A Reliable Fault Classifier for Dependable Systems on SRAM-based FPGAs

Cristiana Bolchini, Chiara Sandionigi
Politecnico di Milano, Dip. Elettronica e Informazione
Milano - Italy
{bolchini, sandionigi}@elet.polimi.it

Luca Fossati, David Merodio Codinachs
European Space Agency
Noordwijk, The Netherlands
{luca.fossati, david.merodio.codinachs}@esa.int

***Abstract*—This paper presents an algorithm for the discrimination of faults in FPGAs based on their recovery possibility; some faults can be recovered by reconfiguring the faulty part of the device, others have a destructive effect. After classification has been carried out, the suitable fault recovery strategy is applied, with the final aim of enabling the exploitation of FPGAs, in particular SRAM-based ones, for critical applications, such as the ones in the space environment. In this scenario, we investigate the reliable implementation of the fault classification algorithm, that can be so integrated in an overall reliable system.**

I. Introduction

SRAM-based Field Programmable Gate Arrays (FPGAs) are an attractive technology for the electronics of critical applications, thanks to the possibility of exploiting their reconfiguration capability to cope with the occurrence of faults. In this work, we identify two categories of faults based on the possibility to recover from them. *Recoverable faults* can be mitigated by reconfiguring the system (possibly only the faulty sub-system portion) with the same configuration used before fault occurrence, whereas *non-recoverable faults*, being characterized by a destructive effect, lead to the necessity of relocating the functionality to a non-faulty region. The identification of the type of fault occurred on the device is thus fundamental to apply the suitable recovery strategy.

This paper presents a module, called *Fault Classifier*, implementing an improved algorithm for fault classification. Since the module will be hosted on an FPGA as well, it is designed to be reliable itself, investigating and comparing different hardening implementations. In the past, few other algorithms have been proposed, and with respect to them the contributions of this work are: i) the formal definition of the parameters characterizing the algorithm, ii) the evaluation of the conditions for correct fault classification, and iii) the investigation of the reliable implementation of the module.

The paper is structured as follows: Sections II and III provide the background for the comprehension of the algorithm, by describing the related work and by showing the envisioned scenario, respectively. Section IV presents the proposed algorithm and Section V discusses its evaluation. Section VI investigates the implementation of the algorithm. Finally, Section VII draws some conclusions.

This work is partially supported by ESA/ESTEC Contract #22079/08/NL/JK and by Italian MIUR-PRIN project #2008K4P7X9.

II. Related work

Few works in literature deal with the problem of identifying faults that physically damage the device. The problem has been investigated in [1], [2], and [3]. The work presented in [1] proposes only a rough and not evaluated algorithm, since the focus is on the design of the overall engine managing fault tolerance. The reliability of the engine is guaranteed by periodically reading its configuration data, that is an expensive operation, prone to errors. The same authors later propose a detailed and refined version of the classification algorithm in [2]. In both approaches, the classification of the fault affecting a portion of the FPGA is based on the analysis of the relative and absolute frequency of the detected faults; if a portion of the FPGA is affected by a fault more frequently than the others, then the absolute frequency is analyzed, by taking into account the whole history of faults. In none of these two approaches, a formal definition and accurate study of the parameters characterizing the algorithms is provided. Another approach dealing with the discrimination of faults in FPGAs is presented in [3], where the authors propose the use of a timer; if two errors are revealed at the same position in a time interval smaller than a pre-defined threshold, it is assumed that they are related to the presence of a fault that has physically damaged the device. The main drawback of the approach is that it is based on the knowledge of the Mean Time Between Failure (MTBF), that is highly variable, being dependent on the specific operating conditions.

The work presented in [4] pursues an objective similar to ours, by proposing a mechanism for the diagnosis of hard faults in microprocessors. When an error is detected in an instruction executed by the microprocessor, an error counter for every Field Deconfigurable Unit (FDU) used by that instruction is incremented and, when a counter exceeds a threshold, the related FDU is identified as affected by a hard fault, and is *deconfigured*. As long as transient errors do not lead to the above-threshold error rates, the error counters are cleared periodically or when a deconfiguration is activated; moreover, the threshold is chosen not to be too small and is related to the specific FDU usage. Although the overall solution is presented, no indication is available on how to derive the parameters' value, and no discussion is introduced on the reliability characteristics of the strategy.

III. Envisioned scenario

This section introduces the adopted fault model and the elements necessary to understand the algorithm design.

