Sánchez-Clemente, A.J., Entrena, L., García-Valderas, M. (2016). Partial TMR in FPGAs Using Approximate Logic Circuits. *IEEE Transactions on Nuclear Science*, Vol. 63, Iusse 4, pp. 2233-2240.

DOI: 10.1109/TNS.2016.2541700

Partial TMR in FPGAs using Approximate Logic Circuits

A. J. Sánchez-Clemente, L. Entrena, M. García-Valderas

***Abstract*— TMR is a very effective technique to mitigate SEU effects in FPGAs, but it is often expensive in terms of FPGA resource utilization and power consumption. For certain applications, Partial TMR can be used to trade off the reliability with the cost of mitigation. In this work we propose a new approach to build Partial TMR circuits for FPGAs using approximate logic circuits. This approach is scalable, with a fine granularity, and can provide a flexible balance between reliability and overheads. The proposed approach has been validated by the results of fault injection experiments and proton irradiation campaigns.**

***Index Terms*— Single Event Upset, Triple Modular Redundancy, FPGA, Approximate circuit, selective mitigation.**

I. Introduction

FPGAs are becoming increasingly attractive for space applications. In comparison with ASICs, they provide higher flexibility and lower cost, particularly for the low volume production which is characteristic of space applications. As technology progresses, new devices with increased resources and performance are becoming available. Unfortunately, FPGAs are susceptible to radiation-induced Single-Event Upsets (SEUs). Thus, SEU mitigation is generally required for applications that operate in a radiation environment [1]-[5].

In SRAM-based FPGAs, SEUs can affect the configuration memory and provoke errors that remain until the device is reconfigured. A commonly used technique to remove configuration errors consists in the periodic refresh of the configuration data [6]. This technique is known as configuration scrubbing. New technologies provide increasing support for configuration scrubbing. For instance, Xilinx 7 Series FPGAs include a module that can automatically check the configuration memory and correct errors [7]. However, scrubbing has significant error detection latency and therefore it cannot prevent temporary erroneous behaviour. For this reason, scrubbing is often combined with Triple Modular Redundancy (TMR) [1]-[5]. Support for the automatic application of TMR techniques is currently provided by some tools, such as TMRTool from Xilinx [8] or BYU-LANL Partial TMR (BLTmr) tool [9].

Although TMR is a very effective mitigation technique, it is often expensive in terms of FPGA resource utilization and power consumption [4]. For applications that can tolerate some temporary misbehaviour, Partial TMR can be used to trade off the reliability with the cost of mitigation. In [3] and [4] an automatic solution for Partial TMR is proposed that is based on the concept of persistence. A persistent configuration bit is a sensitive configuration bit that will cause an error when upset that cannot be recovered by scrubbing, so that even after repairing persistent configuration bits through configuration scrubbing, the FPGA circuit does not return to normal operation. On the contrary, non-persistent bits imply some data loss, but the design returns to normal operation when the error is repaired through configuration scrubbing. Persistent bits can be found by topological analysis, looking for feedback structures. These feedback structures are associated to persistent bits and thus must be triplicated first. If resources allow, mitigation is applied to the non-persistent circuit structures to reduce the remaining design sensitivity.

In this work we propose the use of approximate logic circuits [10] to implement Partial TMR in FPGAs. Given a logic circuit, an approximate logic circuit is a circuit that performs a possibly different but closely related logic function, so that it can be used for error detection or error masking where it overlaps with the original circuit. Then, Partial TMR can be implemented by voting among approximate logic circuits instead of exact copies of the original circuit. The goal is to find approximate logic circuits that cover the persistent or most critical errors and reduce mitigation on errors that are less critical to reduce resource utilization.

The proposed approach is scalable, with a fine granularity, and can provide a flexible balance between reliability and overheads. In particular, an advantage of this approach is that it can selectively provide protection against unidirectional errors, i.e., errors that show up as a change of the output logic value from 0 to 1 or from 1 to 0.

Approaches to build approximate logic circuits for partial mitigation of Single-Event Transients (SETs) in combinational circuits have been proposed in [10]-[16]. However, to the best of our knowledge this is the first time approximate logic circuits have been used for partial error mitigation in FPGAs, with the exception of the preliminary proposal presented in [21]. The proposed approach has been validated with a proton

This work was supported in part by the Spanish Ministry of Economy and Competitiveness under contract ESP2015-68245-C4-1-P.

A. J. Sánchez-Clemente, L. Entrena and M. García-Valderas are with the University Carlos III of Madrid, Electronic Technology Department, Avda. Universidad, 30, Leganes (Madrid), Spain. (e-mails: ajscleme@ing.uc3m.es, entrena@ing.uc3m.es and mgvalder@ing.uc3m.es)

irradiation experiment on an Artix-7 FPGA. In addition, fault injection campaigns have been carried out to elaborate on the analysis. The results show that Partial TMR using approximate logic circuits can be applied to significantly reduce the overhead without degrading the error mitigation for critical errors and even improving it with respect to conventional TMR techniques.

This paper is organized as follows. Section II introduces approximate logic circuits and summarizes previous work on this topic. Section III describes the logic approximation techniques for FPGAs that have been developed in this work. Section IV describes the approach used to select approximations. Section V presents the experimental results. Finally, section VI summarizes the conclusions of this work.

II. Approximate Logic Circuits

Given a logic function *G*, a logic function *G'* which correctly predicts the result of *G* for a fraction of its input space is called an approximate logic function with respect to *G*. Therefore, *G'* can be used for partial error mitigation of *G* in those cases where both functions overlap. The interest of this idea lies in finding approximate functions with a good balance between overheads and protection against faults.

