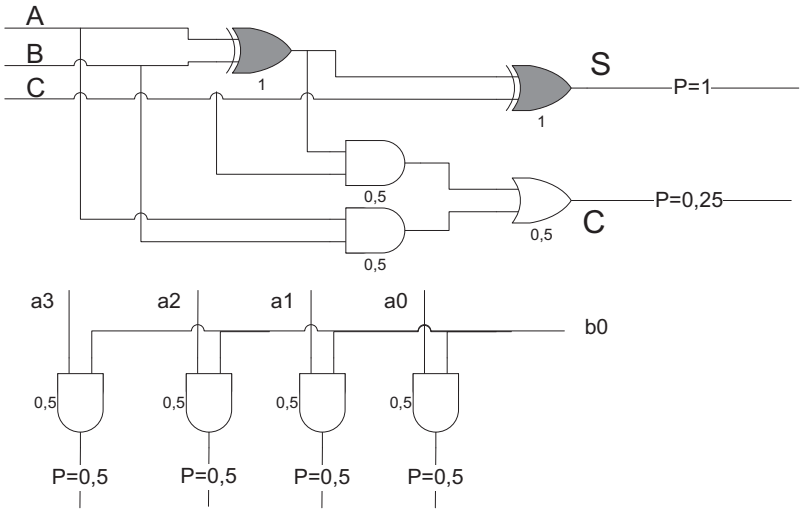

Fig. 2. Full adder: S is more sensitive than C (top); a 4 × 1 multiplier is less sensitive (Bottom).

Table 1
Masking probabilities for some common gates.

	Transmission (%)	*C*2
AND	50	2
OR	50	2
NOT	100	1
XOR	100	1
NAND	50	2
NOR	50	2

4.1.2. Models for error masking

Let us present a detailed model to represent error masking. This model consists in actually examining the combinational network between each pair of nodes, thus calculating the true masking capability of the circuit. This is represented by criterion *C*2. In this way, Msk_i would be calculated as the accumulated masking probability of all nodes that are in the path from node *i* to the output. If several outputs are affected by node *i*, then the one with worse masking would be selected. Notice that this value implicitly depends on *C*1, since the further the node is from the output, the more combinational stages will have to traverse.

$$Msk_i = \min\left(\prod_{(i->\text{output})\text{path}}^{C1} C2\right) \tag{3}$$

The previous model is detailed, but it requires too much computation time. Since one of the goals is to provide an initial partition in a fast way, it would be desirable to have a quicker model. One of the simplest approaches, which has proven well in the experiments, is to consider that each combinational stage will mask SEUs with a probability of 50%. Although this is not true per se, it is a statistical approximation providing that circuits usually contain a great number of combinational elements. Then, the overall masking of node *i*, would depend on its distance to the output: it would accumulate a 50% probability ($C2 = 1/0.50 = 2$) of masking SEUs per combinational stage. This happens to be criterion *C*1 explained before. In this way:

$$Msk_i = C2^{C1} = 2^{C1} \tag{4}$$

Expression (3) presents a detailed model, but needing too much computation time. On the other hand, expression (4) suggests a simple but very generic model (especially if the number of combinational stages is low).

In order to find out an intermediate alternative, a trade-off solution will be presented next. The idea is to measure the masking capability of a combinational network based on the number of inputs that it has (see [16]). More inputs would indicate more computation, and therefore more masking capability. In this way, Msk_i would be determined by the number of inputs that the successors of *i* have (which is represented by topological criterion *C*3). The greater this value is, the more likely that an SEU that has affected node *i* is masked, and therefore, not reaching any output. In this way, the path with minimal masking from node *i* to any output is calculated. This is achieved with a variation of Dijkstra's algorithm [15], in which the sum of all inputs in the path from node *i* to any output is minimized. Therefore, the obtained path will be the one in which its nodes have the minimal number of inputs, thus choosing a worst case from the masking point of view. In this way,

$$Msk_i = \min\left(\sum_{(i->\text{output})\text{path}}^{C1} C3\right) \tag{5}$$

This approach has been the one used in the rest of paper, since the trade-off accuracy vs. computation time has been the best in the conducted experiments. Anyway, the whole model in this paper could be used with approaches (3) or (4) if desired.

An important point to understand the rest of the model is that expressions (3)–(5) are equivalent and work well, but obviously they do not offer the same value. This is not a problem, because what we are looking for is not an absolute value to measure masking, but a relative value that can be used as an ordering criterion. In this way, if the masking probability of node *i* is higher than the masking probability of node *j*, then this has to be reflected by the model regardless of the expression used. In other words, the model does not need to be precise, but faithful.

4.2. Transmission factor (Trn_i)

In the previous subsection, the criteria that improve error masking have been discussed, which decrease the probability that a given node is finally triplicated. In this subsection, the criteria that favor error transmission will be detailed.

4.2.1. Topological criteria that favor error transmission

- $C4$ (*Influence on circuit outputs*): This parameter refers to the number of circuit outputs which are affected by a certain node. An SEU in a register that belongs to more than one critical path has a higher probability to propagate the error, increasing its observability at the output.
- $C5$ (*Number* of immediate successors): This refers to the number of successors that a node has. An SEU in a node with a lot of successors is more likely to reach an output.
- $C6$ (*Number* of loops): This parameter refers to the number of loops that are affected by a node. A loop is affected by a node if that node is part of the loop, or the loop is in a path from the node to any of the circuit outputs. Since loops feed back information, they tend to keep SEUs active in the circuit during more time.
- $C7$ (*Length* of loop): This criterion is complementary to $C6$, and indicates how many nodes belong to a certain loop. If a SEU affects a loop with few nodes, the feedback frequency of the loop will be high, thus circulating SEUs quicker. On the other hand, longer loops (with more nodes) would have a lower frequency of SEU distribution. Therefore, the shorter a loop is, the higher the value of this criterion.

4.2.2. Model for error transmission

We have identified three factors that increase the probability of error transmission, and therefore, that makes a node i more favorable for triplication:

- Number of circuit outputs that are directly or indirectly connected to node i. This is obvious, since nodes that are more connected to outputs would affect the behavior of the circuit easily. This factor corresponds to criterion $C4$.
- Number of immediate successors of node i. If a node has a lot of successors, errors on it would be transmitted through more paths, thus increasing the probability that a circuit output is affected. This factor corresponds to criterion $C5$.
- Loops influenced by node i. This is unarguably the most important factor on error transmission. An error that enters a loop may persist a high number of cycles in the circuit, sometimes until the system is reset. This is a factor that is affected by some of the previously explained criteria. In general, the criteria that would reinforce the negative effects of loops would be $C6$ (number of loops) and $C7$ (length of the loop; the shorter the loop is, the more negative effect produces). The expression that models the effect of loops is the following:

$$loops = C6 + C7 \tag{6}$$

In this way, and considering the previous analysis, the model for error transmission that has been used is the following:

$$Trn_i = (C4 + C5) + loops = (C4 + C5) + (C6 + C7) \tag{7}$$

4.3. Final TPI model

According to the previous expressions, the TPI model (2) can be rewritten as:

$$TPI_i = \frac{(C4 + C5) + (C6 + C7)}{\min\left(\prod_{(i->\text{output})\text{path}}^{C1} C2\right)} \cdot ro_i \tag{8}$$

If less computational time is desired, at the expense of precision, the following alternative can be used, based on expression (5):

$$\text{TPI}_i = \frac{(C4 + C5) + (C6 + C7)}{\min\left(\sum_{(i->\text{output})\text{path}}^{C1} C3\right)} \cdot ro_i \tag{9}$$

The parameter ro_i was defined as the reliability constraint of the most critical output affected by node i.