A. Adopted fault model

Traditionally, the *single fault assumption* is adopted in designing fault mitigation strategies; such assumption implies that: i) faults occur one at a time, and ii) the time between the occurrence of two subsequent faults is long enough to allow detection of the first fault before the second occurs. The fault only produces an observable effect, an *error*, if i) the fault occurs in a used resource and ii) the applied input (sequence) is such that a difference in the data/behavior is caused with respect to the fault-free situation and the adopted detection mechanism identifies such situation (*fault-error relation*). The observability of a fault is related to the probability of the fault to hit a *used* resource (P_r). In [5], the following formula has been proposed for computing P_r:

$$P_r = \frac{\text{dynamic cross section}}{\text{static cross section}} \tag{1}$$

where the `static cross section` is the total sensitive fraction of the device, and `dynamic cross section` is the operational fraction. The other element that should be taken into account is the `latency`, that is the time, related to the fault-error relation, occurring between the instant the fault occurs and the time an error is detected.

B. Envisioned system

The algorithm is devised to operate in a system composed of two elements, as in [1]: the logic implementing the actual application, and a module, called *Reconfiguration Controller*, implementing the fault classification and the recovery strategy.

We envision the application system to be partitioned into n *Independently Recoverable Areas* (IRAs), such that each area hosts a Self-Checking portion of the system and generates error signals to allow the detection of faults (as provided by the methodology proposed in [6]). In order to avoid that an IRA is much more sensible than the others to faults, we assume that the overall observability of faults is uniform for all IRAs in the FPGA. As for the fault-error relation, it is also assumed that it does not sensibly change throughout the entire system/device.

The Reconfiguration Controller submodule implementing the algorithm, namely the Fault Classifier, receives the IRAs' error signals and discriminates between recoverable and non-recoverable faults. If the fault is considered as recoverable, the bitstream portion related to the faulty area is reloaded, otherwise the faulty area is relocated to a *spare area*. It is evident that it is necessary to prevent erroneous reconfigurations by identifying an erroneous behavior of the Reconfiguration Controller as soon as it is observable. When the Reconfiguration Controller signals the presence of a fault in itself, it is handled as an IRA and reconfigured by another controller hosted onto a different FPGA. Hence, fault detection capability is required for the Reconfiguration Controller, and consequently for the Fault Classifier.

IV. Fault classification algorithm

The fault classification algorithm is executed each time an error signal from the IRAs shows the presence of a fault. The proposed classification is based on the analysis of the fault's frequency; an IRA is considered affected by a non-recoverable fault when it has been "recorded" as faulty during the last K subsequent observations.

The challenge with this approach is selecting K; a too small K would lead to erroneously consider most faults as non-recoverable, while a too large K would cause not to recognize non-recoverable faults as such. In order to properly assist the designer of the reliable system in dimensioning K, we introduce the following elements: i) $P_{mis_{r-nr}}$ is the accepted probability of classifying a recoverable fault as non-recoverable, and ii) $P_{mis_{nr-r}}$ is the accepted probability of classifying a non-recoverable fault as recoverable. The value of K can be determined by using the above introduced probabilities of a mistake in the classification as thresholds in relation to the events that actually cause the algorithm to fail in classifying the fault. It is worth noting that, if the first kind of mistake is made, a relocation of the IRA deemed as faulty is performed, discarding an actually still healthy portion of the FPGA. In the second scenario, instead, useless reconfigurations are performed without a real benefit, thus incurring in a waste of time and effort, and in the accumulation of multiple faults.

Recoverable faults are mistaken for non-recoverable ones whenever K subsequent recoverable faults occur all in the same IRA, and such events occurs with a probability $P_{IRA\times K}$ equal to:

$$P_{IRA\times K} = \left(\frac{1}{n}\right)^K \tag{2}$$

Therefore, to keep the probability of a mistake within the desired threshold, we obtain:

$$K \geq \lceil log_{\frac{1}{n}} P_{mis_{r-nr}} \rceil \tag{3}$$

In the definition of K, also the other kind of classification mistake must be taken into account. A fault is mistaken as recoverable if a non-recoverable one is not characterized by K subsequent detections because a recoverable fault occurs within the sequence. As a result, as shown in Figure 1, to avoid this situation, the time necessary to classify a non-recoverable fault must be shorter than the time elapsing between two recoverable faults, defined as $MTBF_{rec}$. The probability of this event occurring is related to the latency lat, that is the time for a given non-recoverable fault to produce an error. Since for the first (K - 1) times the system tries to recover by reconfiguring the IRA assuming it to be a recoverable error, $(K-1) \cdot lat$ indicates the time necessary to recognize a non-recoverable fault, where lat is associated with that specific fault. Nevertheless, as stated above, we assume a homogeneous implementation of the system onto the device, such that this parameter can be assumed as unique for all faults, using an average value. The fault, not yet classified as non-recoverable, is hereafter called *not-recovered*. By considering