When approximate logic circuits are used for fault-tolerance, it is necessary to identify the overlapping cases for comparison. In general, an additional logic function can be used to explicitly mark such cases, which is referred as indicator function [12]. However, this approach requires that the indicator function is robust. A more convenient approach is to implicitly identify the overlapping areas by means of implication relationships between logic functions.

A logic function F which satisfies the implication $F \Rightarrow G$, i.e, $F \subseteq G$, is called an under-approximation or 1-approximation with respect to *G*. Conversely, a logic function H is an over-approximation or 0-approximation of G if $\overline{H} \Rightarrow \overline{G}$ i.e. if $G \subseteq H$. These implication relationships are shown graphically in Fig. 1 [10], where the on-sets of the original and the approximate logic functions are represented. For all input vectors in the gray area, the original and the approximate functions produce the same result. Therefore, an error in any of the functions will be masked by the other two. The white area corresponds to input vectors for which one of the approximate logic functions does not produce a correct result. In this case, an error in any of the other two functions will not be masked.

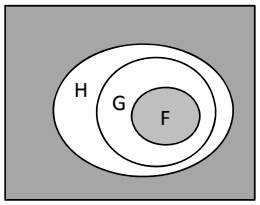


Fig. 1. Input space of the original function G, an under-approximation F, and an over-approximation H.

Approximate logic functions that satisfy the implication relationships can be used to protect against faults in several ways, such as partial error detection by checking the implication relationship or partial error masking with a TMR-like scheme [13].

Consider the schematic shown on Fig. 2(a). This is similar to TMR but using an over-approximation H and an under-approximation F instead of exact replicas of the original circuit *G*. With this schematic, the circuit is protected against single faults as long as the three circuits give the same result, which is represented in Fig. 1 as the grey area. In comparison with conventional TMR, this schematic does not provide full coverage against single faults. However, if approximations are properly chosen, relevant resource savings can be obtained with a low impact on the error masking capabilities. In addition, the implication relationship $F \subseteq G \subseteq H$ guarantees that the correct result is obtained in the absence of faults, because at least one of the two approximations agrees with the original circuit for every input vector. This schematic can be extended for sequential circuits as shown in Fig. 2(b). Approximations of the combinational part of the circuit are generated, and then both outputs and flip-flops are voted. It must be noted that voters can be affected by faults and therefore they should be hardened either by design or by using triple voters.

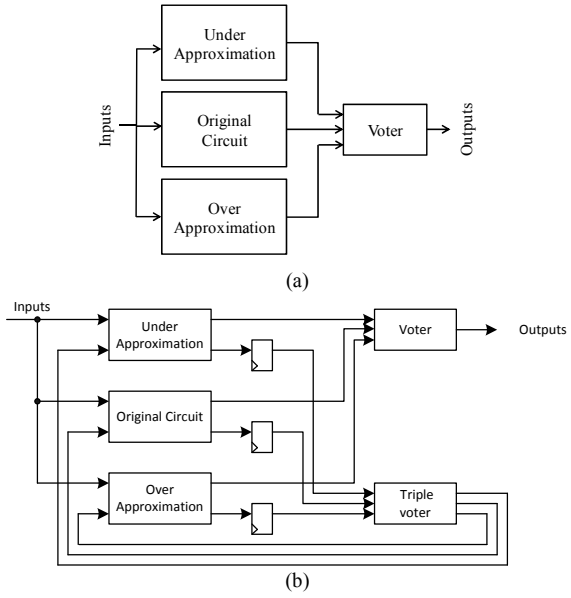


Fig.2. TMR schemes using approximate circuits for (a) a combinational circuit and (b) a sequential circuit.

Several methods have been proposed to generate approximations for a given logic circuit. Most of these methods try to approximate the implementation of logic functions using synthesis techniques [10], [11], [12], [15], [16]. Synthesis techniques depart from a functional model of the original circuit, such as a Binary Decision Diagram (BDD) [10], and apply synthesis algorithms to simplify the logic function while relaxing the constraint of matching the original

function. In particular, references [11] and [12] use cube elimination based on the Observability Don't Cares (ODCs), [15] uses two-level minimization and [16] uses boolean factoring. A disadvantage of synthesis-based approximation techniques is that the error mitigation reduction caused by the approximation transformations is difficult to estimate, because approximation is based on functional considerations. As a result, these techniques generally offer limited scalability.

An alternative approach to synthesis techniques is the line substitution technique [13], [14]. This technique departs from a technology-dependent network (netlist) of the original circuit and produces approximations by substituting some of the lines of the circuit with logic constants. Then, the logic originally used to implement the substituted lines is removed. In this case, each line substitution can be associated to a line stuck-at fault and the probability of the unprotected input combinations can be estimated through fault simulation. Another advantage of the line substitution technique is that, under certain conditions that can be automatically checked [13], each line substitution is guaranteed to produce either an under-approximation or an over-approximation, so that the implication relationships can be preserved by construction.

III. Approximate Logic Circuits in FPGAs

In this work, we have adapted the line substitution technique [13] for the case of FPGA circuits. Ideally, the internal structure of the configuration memory should be taken into account to guide the approximation process and to evaluate the impact of each approximation in the amount of essential bits. However, as this information is not generally available to the average designer, our approach is based on the netlist structure.