Once the TPI index for each node is determined, the list of nodes in the system is ordered based on this parameter. Those nodes with a higher probability of being tripled will be located at the top of the list, and those with lower probability, at the bottom. Now, a threshold has to be determined in order to divide the list of nodes into $\{t'_j\}$ and $\{n'_j\}$. If the list has been properly ordered (with the TPI) and properly divided (with this threshold), the partition suggested by this technique will be very similar to the optimal solution. Therefore, the optimization engine will have to perform few movements to reach the optimal, thus reducing the design time a lot, compared to the scenario where a trivial initial partition is chosen.

5. The TMR threshold

It is important to determine the number of registers that need to be tripled. If this number is not accurate in the initial partition, more movements will have to be performed by the optimization algorithm.

The model to calculate the TMR threshold that we have used is based on the following parameters:

- The reliability constraints, [ro]. As defined before, this is the error tolerance allowed in the circuit outputs. A reliability constraint of 100% would indicate that no errors are tolerated at the output. This is an extreme case which would lead to the triplication of all the registers in the system. However, if this constraint is lower than 100%, a certain number of errors is allowed at the output, what can be used to leave some registers without triplication. The lower the reliability constraint is, the fewer registers to be tripled.
- The error fluence, λ. Let us define this parameter as the number of errors that happen in the system per register in a certain amount of time (let us define this time as T). This models the environment where the circuit is going to work and it is similar to the classic definition of fluence. In this classic definition, fluence is the number of particles that hit the circuit per unit of area. The difference with our model is that we do not use particles but errors (not all particles that hit the circuit produce errors; see a later comment for a detailed explanation), and we approximate the area with the number of registers. If an error happens on the combinational logic, then an SET maybe be produced, which is a problem different from SEUs. The higher the value of λ is, the more registers to be tripled.
- The masking effect of the circuit. Not all the errors that are produced in the registers are propagated to the outputs. The masking factor of the combinational logic is important, as explained previously. The average of the masking factors, avg(Msk_i), has been used to model this case. Expressions (3), (4) or (5) can be used to determine this.

5.1. Calculation of the threshold when λ/T is high

Let us start with the situation in which there are many errors, and besides they come close in time. This means that λ/T is high (or in other words, the error density is high). The reason to start with this scenario is because it is easier to model, what will be explained later.

To determine the threshold of nodes that have to be tripled, the following condition is imposed: the number of cycles in which an

erroneous output is obtained, *total_err*, must be, at most, the one indicated by the reliability constraint, *allowed_err*.

Let us first assume that the number of errors that are produced in the circuit, if no registers are tripled, is $\lambda \cdot N$, per the definition above (being N the number of registers). But only those that reach an output will count for the total number of errors. Statistically, a fraction of the errors will be masked by the circuit, according to the previously defined factor $\text{avg}(Msk_i)$. Therefore, the effective number of errors at the output will be $(\lambda \cdot N)/\text{avg}(Msk_i)$. However if some of these registers are tripled, the number of effective errors will decrease, since those hitting the protected registers will not propagate. In other words, the effective number of errors will be proportional to the number of unprotected registers. Therefore, the number of errors will be

$$total\_err = \frac{\lambda \cdot N_U}{\text{avg}(Msk_i)} \tag{10}$$

where N_U is the number of non-tripled registers.

The number of erroneous cycles at output i that is allowed by the reliability constraint is determined by $[1 - ro_i] \cdot T$, being T the exposure time determined by the fluence λ expressed in clock cycles. Since circuits may have several outputs with different reliability constraints, the most restrictive is chosen to calculate the threshold:

$$allowed\_err = [1 - \max(ro_i)] \cdot T \tag{11}$$

Then, per the previous condition that the number of errors at the output cannot exceed the error tolerance defined by the reliability constraint, both expressions (10) and (11) must meet the following condition:

$$\frac{\lambda \cdot N_U}{\text{avg}(Msk_i)} \leqslant [1 - \max(ro_i)] \cdot T \tag{12}$$

And therefore, the number of registers that may remain non-tripled, while the constraints are still met is:

$$N_U \leqslant \frac{[1 - \max(ro_i)] \cdot \text{avg}(Msk_i) \cdot T}{\lambda} \tag{13}$$

Obviously, the number of registers to remain non-tripled must be lower than the total number of registers, $N_U \leqslant N$. Therefore, this condition together with (13) leads to the following expression:

$$N_U = \min\left(N, \frac{[1 - \max(ro_i)] \cdot \text{avg}(Msk_i) \cdot T}{\lambda}\right) \tag{14}$$

Since N_U must be a natural number, it is rounded down if necessary. This value represents the number of registers that may remain non-tripled, and the threshold to partition the list ordered with the TPI criterion. Therefore, the N_U registers that are in the bottom of the list will form the $\{n'_k\}$ partition, and the rest will go to the $\{t'_j\}$ partition.

Of course, this threshold does not need to exactly determine the optimal $\{n_k\}$ and $\{t_j\}$, since in that case the procedure would be an optimization algorithm itself, something that it is not intended. But the partition will be good enough so that few movements will be needed to reach the optimal in the subsequent optimization process.

After presenting the TMR threshold model, some important considerations must be taken into account:

- As commented before, we have not used the classic definition of fluence, since we are not tackling the problem from the physical point of view. Classically, the fluence is the number of particles that hit the circuit per unit of area, integrated to a certain time. In this paper, since we are working at the logical level and therefore physical phenomena are transparent, we are counting the number of errors that are effectively produced. There is a conversion rate from particle hits to induced errors, which is also the standard parameter of cross-section, σ. In this way, this simple relation can be applied, where λ' is the classical fluence related to the number of particles:

$$\sigma = \frac{\lambda}{\lambda'} \tag{15}$$

- Expression (13) can be also defined in terms of the flux ϕ instead of the fluence, in order to make it independent of T. Considering the relation between fluence and flux:

$$\lambda = \int_0^T \phi \cdot dt \tag{16}$$

If we assume a constant flux during the exposure period, then:

$$\lambda = \phi \cdot T \Rightarrow \phi = \frac{\lambda}{T} \tag{17}$$

Therefore, expression (13) can be redefined as (only if ϕ is a constant):