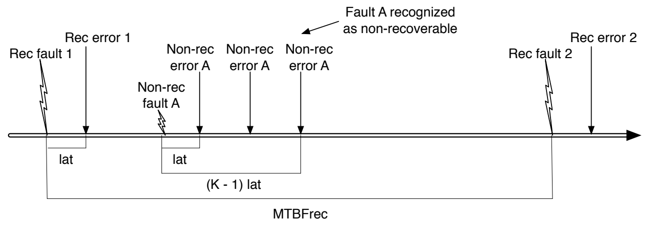

Figure 1. Non-recoverable fault classification

this scenario, the probability that a recoverable fault occurs before a previous, non-recoverable one is identified as such, is the following:

$$P_{lat} = \frac{(K-1) \cdot lat}{MTBF_{rec}} \tag{4}$$

Based on this equation, the threshold K must satisfy the following condition:

$$K \leq \frac{MTBF_{rec}}{lat} P_{mis_{nr-r}} + 1 \tag{5}$$

By selecting a value of K which satisfies both Equation 3 and Equation 5, we obtain an algorithm that correctly identifies non-recoverable faults with desired level of accuracy. In general, though, latency is neglected since the relation between fault and error is of orders of magnitude smaller than fault occurrence, hence Equation 5 could be neglected and no upper bound for K could be identified. However, the latency is a positive value that actually depends on the specific implementation, so an upper bound for K must be considered. We define the optimum value of K as follows:

$$K = \lceil log_{\frac{1}{n}} P_{mis_{r-nr}} \rceil \tag{6}$$

The choice of considering the errors' frequency for the classification is supported by the assumption that, when an IRA is affected by a non-recoverable fault, errors in that IRA are signaled with a higher frequency than errors due to recoverable faults. In fact, when an error is observed, a reconfiguration is triggered; if the fault is recoverable, no error is then detected for the subsequent time window corresponding to the $MTBF_{rec}$. If the fault is non-recoverable, the reconfiguration produces a benefit only until an input causes the fault to be observed again; an error in the same IRA is detected after latency lat. More rigorously, the frequency of a recoverable fault depends on its occurrence and its observability, whereas a not-recovered fault is already present in the system and its frequency depends only on the latency related to the fault-error relation. As stated, this value is of orders of magnitude smaller than the $MTBF_{rec}$, even for harsh environments as the space one. As a consequence, we derive that the $MTBF_{rec}$ is bigger than the time elapsing between observations of the same non-recoverable fault, hence we can expect a number of not-recovered fault observations, here identified as K, before a recoverable fault occurs.