In a FPGA, the combinational circuit structure is typically composed of Look-Up Tables (LUTs). A LUT is a small memory that implements a logic function of its inputs, typically up to 6 inputs in modern FPGA technologies.

In this context, the application of the line substitution technique requires special attention in the sense of which logic transformations are performed to generate approximations. In particular, the most interesting line substitutions are those that effectively reduce the number of LUTs in the circuit, either by eliminating LUTs or by merging contiguous LUTs. The contrary will result in a degradation of the logic function of the circuit without achieving any benefits in terms of resource utilization. Reducing the size of a LUT, by removing some of its inputs, may also contribute to reduce the interconnection needs and the associated configuration bits. Eventually, when the input or the output of a flip-flop is substituted by a constant, the flip-flop can be removed and the voting logic can also be simplified.

To remove a LUT, we can set the output of the LUT to a logic constant. The result of this transformation can be analyzed using Shannon's expansion. Let G be the function implemented by a LUT and x one the inputs to the LUT. Using Shannon's expansion formula, we can express

$$G = G_{\bar{x}}\bar{x} + G_x \mathrm{x} \quad (1)$$

where $G_{\bar{x}}$ and G_x are the cofactors of G with respect to x.

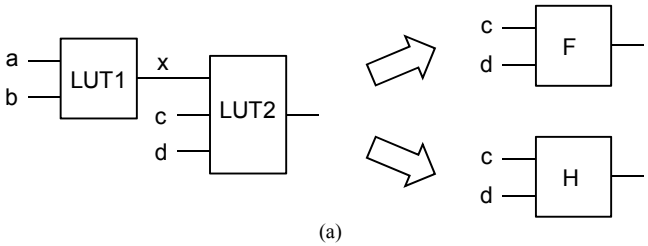
In Boolean Algebra, the classical way to reduce a logic function is to use the consensus and smoothing functions:

$$F = G_{\bar{x}}G_x \quad (2)$$

$$H = G_{\bar{x}} + G_x \quad (3)$$

It is easy to demonstrate that F is an under-approximation of G, because $F = 1 \Rightarrow G_x = G_{\bar{x}} = 1 \Rightarrow G = 1$. Similarly, H is an over-approximation of G, because $H = 0 \Rightarrow G_x = G_{\bar{x}} = 0 \Rightarrow G = 0$. The consensus and smoothing functions for a LUT can be easily computed by simple operations on its truth table.

Fig. 3 shows an example involving two LUTs. To remove LUT1, we can approximate LUT2 on input x. Cofactors and approximation transformations can be easily computed on the truth table of LUT2, as shown in Fig. 3(b). The cases for which F and H differ from the original function are highlighted. It can be clearly seen that $G_{\bar{x}} + G_x$ is an over-approximation, as it covers the on-set of G, and $G_{\bar{x}}G_x$ is an under-approximation, as it covers the off-set of G.



(a)

x	c	d	G	$G_{\bar{x}}$	G_x	$G_{\bar{x}}G_x$	$G_{\bar{x}} + G_x$
0	0	0	0	0	1	0	1
0	0	1	0	0	0	0	0
0	1	0	1	1	1	1	1
0	1	1	1	1	0	0	1
1	0	0	1	0	1	0	1
1	0	1	0	0	0	0	0
1	1	0	1	1	1	1	1
1	1	1	0	1	0	0	1

c	d	F
0	0	0
0	1	0
1	0	1
1	1	0

c	d	H
0	0	1
0	1	0
1	0	1
1	1	1

(b)

Fig. 3. LUT approximation example: a) Structure of under-approximation and over-approximation; b) Truth tables

Alternatively, the approximation can be performed on any of the inputs of LUT1 or LUT2. If one of the inputs of LUT1 is removed, then LUT1 becomes a single-input LUT that can be merged with LUT2 without increasing the number of inputs of LUT2. If the inputs c or d of LUT2 are removed, then LUT1 and LUT2 can be combined into a single 3-input LUT.

A. Relation with stuck-at faults

The next step is to evaluate the effects of the approximations in the error correction capabilities. When an approximation is taken, the approximate circuit differs from the original circuit for some input vectors. For these input

vectors, the Partial TMR circuit is unprotected. The probability of unprotection can be analyzed by relating the difference to stuck-at faults.

To show the relationship between approximation and stuck-at faults, we start with the particular case of LUTs that implement unate functions. A function is said to be *unate* if one of the cofactors is covered by the other. More precisely, a function G is positive unate in x if $G_{\bar{x}} \subseteq G_x$ and negative unate if $G_x \subseteq G_{\overline{x}}$. Otherwise, the function is said to be *binate*. Unate functions have many interesting properties, among them, a positive (negative) unate function can have an expression in which x only appears in uncomplemented (complemented form).

For a unate function, the under-approximation and over-approximation functions coincide with the cofactors. For a positive unate function, $G_{\bar{x}} \subseteq G_x$ and therefore $F = G_{\bar{x}}G_x = G_{\bar{x}}$ and $H = G_{\bar{x}} + G_x = G_x$. Similarly, for a negative unate function we have $F = G_x$ and $H = G_{\bar{x}}$.

From these results, we can easily see that the result of approximating a function G for a variable x is equivalent to forcing the variable x to a logic constant value, 0 or 1. In the case of a positive unate function, the under-approximation is obtained by forcing $x = 0$ and the over-approximation is obtained by forcing $x = 1$. Thus, the analysis of the differences between the approximate and the original circuit is related to the faults x stuck-at 0 or x stuck-at 1, respectively. The probability that the approximate circuit differs from the original circuit is the probability of the input vectors that test the associated stuck-at fault.