$$N_U \leqslant \frac{[1 - \max(ro_i)] \cdot \text{avg}(Msk_i)}{\phi} \tag{18}$$

- In order to calculate the total number of errors that will reach the output, we have considered that each error that affects a register produces only one wrong cycle if it is not masked and reaches an output. This may seem simplistic, especially when loops have been considered a negative factor for reliability because they feed back errors, making them persistent in time, and therefore producing several wrong cycles. Initially, we modeled the threshold calculation taking loops into account, and penalizing errors hitting those loops with more wrong cycles. However, we experimentally noticed that this approach provided worse results than the presented one. Studying why a supposedly more refined model did not work better, we found the following explanations:
  i. *Error accumulation problem*. This effect appears when working with high values of λ/T, which is the worst scenario for reliability (this is the reason why we have studied this scenario first). In this case errors are induced in the circuit with a separation of few cycles and many of them will coincide in time inside the affected loops. This makes that the effects of several consecutive errors overlap in time, sharing the wrong outputs. Let us consider a simple example, in which 20 errors hit a given loop in consecutive cycles, producing each error 10 wrong cycles at the output. Then, the total number of wrong cycles will not be 200. Many of them will coincide in time, contributing at the same time to the same wrong output cycle. In this simplistic case, the wrong cycles will be 10 (produced by the first error to arrive) plus 19 (one per each of the remaining errors, which would overlap most of their effect).
  ii. *Register ordering*. The calculation of the threshold is oriented to divide a list that has been previously ordered. If the TPI index ordering has been effective, most of the registers that form part of a loop will be located at the top of that list. This means that if λ/T is high, the number of registers selected for triplication will be presumably high too. Therefore, those registers affecting loops are very likely to be selected for this. The effect of multiple wrong output cycles per error only happens for those registers in loops that are unprotected. Since very few of them will meet this condition (because they probably have a high TPI index), this effect is negligible. This reason together with the previous one, are the explanation of why considering that each error would produce only one wrong output cycle (if not masked) works well in the model for high values of λ/T.

5.2. Calculation of the threshold for lower values of λ/T

When there are fewer errors in the system, and they come more separate in time, some of the statements in the previous subsection do not apply. Particularly, the idea of error accumulation is not true. Let us imagine a situation in which only one error is produced, affecting a register in a loop. Due to the effect of the loop, this error may persist many cycles in the system and produce many wrong outputs. Then, the total number of wrong outputs will not be correctly modeled by (10), as this expression assumed one wrong output by cycle. Since the value given by (10) will be much smaller than the real value, the number of non-tripled nodes as per (13) will be much higher than the right value. As a conclusion, very few registers will be chosen for triplication, and eventually leaving some of the registers in loops without protection.

Let us study the scenario in which the previous model will not properly work, i.e., with a low density of errors λ/T. The simplest way to study this scenario is to separately consider the registers involved in loops and those not involved in loops.

Let us introduce the following notation:

- N^{loops}: The number of all registers inside loops.
- N_U^{loops}: The number of registers inside loops that may remain unprotected and still meet the reliability constraints.
- N^{no_loops}: The number of all registers outside loops.
- $N_U^{no\_loops}$: The number of registers outside loops that may remain unprotected and still meet the reliability constraints.

In this way, the total number of non-tripled (unprotected) registers may be divided into:

$$N_U = N_U^{no\_loops} + N_U^{loops} \tag{19}$$

where the first term is the number of unprotected registers that do not affect loops and the second term is the number of unprotected registers that do affect any loop.

A single error in the former will still produce only one wrong output cycle. However, an error in the latter will eventually produce many wrong output cycles. If we define the average number of wrong output cycles as K (i.e., the error persistence), then (10) can be rewritten as:

$$total\_err = \frac{\lambda \cdot \left(N_U^{no\_loops} + K \cdot N_U^{loops}\right)}{\text{avg}(Msk_i)} \tag{20}$$

And (13) would be:

$$\left(N_U^{no\_loops} + K \cdot N_U^{loops}\right) \leqslant \frac{[1 - \max(ro_i)] \cdot \text{avg}(Msk_i) \cdot T}{\lambda} \tag{21}$$

K should be estimated in each circuit, since it depends on the loop structure. Anyway, let us assume it is a quite large value, because otherwise the initial model in (13) would not be significantly affected.

In this situation,

$$N_U^{no\_loops} \ll K \cdot N_U^{loops} \tag{22}$$

And therefore,

$$N_U^{loops} \leqslant \frac{[1 - \max(ro_i)] \cdot \text{avg}(Msk_i) \cdot T}{K \cdot \lambda} \tag{23}$$

The greater K is, the higher the effect of loops will be, and therefore the number of unprotected registers in loops will have to be lower. Depending on this, two scenarios can be identified.

- K is not large enough to make the right term of (23) lower than 1. In this case the number of unprotected loop registers is directly determined with (23). And per expression (22), all the registers outside loops will remain unprotected. So, in this case, the threshold will be:

  $$N_U = N_U^{loops} + N^{no\_loops} \tag{24}$$

  In other words, the registers that may remain unprotected will be all the registers outside loops and a part of the registers affecting loops.
- K is large enough to make the right term of (23) lower than 1. Then, per expression (23), no registers in loops will remain unprotected ($N_U^{loops} = 0$). In this situation expression (22) does not hold, since the right term will be zero, and expression (21) will then be:

  $$N_U^{no\_loops} \leqslant \frac{[1 - \max(ro_i)] \cdot \text{avg}(Msk_i) \cdot T}{\lambda} \tag{25}$$

  This expression will be equivalent to (13), and will give the number of nodes outside loops that can remain unprotected. So, in this case, the threshold will be:

  $$N_U = N_U^{no\_loops} \tag{24}$$

  In other words, the registers that may remain unprotected will be a part of the registers outside loops, but none of the registers affecting loops.

As a conclusion, an easy way to handle the scenarios in which λ/T is low, is to separate the nodes outside loops from the nodes in loops. The condition of λ/T being low may be ambiguous for intermediate cases. However, this is not a problem since expression (13) is a particular case of (21) for $K = 1$ (error saturation). Therefore, the latter may be used always, with the correct value of K.

Determining K, i.e., the average number of cycles that an error will persist, is not straightforward. Approximations based on behavioral profiling of the circuit may be used. This will be further studied in future works.

6. Experimental results

6.1. Example of the model

Before presenting the experimental results, the TPI model discussed in previous sections will be shown with an example (see Fig. 3). A generic circuit (the combinational logic is not depicted for simplicity) has been used in order to demonstrate this model.

The objectives of this example are the following:

(1) To show that calculating a smart initial partition with the model presented in this paper can speed up the optimization process, reaching the optimal in less time than in the case of starting with a default partition. In this example, our smart initial partition will be compared with an all-TMR initial partition (in the next subsection, our initial partition will be compared with a random initial partition, in order to make the experiments more general). So, this objective is oriented to show improvement in the design time.

(2) To show that it is worth using an optimization process to calculate an optimal selective TMR solution, instead of applying the standard full TMR solution. Of course, the latter will need less design time, but the former can produce substantial area savings. Besides, the extra design time of the former can be mitigated with our initial partition model, as proved in the previous objective. So, this objective is oriented to show a benefit in the area cost.