Table I
PREDICTED λ_{rec} AT THE CONSIDERED ENVIRONMENTAL CONDITIONS

	Orbit	$\lambda_{rec_{min}}$ [SEU/day]	$\lambda_{rec_{max}}$ [SEU/day]	$\lambda_{rec_{week}}$ [SEU/day]	$\lambda_{rec_{day}}$ [SEU/day]
Multiplier					
exp1	LEO	$4.5 \cdot 10^{-2}$	$2.6 \cdot 10^{-2}$	$2.6 \cdot 10^{-2}$	$2.6 \cdot 10^{-2}$
exp2	Polar	$10.4 \cdot 10^{-2}$	$8.4 \cdot 10^{-2}$	$171.6 \cdot 10^{-2}$	$592.8 \cdot 10^{-2}$
exp3	MEO	$6.5 \cdot 10^{-2}$	$71.5 \cdot 10^{-2}$	$577.2 \cdot 10^{-2}$	$2028.0 \cdot 10^{-2}$
Counter					
exp4	LEO	$1.9 \cdot 10^{-2}$	$1.1 \cdot 10^{-2}$	$1.1 \cdot 10^{-2}$	$1.1 \cdot 10^{-2}$
exp5	Polar	$4.3 \cdot 10^{-2}$	$3.5 \cdot 10^{-2}$	$71.3 \cdot 10^{-2}$	$246.2 \cdot 10^{-2}$
exp6	MEO	$2.7 \cdot 10^{-2}$	$29.7 \cdot 10^{-2}$	$239.8 \cdot 10^{-2}$	$842.4 \cdot 10^{-2}$
Synthetic					
exp7	LEO	$1.7 \cdot 10^{-2}$	10^{-2}	10^{-2}	10^{-2}
exp8	Polar	$4.0 \cdot 10^{-2}$	$3.2 \cdot 10^{-2}$	$66.0 \cdot 10^{-2}$	$228.0 \cdot 10^{-2}$
exp9	MEO	$2.5 \cdot 10^{-2}$	$27.5 \cdot 10^{-2}$	$222.0 \cdot 10^{-2}$	$780.0 \cdot 10^{-2}$
DSP kernel					
exp10	LEO	$4.8 \cdot 10^{-2}$	$2.8 \cdot 10^{-2}$	$2.8 \cdot 10^{-2}$	$2.8 \cdot 10^{-2}$
exp11	Polar	$11.0 \cdot 10^{-2}$	$9.0 \cdot 10^{-2}$	$182.2 \cdot 10^{-2}$	$629.3 \cdot 10^{-2}$
exp12	MEO	$6.9 \cdot 10^{-2}$	$75.9 \cdot 10^{-2}$	$612.7 \cdot 10^{-2}$	$2152.8 \cdot 10^{-2}$

V. ALGORITHM EVALUATION

This section provides an evaluation of the algorithm and compares it with the related work described in Section II.

A. Experimental evaluation

The algorithm has been tested by simulating the behavior of 4 systems in the space environment, where recoverable faults are caused by radiations without destructive effect (Single Event Upset, *SEU*), and non-recoverable faults are caused by radiations' accumulation (Total Ionizing Dose, *TID*) and device aging (Time Dependent Dielectric Breakdown, *TDDB*, and Electromigration, *EM*) [2]. Ten-year long missions at three different orbits have been envisioned: Low-Earth Orbit (LEO), Polar orbit, and Medium Earth Orbit (MEO). We assume five years of Solar Minimum and five of Solar Maximum (characterized by recoverable faults rates $\lambda_{rec_{min}}$ and $\lambda_{rec_{max}}$, respectively), with two solar flares lasting one week and one hour ($\lambda_{rec_{week}}$ and $\lambda_{rec_{day}}$, respectively). The frequency λ_{rec} predicted for each system has been computed as $P_r \cdot \lambda_{SEU}$, where P_r and λ_{SEU}, that is the forecast SEU rates, are computed by using the data provided in [5]. The values are reported in Table I. Such data is the basis for the execution of 12 experimental conditions: 3 different orbits for each of the 4 considered systems. Also prediction of non-recoverable faults' rates has been performed to identify a rough timeline for simulating non-recoverable faults. A rough prediction is sufficient since the algorithm accuracy in identifying non-recoverable faults does not depend on their time occurrence. The rate of non-recoverable faults due to TID effect is computed by considering a predicted dose rate of 86.4 rad/day [7] and, by taking into account a Xilinx FPGA XCV1000, a TID of 60 krad [8]; we obtain $MTBF_{TID}$ = 694 days. Finally, we have considered the MTBFs due to device aging computed in [9], i.e., $MTBF_{TDDB}$ = 410 days and $MTBF_{EM}$ = 1460 days.

The algorithm has been implemented in a software version for the evaluation. A SystemC module implements the algorithm, activated by signals simulating faults in the system. Recoverable faults are simulated by generating an error in an IRA randomly chosen. A non-recoverable fault is simulated by activating a fault in an IRA and by keeping the signal active until it is classified as non-recoverable. The experimental sessions consisted of measuring the algorithm robustness and evaluating the condition for correct fault classification.

In the first experimental session, we varied K according to Equation 6. In a first evaluation phase, we analyzed the impact of the number of IRAs, by using $n = \{3, 5, 10, 15, 20, 30\}$, yielding a total of 72 experiments. The accepted probability of performing an erroneous classification of the recoverable faults has been set to $P_{mis_{r-nr}} = 0.005$. For each experimental condition and for each value of n, we considered the rate of recoverable faults mistakenly classified ($R_{mis_{r-nr}}$) as the fraction between the number of mis-classifications and the number of injected recoverable faults. Table II presents the results by reporting the average value of $R_{mis_{r-nr}}$ between the 12 conditions. It can be noted that, by decreasing K, more errors are performed by marking recoverable faults as non-recoverable. However, $R_{mis_{r-nr}} \simeq P_{mis_{r-nr}}$ when the algorithm operates in a safe region, which means for sufficiently high values of K. In a second evaluation phase, we have set n to a fixed value ($n = 5$) and have varied $P_{mis_{r-nr}}$. Table III reports the average value of $R_{mis_{r-nr}}$. Results are not reported for $P_{mis_{r-nr}} > 0.1$ since we would have $K < 1$, completely hindering a proper classification. It can be noted that, also by varying $P_{mis_{r-nr}}$, $R_{mis_{r-nr}} \simeq P_{mis_{r-nr}}$.