In the general case of a binate function, any expression of the function will contain x in both uncomplemented and complemented form. However, the approximation can also be associated to stuck-at faults. Shannon's expansion formula separates the contribution of the input x in complemented and uncomplemented form, so that x can be represented in the logic circuit by two lines x_0 (complemented) and x_1 (uncomplemented). In addition, let us extend Shannon's expansion formula (1) by adding the redundant consensus term $F = G_{\bar{x}}G_x$

$$G = G_{\bar{x}}\overline{x_0} + G_x x_1 + G_{\bar{x}}G_x \quad (4)$$

If we force $x_0 = 0$ and $x_1 = 1$ in this expression, we obtain $G = G_{\bar{x}} + G_x + G_{\bar{x}}G_x = H$. Thus, the over-approximate circuit is related to the union of the faults x_0 stuck-at 0 and x_1 stuck-at 1. Note that the sets of test vectors for x_0 stuck-at 0 and x_1 stuck-at 1 are disjoint, because they require different values at x. Therefore, the probability that the approximate circuit differs from the original circuit is the sum of the probabilities of the input vectors that test the associated stuck-at faults Analogously, if we force $x_0 = 1$ and $x_1 = 0$ in this expression, we obtain $G = G_{\bar{x}}G_x = F$ and the under-approximate circuit is related to the union of the faults x_0 stuck-at 1 and x_1 stuck-at 0.

Once the relationship between approximations and stuck-at faults has been established, we can use a stuck-at fault simulator to evaluate the quality of the approximations. To this purpose, the original LUT-based netlist is transformed into a logic netlist using the extended Shannon's expansion expression (4) for binate functions. Fig. 4 shows this representation for the example in Fig. 3. In this representation, we simulate the stuck-at faults at the inputs of all LUTs, including the complemented and uncomplemented lines for binate inputs. The quality of each approximation is measured by counting the number of clock cycles for which the associated stuck-at faults are detected. In this work we have used the stuck-at fault simulator HOPE [17] for this purpose. For instance, the over-approximation of LUT2 for input c is associated to the fault c stuck-at 1 because LUT2 is positive unate on c. On the other hand, to evaluate the effect of removing input x and hence LUT1, we sum the number of clock cycles for which the faults x_0 stuck-at 0 and x_1 stuck-at 1 are detected (over-approximation) or the faults x_0 stuck-at 1 or x_1 stuck-at 0 are detected (under-approximation).

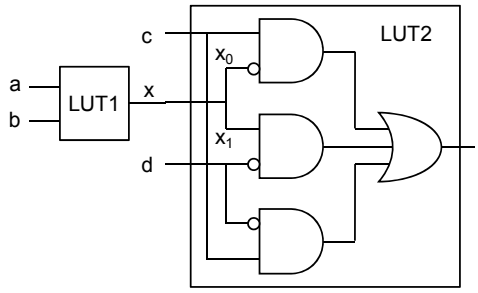


Fig. 4. Logic representation of the example of Fig. 3

IV. SELECTING APPROXIMATIONS

Identifying the most critical or persistent bits and the circuit primitives affected by them is a difficult task. Fault injection can be used for this purpose, as proposed in [3]. Although there are some known differences between the fault-injection method and a radiation testing environment, it is shown in [3] that a good estimate of the dynamic cross section can be obtained by fault injection with a careful test design. However, this approach is very time consuming. In practice, the authors reason that circuit primitives that are part of feedback structures within the design contribute to the persistent error behaviour, and use a tool to identify the feedback structures [4].

In our case, we use a probabilistic approach to identify the most critical circuit lines. Our approach is based on analyzing the number of clock cycles for which the faulty circuit response differs from the correct one. Then, we consider a line as critical if a fault in the line produces a large amount of differences. The rationale of this criticality metric is that a fault that produces a very different output response implies a large data loss and most likely means the circuit functionality cannot be recovered. Conversely, an error that produces an almost correct response involves some data loss but the circuit can be considered as being operational.

Fig. 5 shows the results of this analysis for the circuit that has been used as a test case in the experiments. For each line in the circuit, we measured the effect of substituting it with a constant value, 0 or 1. As shown in the previous section, this

effect can be associated to permanent stuck-at faults. We used the stuck-at fault simulator HOPE [17] to count the number of clock cycles for which the circuit response is wrong. The results are shown in Fig. 5 in increasing order of the proposed criticality metric. It can be seen that there is an abrupt change around 25% of the faults. The faults on the left can be considered as non-critical and the faults on the right as critical. In order to establish a threshold, we have considered a fault as critical if it produces an erroneous response for more than 15% of the clock cycles in a representative testbench. This threshold can be changed according to the reliability requirements of the application. For the particular threshold used, we have verified that the non-critical faults actually belong to feed-forward logic. Therefore, this criterion coincides with the feedback criterion proposed in [4] for the studied circuit. In any case, the approximation method can use other criticality criteria.

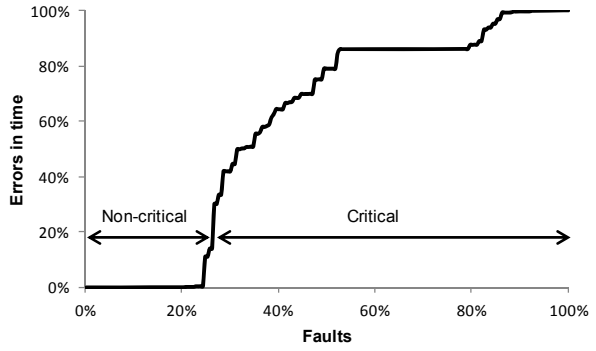


Fig. 5. Criticality estimation for the test case

After a classification of criticality is obtained, partial mitigation can be achieved by approximating the less critical lines first. The more lines are approximated, the lower the mitigation and the smaller the approximate circuits. Thus, the solution can be scaled with fine granularity to trade off the error mitigation with the resource utilization.