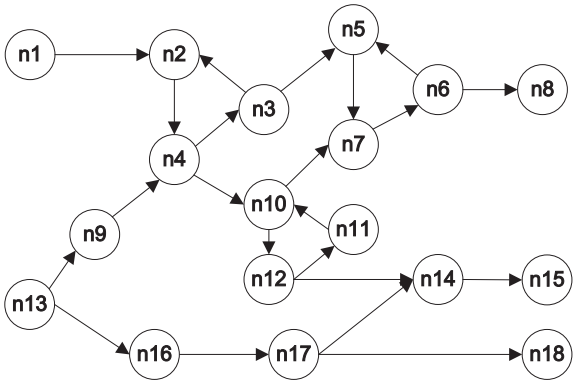

Fig. 3. Generic circuit in graph representation.

So, a comparison between the initial partition calculated through the TPI model and a default one will be carried out next. In this case, the default partition starts with all nodes triplicated in $\{t'_j\}$. This choice is reasonable, since a very high reliability constraint (very low accepted error rate) has been chosen ($ro = 99\%$) for all outputs, and the optimal design will presumably have most registers with TMR.

Other parameters for this experiment are $\lambda = 1000/18$ and $T = 1000$ cycles.

Initially, using the TPI model on the circuit, the methodology orders the list of registers in this way (first nodes have a higher probability of being triplicated):

$$\text{Sorted list} = \{6, 7, 13, 18, 15, 8, 17, 16, 14, 12, 10, 5, 4, 11, 2, 3, 9, 1\}$$

The TPI index for each register is shown graphically in Fig. 4.

Next, applying (14), the TMR threshold has been calculated. Expression (14) can be used since in this case $\lambda/T = 1/N$, which is a high error density. The result is that the threshold of unprotected nodes, N_U, is 5. So, the last 5 nodes in the TPI list will go to $\{n'_k\}$, and the other 13 (the first ones) will go to $\{t'_j\}$.

Therefore, according to this threshold, the initial partition suggested by the TPI model for this example is:

Table 2
Performance comparison between the two initial partitions.

Initial partition	Initial tripled registers $\{t'_j\}$	Initial non-tripled registers $\{n'_k\}$	Number of iterations to the optimal
Proposed (TPI)	18-17-16-15-14-13-12-10-8-7-6-5-4	11-9-3-2-1	**112**
Full TMR	All	$\varnothing$	**168**

$$\{t'_j\}\text{TPI} = (6, 7, 13, 18, 15, 8, 17, 16, 14, 12, 10, 5, 4)$$
$$\{n'_k\}\text{TPI} = (11, 2, 3, 9, 1)$$

If, alternatively, the default initial partition previously defined is used (which assumes that all nodes will be implemented with TMR), then:

$$\{t'_j\}\text{TMR} = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18)$$
$$\{n'_k\}\text{TMR} = (\varnothing)$$

After performing the optimization process, the optimal solution in this case happens to be:

$$\{t_j\} = (6, 7, 8, 10, 12, 13, 14, 15, 17, 18)$$
$$\{n_k\} = (1, 2, 3, 4, 5, 9, 11, 16)$$

It can be clearly seen that the initial partition determined by the TPI methodology is closer to the optimal solution than the default initial partition (all-TMR in this case).

The consequence of this is that the number of iterations needed by the optimization engine using TPI is clearly reduced, as well as the computation time. In Table 2, the performance results are summarized for the two initial partitions.

A performance improvement can be seen with the use of TPI, which needs 112 iterations compared with the 168 of the default initial partition, which represents an improvement of 33.3%. Therefore, the initial objective of reducing the computation time of the optimization engine has been achieved. Besides, it has been detected that the time needed to determine the initial partition (graph analysis, criteria calculation and decision based on the TPI index) is negligible (around 3 s) compared to the overall optimization time (more than 3 h).

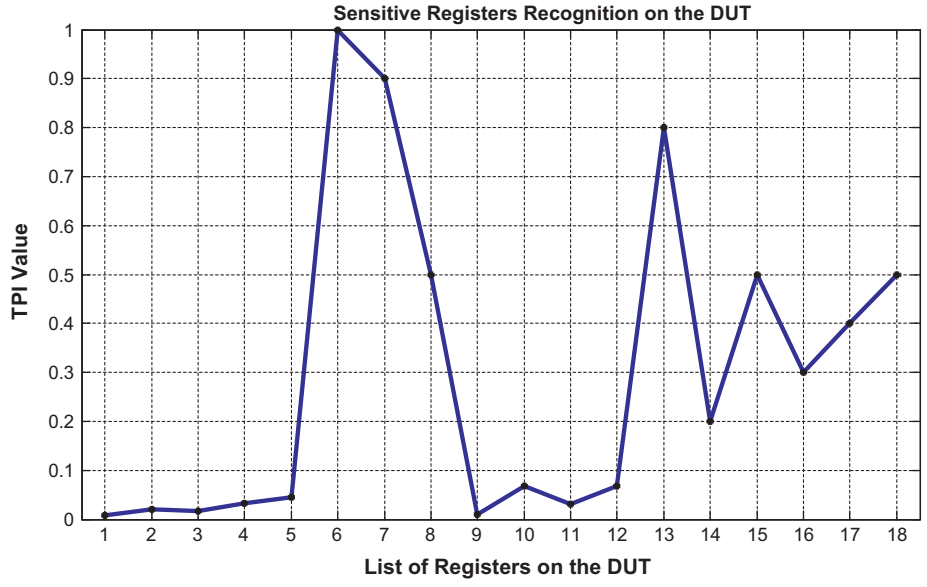

Fig. 4. TPI index for all the registers.

Tables 3–12
Experimental results. IT: iterations to reach solution and Ti: number of tripled registers in the solution.