Table II
AVERAGE $R_{mis_{r-nr}}$ BY SETTING $P_{mis_{r-nr}}$=0.005

	n=3 (K=5)	n=5 (K=4)	n=10 (K=3)	n=15 (K=2)	n=20 (K=2)	n=30 (K=2)
avg $R_{mis_{r-nr}}$	0.005	0.004	0.104	0.065	0.04	0.049

Table III
AVERAGE $R_{mis_{r-nr}}$ BY SETTING N=5

	$P_{mis_{r-nr}}$=0.001 (K=5)	$P_{mis_{r-nr}}$=0.004 (K=4)	$P_{mis_{r-nr}}$=0.01 (K=3)	$P_{mis_{r-nr}}$=0.1 (K=2)
avg $R_{mis_{r-nr}}$	0	0.003	0.032	0.169

In the second experimental session, we evaluated the condition for correct fault classification by computing the maximum latency according to Equation 4, as follows:

$$lat \leq \frac{P_{mis_{nr-r}} \cdot MTBF_{rec}}{K-1} \tag{7}$$

For the values of K estimated in the previous experimental session, we computed the latency by setting sufficiently low values of the misclassification error probability, $P_{mis_{nr-r}} = \{0.005, 0.01, 0.05, 0.1\}$, and computing $MTBF_{rec} = \frac{1}{\lambda_{rec}}$, where λ_{rec} is the worst value between the environmental conditions. Table IV presents the results by reporting the average value of lat in the 12 conditions. Results show fault classification succeeds even with lat up to the order of hours (with a few exceptions in the range of tens of minutes), which is a very high value considering a typical fault-error relation, thus confirming the algorithm rationale.

Table IV
AVERAGE LATENCY [HOUR]

	$K=2$	$K=3$	$K=4$	$K=5$
$P_{mis_{nr-r}}$=0.005	1.54	0.77	0.51	0.39
$P_{mis_{nr-r}}$=0.01	3.09	1.54	1.03	0.77
$P_{mis_{nr-r}}$=0.05	15.45	7.72	5.15	3.86
$P_{mis_{nr-r}}$=0.1	30.90	15.45	10.30	7.72

B. Comparison with related work

Our algorithm has been compared with the contributes proposed in [4], [3], and [2]. No comparison with the work presented in [1] could be performed since no indication for setting the parameters of the algorithm was provided.

The algorithm in [4], targeted for microprocessors, is quite similar to ours. We verified that by setting the period for clearing the counters as MTBF of recoverable faults and the threshold as K, the algorithm in [4] has the same classification performance as ours. However, our approach only requires the use of one counter of length K, whereas, in [4], it is necessary to set as many counters as the IRAs, whose number can be very high; counters can considerably impact the area occupation of the Fault Classifier both for their number and for the hardening process. Moreover, the structure of our algorithm enables a formal and rigorous study of the effect of each parameter.

The approach in [3] can be considered equivalent to ours when K is set to 2, if we do not fix a MTBF (not so easily predictable). Such value of K is suitable when the number of IRAs is high enough (greater than 10, according to our approach), otherwise it is possible to fall into a mis-classification of a recoverable fault as non-recoverable.

Finally, we have implemented the algorithm in [2]. We have considered that a fault occurs relatively more frequently than the others when only the faulty IRA is registered in a buffer of length K. Moreover, we have computed the thresholds related to non-recoverable faults to analyze their absolute frequency. We have evaluated such algorithm by considering the number of fault observations required to classify the fault as non-recoverable, and by comparing it with K, as shown in Figure 2. Results show that, in the approach proposed in [2], the number of fault observations to recognize a non-recoverable fault can be very high, whereas in our approach is fixed.