When TMR is used for partial mitigation, the choices for each component are to include it in the mitigated section or not. However, approximate logic circuits can consider an intermediate solution for unidirectional errors. For instance, an error that changes the logic value of a line from 0 to 1 may be critical while the opposite may be not. In such a case, we can replace the line by a constant 0 in just one of the approximate circuits. The overall effect is equivalent to having a non-critical unidirectional hardwired error in one of the three subcircuits that are voted.

A circuit that contains a unidirectional approximation still works correctly in the absence of SEUs because there are always two correct copies for voting. The circuit is not protected for non-critical errors in the same direction, because it is enough to have such an error in any of the other two copies for the circuit to fail. However, the circuit is still protected for critical errors in the opposite direction. As a matter of fact, for such errors to be unmasked it is necessary that the two correct copies have errors in the same critical direction. Critical errors are less likely to happen than in the TMR circuit, because in the TMR circuit an error is observed when two out of the three copies fail. Thus, the overall effect is a shift of the critical cross-section to the non-critical cross-section with respect to the TMR circuit.

Fig. 6 shows the flow diagram to obtain the final circuit using the proposed approach. First, a logic representation of the original circuit is obtained and stuck-at fault simulation is performed using a typical testbench. The results of fault simulation are used as a criterion to estimate the criticality of approximations and to select the less critical ones. Then, the original circuit is divided into the combinational components, mainly formed by LUTs, and sequential components (FFs). From the combinational component we generate an under-approximation and an over-approximation circuit based on the previous criticality estimation. The sequential components are hardened using TMR. Then, all the components are merged following the schematic in Fig. 2. This circuit is resynthesized, to optimize the logic resulting from the approximation process. Appropriate synthesis directives are used to avoid removing the redundant logic. With the exception of the stuck-at fault simulation, this process is performed automatically with in-house software tools. This approach can be extended to include other FPGA components, such as memories or DSP slices, which may require to be dealt with in a specific way.

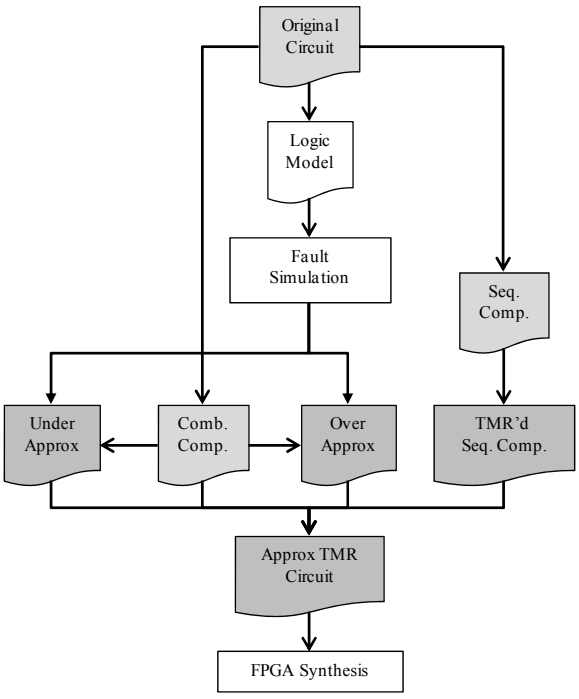


Fig. 6. Flow Diagram of the proposed approximate TMR approach.

V. Experimental Results

The proposed approach has been tested with the B13 benchmark from the ITC'99 set. This benchmark was selected for compliance with current efforts towards a common set of benchmarks that can be used for comparison among different experiments [18]. It also includes a set of pre-generated input

vectors that were designed by the ATPG community to fully cover the functionality of the circuit.

Firstly, we implemented a TMR version of B13 by triplicating the logic and using triple voters at the output of each flip-flop. The outputs of the circuit were also voted using single voters. Inputs and clock lines were not triplicated for the sake of simplicity. From this TMR design, we implemented four different Partial TMR designs (A1 to A4) using approximate circuits, where A1 has the lowest number of approximated lines and A4 the highest. Approximations were based on the results of the criticality analysis described in the previous section. As a general target, we considered only approximations that produce less than 15% erroneous responses in the execution of the full set of input stimuli. Then, for each design, lines with a percentage of errors below a selected threshold were replaced by constants. Namely, we used thresholds of 0.05%, 0.2%, 2% and 15% for A1 to A4, respectively. For the sake of comparison, we also considered a TMR design using single voters (SV) and the original unmitigated design (ORIG) of B13.

The synthesis results for all considered designs, as given by Xilinx Vivado tool, are shown in Table I. The designs are ordered from the most protected (TMR) to the less protected (ORIG). The last two columns show the relative use of resources with respect to the TMR version.