	99.9% it-t_i	99% it-t_i	95% it-t_i	90% it-t_i	85% it-t_i	75% it-t_i	65% it-t_i	50% it-t_i
Circuit 1 reliability								
λ = 100								
TPI model	112-13	112-10	85-3	19-0	19-0	19-0	19-0	19-0
Random	112-13	168-10	**112-4**	19-0	19-0	19-0	19-0	19-0
λ = 500								
TPI model	112-13	112-13	85-11	99-6	19-6	19-0	19-0	19-0
Random	112-13	112-13	124-11	**124-9**	135-6	**145-1**	124-0	19-0
λ = 1000								
TPI model	99-14	112-13	99-12	85-11	85-9	70-6	54-2	19-0
Random	99-14	112-13	124-12	135-11	124-9	145-6	154-2	154-0
Circuit 2 reliability								
λ = 100								
TPI model	60-19	78-16	60-5	21-0	21-0	21-0	21-0	21-0
Random	60-19	111-16	**78-8**	21-0	21-0	21-0	21-0	21-0
λ = 500								
TPI model	21-20	60-19	95-15	60-13	78-9	60-1	21-0	21-0
Random	21-20	60-19	**111-16**	**95-15**	**95-13**	**140-4**	140-0	21-0
λ = 1000								
TPI model	21-0	21-0	95-17	95-15	60-13	60-8	95-2	21-0
Random	21-0	21-0	111-17	**111-16**	**111-15**	**78-13**	**186-3**	177-0
Circuit 3 reliability								
λ = 100								
TPI model	78-18	111-14	111-4	60-1	21-0	21-0	21-0	21-0
Random	78-18	140-14	165-4	60-1	21-0	21-0	21-0	21-0
λ = 500								
TPI model	60-19	60-18	111-15	140-10	153-7	60-1	21-0	21-0
Random	60-19	78-18	126-15	165-10	176-7	195-1	140-1	21-0
λ = 1000								
TPI model	60-19	60-19	111-16	126,313	60-13	78-7	60-4	21-0
Random	60-19	60-19	126-16	153-13	176-13	186-7	176-4	186-0
Circuit 4 reliability								
λ = 100								
TPI model	126-15	126-13	140-6	60-1	21-0	21-0	21-0	21-0
Random	126-15	**111-14**	**140-8**	60-1	21-0	21-0	21-0	21-0
λ = 500								
TPI model	111-16	126-15	126-13	78-12	111-9	126-5	78-2	21-0
Random	111-16	126-15	126-13	153-12	165-9	**140-6**	111-2	21-0
λ = 1000								
TPI model	111-16	126-15	95-14	126-13	126-13	60-10	140-3	21-0
Random	111-16	126-15	**60-15**	**95-15**	**78-15**	**111-11**	186-3	186-0
Circuit 5 reliability								
λ = 100								
TPI model	85-14	85-7	19-0	19-0	19-0	19-0	19-0	19-0
Random	**85-15**	**145-9**	154-0	19-0	19-0	19-0	19-0	19-0
λ = 500								
TPI model	99-14	112-11	112-7	112-5	54-1	19-0	19-0	19-0
Random	99-14	231-11	**145-9**	162-5	**169-2**	154-0	124-0	19-0
λ = 1000								
TPI model	70-16	112-13	124-9	124-6	19-3	19-0	19-0	19-0
Random	70-16	112-13	154-9	169-6	124-3	184-0	175-0	154-0
Circuit 6 reliability								
λ = 100								
TPI model	121-6	91-5	16-0	16-0	16-0	16-0	16-0	16-0
Random	121-6	**211-6**	115-0	16-0	16-0	16-0	16-0	16-0
λ = 500								
TPI model	115-7	108-7	91-5	70-3	16-0	16-0	16-0	16-0
Random	115-7	115-7	121-5	**121-3**	126-0	112-0	91-0	16-0
λ = 1000								
TPI model	108-8	115-7	108-6	100-4	100-1	45-0	96-0	16-0
Random	108-8	115-7	121-6	126-4	133-1	133-0	126-0	115-0

Tables 3–12 (*continued*)

	99.9% $it\text{-}t_i$	99% $it\text{-}t_i$	95% $it\text{-}t_i$	90% $it\text{-}t_i$	85% $it\text{-}t_i$	75% $it\text{-}t_i$	65% $it\text{-}t_i$	50% $it\text{-}t_i$
Circuit 7 reliability								
$\lambda = 100$								
TPI model	75-13	98-6	17-0	17-0	17-0	17-0	17-0	17-0
Random	75-13	120-6	125-0	17-0	17-0	17-0	17-0	17-0
$\lambda = 500$								
TPI model	62-14	98-10	75-7	75-4	62-2	17-0	17-0	17-0
Random	62-14	**98-11**	**108-9**	**125-5**	138-2	**117-0**	98-0	17-0
$\lambda = 1000$								
TPI model	62-14	87-12	108-8	108-5	98-2	17-0	17-0	17-0
Random	62-14	87-12	125-8	**121-8**	**143-3**	147-0	143-0	125-0
Circuit 8 reliability								
$\lambda = 100$								
TPI model	490-28	145-0	31-0	31-0	31-0	31-0	31-0	31-0
Random	**495-30**	493-0	430-0	31-0	31-0	31-0	31-0	31-0
$\lambda = 500$								
TPI model	460-9	468-3	145-0	31-0	31-0	31-0	31-0	31-0
Random	460-9	496-3	493-0	481-0	460-0	391-0	286-0	31-0
$\lambda = 1000$								
TPI model	441-11	468-6	405-1	145-0	31-0	31-0	31-0	31-0
Random	441-11	468-8	**493-2**	493-0	464-0	475-0	451-0	31-0
Circuit 9 reliability								
$\lambda = 100$								
TPI model	126-14	95-11	21-0	21-0	21-0	21-0	21-0	21-0
Random	126-14	140-11	186-0	21-0	21-0	21-0	21-0	21-0
$\lambda = 500$								
TPI model	140-14	153-13	126-9	126-5	78-2	21-0	21-0	21-0
Random	140-14	153-13	165-9	195-5	203-2	186-0	140-0	21-0
$\lambda = 1000$								
TPI model	95-17	129-14	111-14	165-8	140-6	75-2	21-0	21-0
Random	95-17	140-14	111-14	**153-11**	**150-7**	250-2	210-0	186-0
Circuit 10 reliability								
$\lambda = 100$								
TPI model	46-10	46-4	13-0	13-0	13-0	13-0	13-0	13-0
Random	46-10	55-8	76-0	13-0	13-0	13-0	13-0	13-0
$\lambda = 500$								
TPI model	13-12	36-10	46-4	36-1	13-0	13-0	13-0	13-0
Random	13-12	46-10	**46-9**	88-1	88-0	76-0	63-0	13-0
$\lambda = 1000$								
TPI model	13-12	36-11	55-6	63-2	46-1	13-0	13-0	13-0
Random	13-12	36-11	76-6	**46-9**	**85-3**	88-0	85-0	76-0

Finally, the benefits of the Selective TMR technique are obvious. An area reduction of 29.6% has been achieved respect to tripling all the registers, with a loss in reliability of only 0.1% (99% vs. 100% of full TMR).

6.2. Experimental results

Finally, a set of experiments have been conducted in order to verify the quality of the model. Ten random circuits have been generated, with different number of nodes, depths and critical paths. Although ten representative examples have been chosen to illustrate the technique, the whole derivation process has required the analysis of hundreds of circuits, since it strongly depends on statistics. All these circuits have been randomly generated, but not arbitrarily. The whole generation process has been designed in a structured way, following similar principles to the ones described in [27]. For each circuit, several scenarios with different error fluence ($\lambda = 100$, 500 and 1000) and several reliability constraints (50–99.9%) have been studied. For each scenario, two different initial partitions have been tested: one calculated with the TPI model, and another one quasi-randomly generated. Then, the time (represented in number of cycles) needed to reach the final solution from each initial partition has been measured and compared. Also, the final solutions reached from each initial partition have been analyzed, in order to verify that the optimal one has been chosen.

The results for each of the ten circuits are reported in Tables 3–12.

Analyzing these results, it can be noticed that the selection of the initial partition through the TPI model reduces the number of iterations in most cases (92.08%), respect to starting with a random initial partition. This reinforces one of the goals of the model, which is the reduction of the design time.

Besides, it can be seen that starting from the TPI initial partition has led to the optimal solution always. On the other hand, starting with a random partition has led to a non-optimal solution in 7.92% of the experiments (in bold in the tables), which implies an unneeded area overhead.