TABLE I. SYNTHESIS RESULTS

Design	#FFs	#LUTs	%FFs	%LUTs
TMR	135	298	100%	100%
A1	127	287	94%	96%
A2	119	276	88%	93%
A3	117	265	87%	89%
A4	115	261	85%	88%
SV	135	213	100%	71%
ORIG	45	53	33%	18%

The experiments were run on an Artix7 XC7A100T FPGA from Xilinx. As the B13 benchmark is rather small, we included 24 instances of each design in the same FPGA. Synthesis options were used to ensure instances of each circuit version were not optimized away. However, the placement was not constrained. All designs run concurrently using the same input stimuli, which are the ITC'99 proposed input stimuli. We also included a small checker circuit to detect if any of the design copies produces an error or if the percentage of errors in a single execution of the full set of input stimuli is greater than the selected target of 15%. The checkers and the interface to the host are tripled to reduce the impact of errors in these modules on the measures. The complete circuit, including all copies of all designs, used 66% of the LUTs and 15% of the flip-flops of the FPGA device.

A. Radiation test results

Radiation ground testing has been carried out in CNA facilities (Centro Nacional de Aceleradores, Sevilla, Spain). We used a cyclotron, capable of accelerating protons and deuterons up to 18 MeV. Although it was originally intended for radioisotope production in medical applications, an external beam allows testing electronic circuits in either vacuum or open air.

FPGA test has been performed with protons in open air, with 18 MeV energy and flux range between 10^8 to 10^9 p/cm^{2}s. A single device has been exposed, as shown in Table II. The device was allowed to run without scrubbing or reconfiguring it until we observed most of the design versions had a critical error. Then it was fully reconfigured and checked again. In Run 3, with the largest fluence, 120 reconfiguration cycles were completed.

TABLE II. XC7A100T IRRADIATION TEST

Run	Energy (MeV)	Flux (p/cm^{2}s)	Fluence (p/cm^2)
1	18	10^8	1.23×10^{11}
2	18	2.5×10^8	11.0×10^{11}
3	18	10^9	50.3×10^{11}
Total			62.53×10^{11}

Fig. 7 shows a comparison of the Mean Time To Failure (MTTF) obtained with the largest fluence. The results for lower fluences are similar, but they are not reported here because the number of collected errors is small and the confidence intervals are wide. Cross-section can be obtained as the inverse of the product of MTTF times the flux (10^9p/cm^{2}s). For each version, the non-critical MTTF was measured as the mean time until the first error is detected in any of the 24 design copies. The critical MTTF was measured as the mean time until the first critical error is detected, i.e., until the percentage of erroneous responses in a single execution of the full set of input stimuli is greater than the selected target of 15%. The confidence intervals for 95% confidence level are also displayed in the figure.

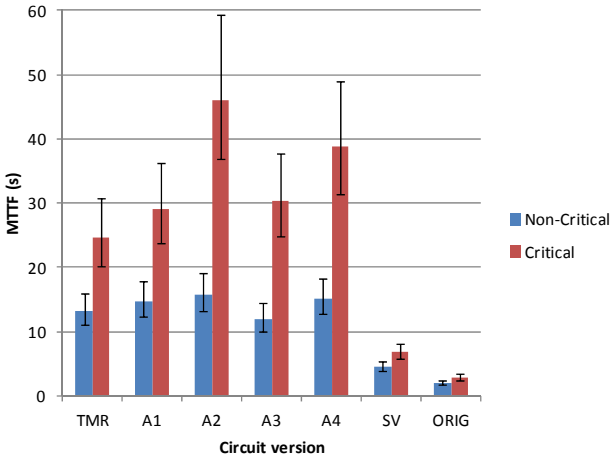


Fig. 7. MTTF for the tested design versions.

As expected, the original unmitigated design shows a low MTTF in comparison with the mitigated versions. The critical and non-critical MTTF are very close, because most errors produce a critical effect. The version using single voters

improves MTTF with respect to the original version by a factor of approximately 2.4. For the full and all the Partial TMR versions, the non-critical MTTF is similar and between 6 to 8 times greater than that of the original design. Even though some degradation of the non-critical MTTF can be expected for the approximate logic circuits, the differences in the results are small and some Partial TMR versions can even be better than TMR. On the other hand, the results clearly show that the MTTF for critical errors is significantly improved in the Partial TMR versions. This behaviour can be explained by the reduction in size produced by the use of approximate logic circuits in combination with the shift of the critical cross-section to the non-critical cross-section that can be achieved by unidirectional approximation.

B. *Fault injection results*

To complement the analysis, we have also performed several fault injection campaigns with the same circuit versions and the same FPGA. Fault injection was implemented using the Soft Error Mitigation (SEM) Core from Xilinx [7].

It must be noted that fault injection results may not accurately match radiation results due to several reasons [9], [19], [20]. First of all, fault injection must rely on the mechanisms provided by the manufacturer for the access to the configuration memory. The documentation about such mechanisms is generally very limited, so that we cannot guarantee a fair fault injection campaign. In particular, fault injection cannot emulate changes in the internal proprietary state of an FPGA (i.e., internal registers or global logic for managing the device). On the other hand, the SEM Controller is implemented in the FPGA and it can be affected by the injected faults. Finally, we have only injected single bit errors in the configuration memory and assume that all configuration bits are equally vulnerable, which may not be true in real devices. In spite of these limitations, fault injection plays a very important part in evaluating and validating FPGA designs for use in radiation environments [19]. We will show here that fault injection results for the proposed approach follow a pattern similar to the one of radiation test results.