In Fig. 5, a summary of the time and area improvement obtained in the whole set of experiments is depicted. Fig. 5 – Top shows the average saved area produced by using selective TMR instead of the standard full TMR. It can be seen, as expected, that lower reliabilities get more area savings. This is one of the principles of the presented model, which proposes an appropriate partition for each reliability. Also, the area savings are larger for lower fluence values

(it is easier to meet the reliability constraints with smaller error densities).

In Fig 5 – Bottom, a summary of the time needed to reach the optimal solution is shown. The time improvement has been calculated comparing the scenario in which the initial partition has been chosen through the TPI model versus a quasi-random initial partition. In order to understand the results, the details of this quasi-random partition need to be clarified. In this partition, the decision of how many nodes will start protected and how many will not (threshold), is not a random decision. This threshold is determined as a linear function of the reliability constraint: the higher this constraint, the more nodes are decided to protect. Then, once the threshold is determined, the decision of which nodes are tripled is purely random. The reason of determining the partition threshold in a coherent way is because otherwise the initial partitions are too bad, giving the TPI model an extra advantage. In this way, the results offered by the quasi-random partition are more competitive, measuring the true quality of the TPI model more effectively.

In Fig. 5 – Bottom, it can be seen that the initial partition through the TPI model gets better performance results for lower fluence values. An important observation is that for two of the three environments, the TPI model does not produce any time improvement respect to the quasi-random initial partition (for lower reliability constraints). This is because of the reason explained before: this partition has a certain degree of knowledge which makes it more efficient. A purely random initial partition would have offered worst performance results than the TPI model even for a lower reliability.

7. Case study

A real case study is going to be used to discuss the proposed technique and also to evaluate its effectiveness in a realistic design. The selected circuit in this case study is a Feed Forward Equalizer (FFE) [10,11], which is a special type of adaptive filter. The basic structure of the FFE is shown in Fig. 6. It is widely used in communication applications, and therefore needs a high level of fault tolerance.

The FFE is composed of three main blocks: (i) a traditional FIR filter with coefficients $h[i]$, input signal $x[n]$ and output signal $y[n]$; (ii) an adaptation logic that updates the values of the coefficients $h[i]$ periodically, according to the error $e[n]$; and (iii) a slicer

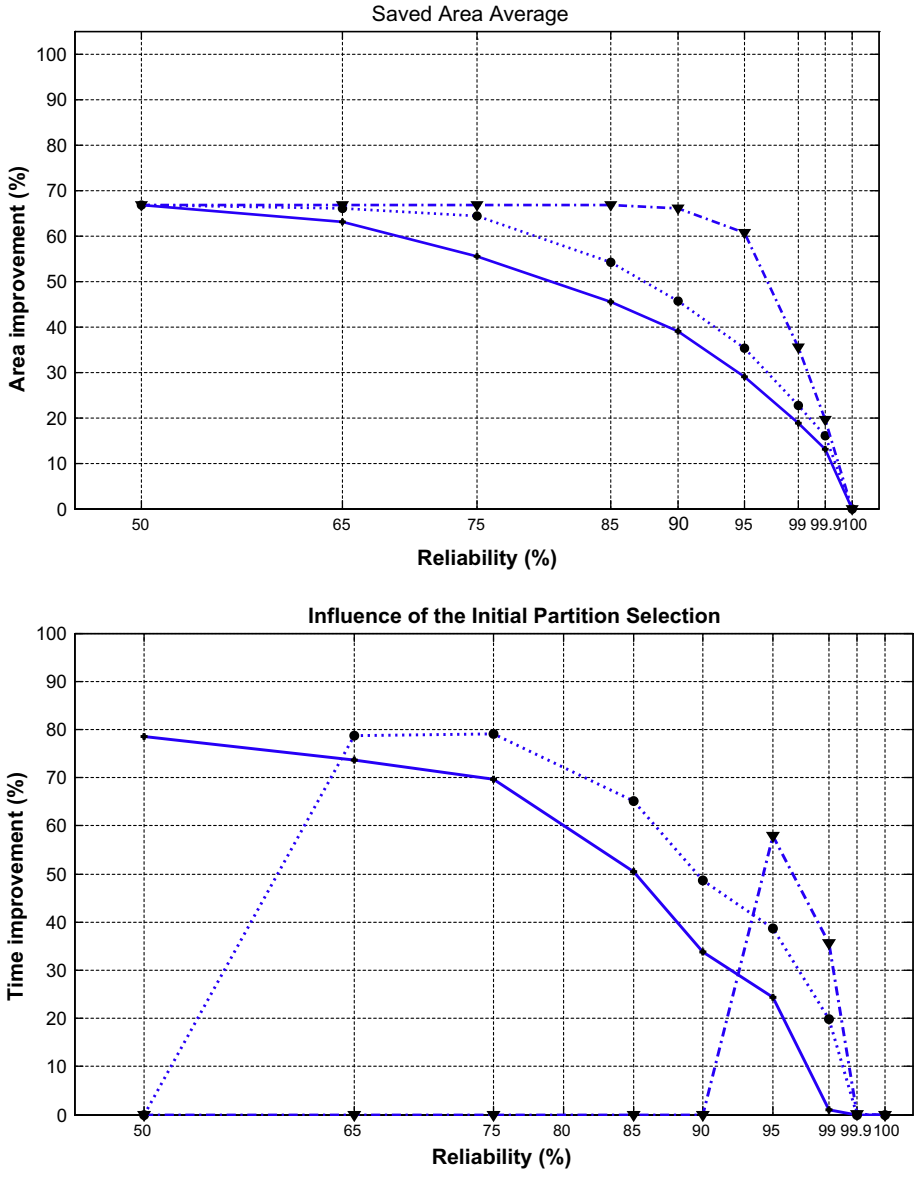

Fig. 5. Average saved area (Top); average performance improvement (Bottom).