To implement fault injection, we added the SEM Controller to the same FPGA design used for radiation testing. Fault injection was monitored through the serial interface provided by the SEM Controller. This interface allowed us to observe the internal state of the SEM Controller and inject faults. The controller can also correct the internal configuration memory errors. Because faults are injected randomly in the configuration memory of the FPGA, the SEM Controller may fail. However, this situation can be detected through the serial interface. When an error cannot be corrected or we observe abnormal operation of the SEM Controller through the serial interface, the FPGA is fully reconfigured and the fault injection process is resumed.

In the first fault injection campaign, we injected faults in random addresses of the configuration memory at regular time intervals. A sufficiently large time interval (12.2 ms) was selected to allow for the complete execution of the full set of input stimuli (7640 input vectors, 0.15 ms). Following the same approach as in the radiation testing experiment, the device was allowed to run without scrubbing or reconfiguring it until we observed most of the design versions had a critical error or the SEM Controller failed. Then the device was fully reconfigured and checked again.

Fig. 8 summarizes the results obtained with this fault injection campaign. We run 215 reconfiguration cycles with a total of 389,764 injected faults. As as similar metric to MTTF, we have used the ratio of injected faults to failures, called here MIFTF (Mean Injected Faults To Failure). As in the radiation test results, the difference between the non-critical MIFTF and the critical MIFTF is small for the original unprotected design and for the design using single voters, while the protected versions (TMR and A1 to A4) clearly improve the critical MIFTF. The Partial TMR versions show similar non-critical MIFTF to the full TMR version but the critical MIFTF is generally improved.

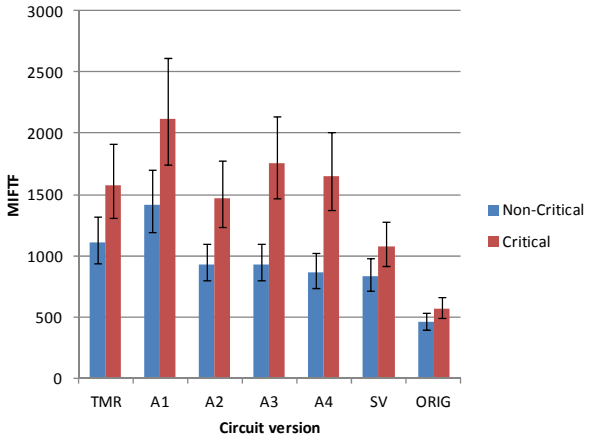


Fig. 8. MIFTF for the tested design versions using fault injection without error correction.

In a second fault injection campaign we used the error correction capabilities of the SEM Controller. In this case, we injected a single fault in a random address of the configuration memory, waited for sufficient time to make sure the full set of input stimuli has been executed and then let the SEM Controller attempt to correct the injected fault. This procedure is repeated until the SEM Controller reports an uncorrectable error or the SEM Controller itself fails. Then, the FPGA is fully reconfigured before continuing the fault injection campaign.

The SEM Controller supports three different error correction modes [7]:

- Repair: ECC algorithm-based correction
- Enhanced Repair: ECC and CRC algorithm-based correction
- Replace: Data reload based correction

In our fault injection experiments we used the Enhanced Repair mode, because it provides higher error correction capabilities, namely correction of configuration memory frames with single-bit errors or double-bit adjacent errors. The Replace mode supports correction of configuration memory

frames with arbitrary errors, but requires partial reconfiguration of the FPGA.

The results for the fault injection campaign using error correction are shown in Fig. 9. In this campaign we injected 75,620 faults with a time interval of 37.4 ms between faults in 86 fault injection cycles. In this case we can see that the non-critical MIFTF decreases as the protection for non-critical errors decreases from TMR to A4. On the other hand, the critical MIFTF is similar in all these circuit versions. This is the expected behaviour, because these versions differ in the protection for non-critical errors but keep a similar protection for critical errors.

Comparing the results with and without error accumulation, shown in Fig. 8 and Fig. 9, respectively, we can see that the full TMR version produces the best MIFTF when accumulation is prevented by error correction (Fig. 9), while this may not be true with accumulation (Fig. 8). The full TMR version has higher protection for non-critical errors at the expense of increasing the size of the circuit. The additional size increases the chances of error accumulation, for which the circuit is not protected, resulting in a smaller critical MIFTF. Thus, size plays a significant role when error accumulation is possible. By reducing the protection for non-critical errors, the approximate circuit versions can reduce the size and improve the critical MIFTF.

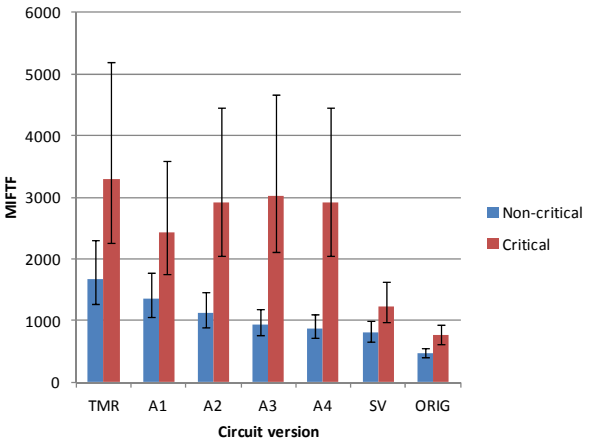


Fig. 9. MIFTF for the tested design versions with error correction.

VI. Conclusions

The application of full TMR to FPGA designs often results in very large overheads in resource utilization and, consequently, power consumption. For applications that can tolerate some temporary misbehaviour, Partial TMR can be used to trade off the reliability with the cost of mitigation.

In this work we have proposed a new approach to build Partial TMR circuits for FPGAs using approximate logic circuits. Experimental results demonstrate that this approach can provide protection for critical errors at lower hardware cost than full TMR. Moreover, it can provide higher protection than full TMR if errors are not corrected and may accumulate. For the case study used in the experiments, up to 15% of resources could be saved while the MTTF for critical errors could be improved by 57%.

References

[1] N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham, "Evaluating TMR techniques in the presence of single event upsets", Proc. 6th Annual Int. Conf. on Military and Aerospace Programmable Logic Devices (MAPLD), P63, Sept. 2003.

[2] P. K. Samudrala, J. Ramos, and S. Katkoori, "Selective Triple Modular Redundancy (STMR) Based Single-Event Upset (SEU) Tolerant Synthesis for FPGAs", IEEE Trans. on Nuclear Science, vol. 51, no. 5, pp. 2957–2969, Oct. 2004

[3] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin, "SEU-induced persistent error propagation in FPGAs," IEEE Trans. on Nuclear Science, vol. 52, no. 6, pp. 2438–2445, Dec. 2005.

[4] B. Pratt, M. Caffrey, P. Graham, E. Johnson, K. Morgan, and M. Wirthlin, "Improving FPGA design robustness with partial TMR," Proc. Int. Reliability Physics Symp. (IRPS), Mar. 2006.

[5] F. L. Kastensmidt, L. Sterpone, L. Carro, and M. Sonza Reorda, "On the Optimal Design of Triple Modular Redundancy Logic for SRAM-based FPGAs", Proc. Design, Automation and Test in Europe Conf. (DATE), vol. 2, pp. 1290–1295, Mar. 2005.

[6] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, and K. A. LaBel, "Effectiveness of Internal Versus External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA: Design, Test, and Analysis", IEEE Trans. on Nuclear Science, vol. 55, no. 4, pp. 2259–2266, Aug.. 2008.

[7] "Soft Error Mitigation Controller v4.1", Product Guide, Xilinx Corporation, PG036, Nov. 19, 2014

[8] "Xilinx TMRTool", Product Brief, Xilinx Corporation, 2009.

[9] B. Pratt, M. Caffrey, J. F. Carroll, P. Graham, K. Morgan and M. Wirthlin, "Fine-Grain SEU Mitigation for FPGAs Using Partial TMR", IEEE Trans. on Nuclear Science, vol. 55, no. 4, pp. 2274–2280, Aug. 2008.

[10] B. D. Sierawski, B. L. Bhuva, and L. W. Massengill, "Reducing Soft Error Rate in Logic Circuits Through Approximate Logic Functions", IEEE Trans. on Nuclear Science, vol. 53, no. 6, pp. 3417–3421, Dec. 2006.

[11] M. R. Choudhury, and K. Mohanram, "Approximate logic circuits for low overhead, non-intrusive concurrent error detection", Proc. Design Automation and Test in Europe Conf. (DATE), pp. 903–908, 2008.

[12] M. R. Choudhury, and K. Mohanram, "Low Cost Concurrent Error Masking Using Approximate Logic Circuits", IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 32, no. 8, pp. 1163– 1176, Aug. 2013.

[13] A. Sánchez-Clemente, L. Entrena, M. García-Valderas, and C. López-Ongil, "Logic Masking for SET Mitigation Using Approximate Logic Circuits", Proc. Int. On-Line Testing Symp. (IOLTS), pp. 176–181, July 2012.

[14] A. Sánchez-Clemente, L. Entrena, and M. García-Valderas, "Error masking with approximate logic circuits using dynamic probability estimations". Proc. Int. On-Line Testing Symp. (IOLTS), pp. 134-139, July 2014.

[15] H. Xie, L. Chen; R. Liu, A. Evans, D. Alexandrescu, S.-J. Wen, and R.Wong, "New approaches for synthesis of redundant combinatorial logic for selective fault tolerance", Proc. Int. On-Line Testing Symp. (IOLTS), pp. 62–68, July 2014.

[16] I. A. C. Gomes, M. Martins, F. L. Kastensmidt, A. Reis, R. Ribas and S.P. Novales, "Methodology for achieving best trade-off of area and fault masking coverage in ATMR", Proc. 15th Latin-American Test Symp. (LATS), pp. 1–6, March 2014.

[17] H. K. Lee, and D. S. Ha. "HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits", IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 15, no. 9, pp. 1048–1058, Sep. 1996.

[18] H. Quinn et al., "Using Benchmarks for Radiation Testing of Microprocessors and FPGAs", IEEE Trans. on Nuclear Science, vol. 62, no. 6, pp. 2547–2554, Dec. 2015.

[19] N. A. Harward, M. R. Gardiner, L. W. Hsiao, and M. J. Wirthlin, "Estimating Soft Processor Soft Error Sensitivity Through Fault Injection", Proc. IEEE 23rd Annual Int. Symp. on Field-Programmable Custom Computing Machines (FCCM), pp.143–150, May 2015.

[20] J. Tarrillo, J. Tonfat, L. Tambara, F. L. Kastensmidt, and R. Reis, “Multiple Fault Injection Platform for SRAM-based FPGA based on Ground-level Radiation Experiments”, Proc. 16th Latin-American Test Symposium (LATS), pp. 1–6, March 2015.

[21] A. J. Sánchez-Clemente, L. Entrena, and M. García-Valderas, “Partial TMR in FPGAs using Approximate Logic Circuits”, Proc. Radiation Effects on Components & Systems (RADECS), Sep. 2015.