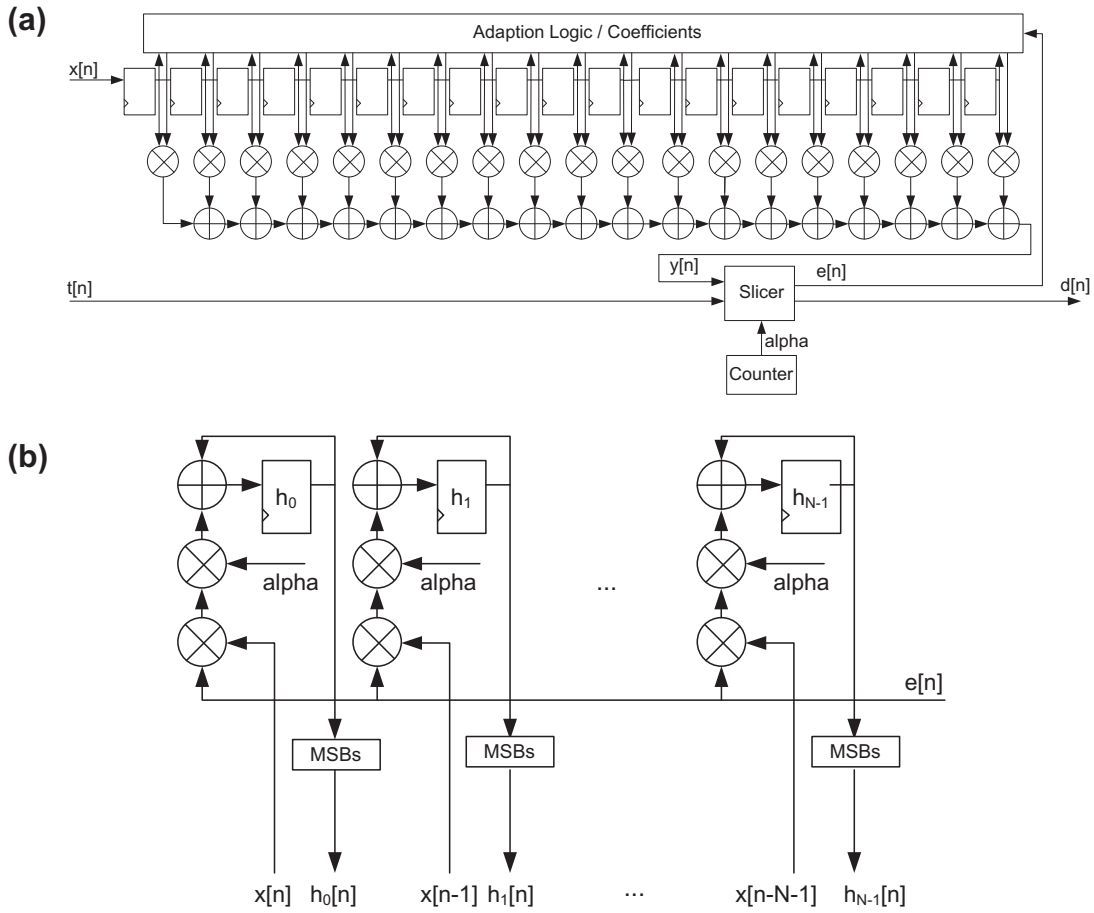

Fig. 6. Case study: (a) Feed Forward Equalizer structure. (b) Adaption logic.

unit estimating the received symbol $d[n]$ from the filter output $y[n]$.

The proposed methodology has been applied to this implementation of the FFE, with a set of reliability constraints $[ro] = 95\%$ (which implies an ER of 5%), fluence $\lambda = 100{,}000/32$ and time $T = 500{,}000$ cycles. The number of taps is equal to 16, what makes the number of registers $N = 32$. The study of the SEU sensitivity has considered the two main register sets of the circuit: the coefficients ($h[i]$) and the delay line ($x[n]$). In a similar way to the previous section, the optimization is carried out using two different initial partitions: one calculated with the TPI index and another one starting with all registers in TMR.

The initial TPI values obtained with the model can be seen in Fig. 7 – Top. Since the obtained threshold has been $N_U = 16$, that means that the 16 registers with lower TPI will remain unprotected (the delay line registers), and the rest will be tripled (the coefficient registers).

On the other hand, the optimal solution for this particular problem is depicted in Fig. 7 – Bottom. In this case, both designs coincide, meaning that the TPI analysis has been able to perfectly determine all the nodes that need to be tripled. However, this level of accuracy is not expected in most cases, since the purpose of the TPI methodology is to calculate a good initial partition, but not the optimal itself.

The numerical results can be seen in Table 2. Obviously, in this case, starting with the partition calculated though TPI results in a much lower number of iterations to reach the optimal solution, than using the all-TMR default partition (30 vs. 441, what represents a reduction of 93.2%).

Notice that although the initial partition determined by TPI coincides with the optimal solution, 30 iterations have been consumed by the optimization engine. These are the trials that the engine has performed looking for a better solution, without success (see Table 13).

Also, an area reduction of 50% has been achieved using Selective TMR compared to a full implementation of TMR (other area costs different from registers have been disregarded). The reliability of the circuit has only been reduced from 100% to 95% with this decision.

A final conclusion is that the decisions made by the TPI methodology to determine which nodes need to be tripled coincide with the knowledge that a signal processing expert would provide. If the behavior of the circuit is studied in depth [28], this would confirm that an SEU in the delay line would remain only until it is shifted out of the pipeline (in the worst case, in a number of cycles equal to the number of taps). However, an SEU in the coefficients will remain for a much longer time, because the filter would be destabilized, and although the adaptation algorithm would be able to recover it, this would consume a lot of cycles. Therefore, the decision of tripling the coefficients is correct, since an SEU on them would generate a great increment of the error rate.

8. Conclusions

In this paper, a novel methodology to reduce the computation time of a selective TMR optimization process has been presented. The main conclusions of the paper are:

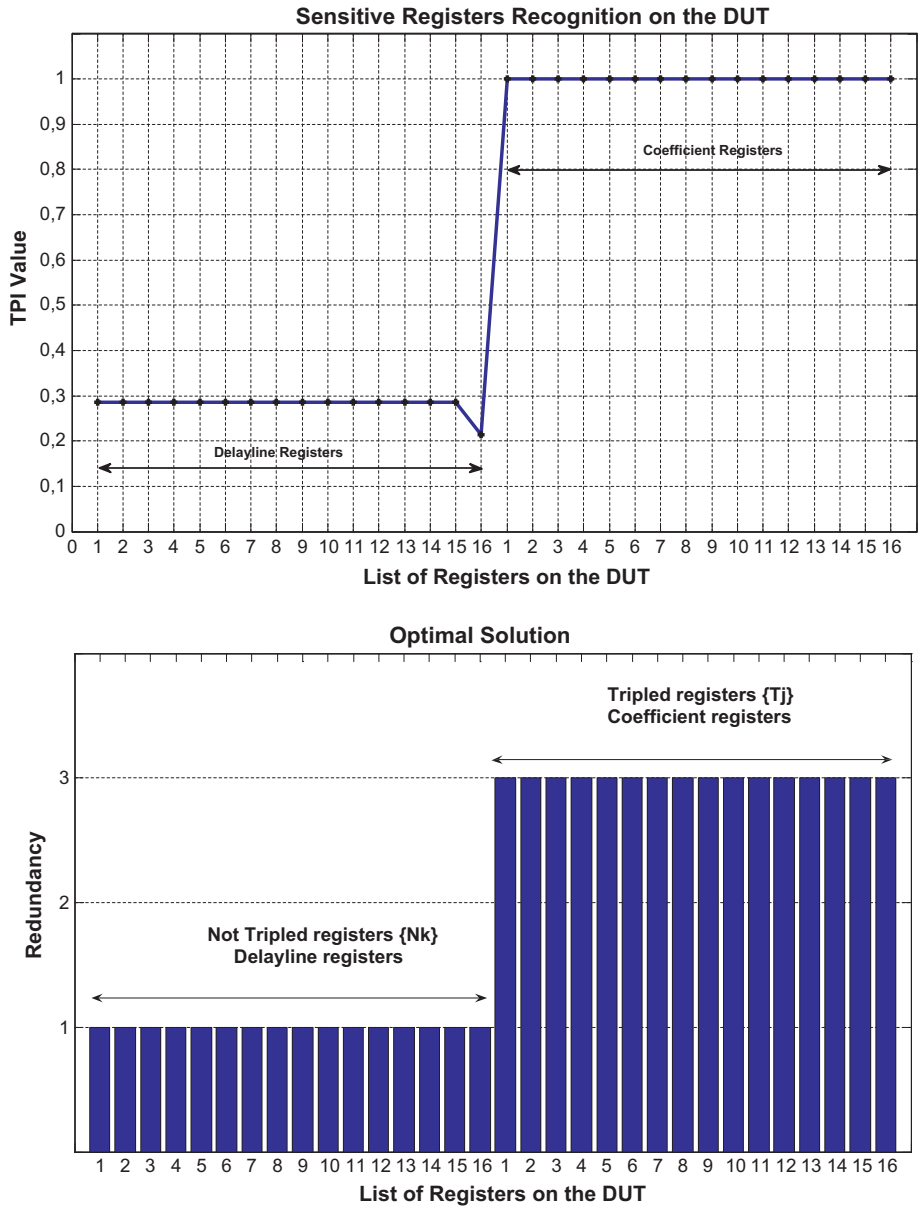

Fig. 7. Case study: initial partition according to the TPI model (Top); optimal solution (Bottom).

Table 13
Comparison of improvement for both area and speedup.

Initial partition	Initial tripled registers $\{t'_j\}$	Initial non-tripled registers $\{n'_k\}$	Number of iterations to optimal
Proposed (TPI)	Coefficients $h[i]$	Delayline $x[i]$	**30**
Full TMR	All	$\varnothing$	**441**

- The benefit of applying an optimization process with a selective initial partition based on topological criteria (TPI) has been shown, reducing the computation time to achieve the optimal solution compared to the default case of choosing an arbitrary initial partition.
- Also, a real example of a space circuit has been used, the Feed Forward Equalizer (FFE), proving that the presented methodology works well with realistic circuits.

Acknowledgments

This work was supported by the Spanish Ministry of Science and Education under Grant AYA2009-13300-C03-01, the Regional Government of Madrid and the European Union FEDER programme.

References

[1] Schrimpf RD, Fleetwood DM. Radiation effects and soft errors in integrated circuits and electronic devices. World Scientific Publishing; 2004 [ISBN: 981-238-940-7].
[2] Dodd PE, Massengill LL. Basic mechanisms and modeling of single-event upset in digital microelectronics. IEEE Trans Nucl Sci 2003;50(3):583–602.
[3] Carmichael C. Triple modular redundancy design techniques for Virtex Series FPGA, Application Notes 197, San Jose, USA, Xilinx; 2000.
[4] Hentschke R, Marques F, Lima F, Carro L, Susin A, Reis R. Analysing area an performance penalty of protecting different digital modules with hamming code and triple modular redundancy. In: Symposium on integrated circuits and systems design, proceedings, September 2002.
[5] Reviriego P, Maestro JA, Ruano O. Efficient protection techniques against SEUs for adaptive filters: an echo canceller case study. IEEE Trans Nucl Sci 2008;55(3):1700–7.

[6] Reddy A, Banarjee P. Algorithm-based fault detection for signal processing applications. IEEE Trans Comput 1990;39(10):1304–8.
[7] Samudrala PK, Ramos J, Katkoori S. Selective triple modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs. IEEE Trans Nucl Sci 2004;51(October):2957–69.
[8] Ruano O, Maestro JA, Reviriego P. A methodology for automatic insertion of selective TMR in digital circuits affected by SEUs. IEEE Trans Nucl Sci 2009;56(4):2091–102.
[9] Fiduccia C, Mattheyses R. A linear time heuristic for improving network partitions. In: Proceedings of the 19th IEEE design automation conference; 1982. p. 175–81.
[10] Qureshi S. Adaptive equalization. IEEE Commun Mag 1982;20(2):9–16.
[11] Treichler JR, Fijalkow I, Johnson CR. Fractionally spaced equalizers. IEEE Signal Proc Mag 1996;13(3):65–81.
[12] De Micheli G. Synthesis and optimization of digital circuits; 1994 [ISBN:0-07-016333-2].
[13] Gonzalez D. Single event upset simulation tool functional description. ESA report TEC-EDM/DCC-SST2; July 2004.
[14] Parrotta B, Rebaudengo M, Reorda MS, Violante M. New techniques for accelerating fault injection in VHDL descriptions. In: On-Line Testing Workshop, IEEE; 2000. p. 61–6.
[15] Stallings W. Data & computer communications. Pearson Prentice Hall; 2007 [ISBN-10: 0132433109].
[16] Almukhaizim S, Makris Y. Soft error mitigation through selective addition of functionally redundant wires. IEEE Trans Reliab 2008;57(1):23–31.
[17] European Space Agency, July 2009 <http://spacewire.esa.int/content/Home/HomeIntro.php>.
[18] Woo L, Guthaus R. Fault-tolerant synthesis using non-uniform redundancy. In: International conference on computer design (ICCD), Lake Tahoe, (CA) IEEE; 2009.
[19] Krishnaswamy S, Plaza SM, Markov IL, Hayes JP. Signature-based SER analysis and design of logic circuits. IEEE Trans Comput Aid Des Integr Circ Syst (TCAD) 2009;28(1):74–86.
[20] Zhou Q, Mohanram k. Gate sizing to radiation harden combinational logic. IEEE Trans Comput Aid Des Integr Circ Syst (TCAD) 2006;25(1):155–66.
[21] Dabiri F, Nahapetian A, Massey T, Potkonjak M, Sarrafzadeh M. General methodology for soft-error-aware power optimization using gate sizing. Trans Comput Aid Des Integr Circ Syst (TCAD) 2008;27(10):1788–97.
[22] Omana M, Rossi D, Metra C. Latch susceptibility to transient faults and new hardening approach. IEEE Trans Comput 2007;56(9):1255–68.
[23] Nicolaidis M. Design for soft error mitigation. IEEE Trans Device Mater Rel 2005;5(3):405–18.
[24] Touloupis E, Flint A, Chouliaras A, Ward D. Study of the effects of SEU-induced faults on a pipeline protected microprocessor. IEEE Trans Comput 2007;56(12):1585–96.
[25] Rao R, Joshi V, Blaauw D, Sylvester D. Circuit optimization techniques to mitigate the effects of soft errors in combinational logic. ACM Trans Des Automat Electron Syst (TODAES) 2009;15(1):1–27.
[26] Lee K, Shrivastava A, Issenin I, Dutt N, Venkatasubramanian N. Partially protected caches to reduce failures due to soft errors in multimedia applications. IEEE Trans Very Large Scale Integr (VLSI) Syst 2009;17(9):1343–7.
[27] Hutton MD, Rose JS, Corneil DG. Automatic generation of synthetic sequential benchmark circuits. IEEE Trans Comput-Aid Des Integr Circ Syst 2002;21(8):928–40.
[28] Reviriego P, Maestro JA, Liu S. Efficient soft error-tolerant adaptive equalizers. IEEE Trans Circ Syst I 2010;57(8):2032–40.