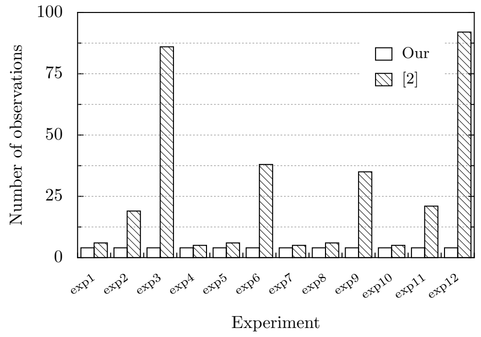

Figure 2. Comparison of the number of fault observations to recognize a non-recoverable fault in our approach and the one proposed in [2].

VI. Fault Classifier's reliable implementation

The module implementing the fault classification algorithm receives in input the `error_signals` from the IRAs and an `enable` signal specifying when the error signals are to be considered valid (the Fault Classifier is disabled during the recovery phase). We assume that on the error signals the IRAs adopt Two-Rail Code (TRC) [10], whereas the enable signal, generated within the Reconfiguration Controller, is not encoded since the presence of faults in the overall controller is revealed by an error signal generated by the controller itself. The Fault Classifier generates three outputs: `fault_identified` to signal whether a fault has been detected, `fault_type` to specify whether it is recoverable or not, and `faulty_ira` to specify the IRA where the fault has occurred. The input signals are used, together with the following local information, to classify the fault: the IRA identified as faulty the last time an error has been detected (`last_ira`), the number of consecutive errors detected on the IRA specified in the above mentioned register (`error_counter`), and the parameter K, specifying the threshold to classify a fault as non recoverable. The resulting structure of the nominal Fault Classifier module is reported in Figure 3; it is possible to identify the internal signals used to characterize the situation, `error_detected` (when an IRA is identified as faulty), `same_ira` (whether the IRA identified as faulty is the same as the last IRA detected as faulty), and `max_count` (whether the number of consecutive fault detections in the same IRA has reached the pre-defined threshold K).

As discussed in Section III, the implementation of an overall reliable system requires the hardening of the Fault Classifier, such that it will be possible to detect faults affecting any portion of the controller thus triggering a suitable recovery phase. We deem sufficient to provide fault detection properties only, since, when a fault is detected, a reconfiguration (and reset) of the controller would only cause a minor delay in classifying a fault as not-recoverable. In fact, in case of recoverable faults, the most frequent ones, a loss of information has not negative effects, otherwise an increased number of observations (at most $2K$-1 instead of K) is required to obtain the correct diagnosis; given the actual values used for K, the negative effect of a longer observation is quite limited, since, as explained in Section IV, the frequency for observing errors due to the same non-recoverable fault is high enough. Hence, even the most critical registers for discriminating between faults, i.e., the ones related to the `error_counter` and the `last_ira`, when reset, do not compromise seriously the correctness of the classification. Nevertheless, for the sake of completeness, implementations achieving fault tolerance properties have been investigated. In particular, our analysis took into account the following hardened solutions: i) an implementation that applies Duplication With Comparison (DWC) on the module, ii) a Self-Checking (SC) implementation based on the application of error detection codes, iii) a Self-Checking implementation coupled with Triple Modular Redundancy (TMR) applied to the most critical registers (SC+), and iv) an implementation where TMR is applied to the entire system with a fine granularity, by following the approach offered by Xilinx's TMRTool (X-TMR) [11].

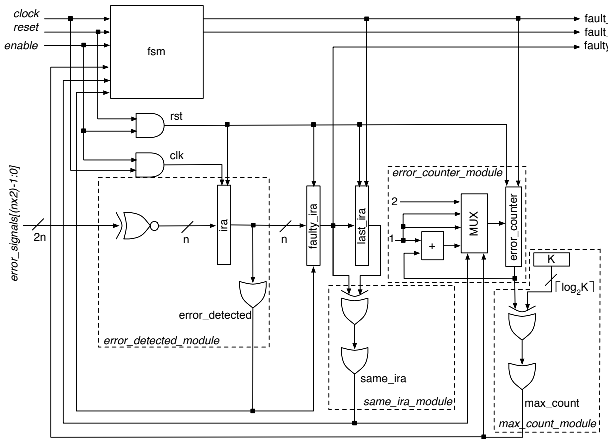

Figure 3. Fault Classifier structure.

A. Duplication With Comparison: DWC

In this implementation, two replicas of the Fault Classifier receive the same inputs and their outputs are compared by Two-Rail Code Checkers (TRCCs) for mismatch. The design is straightforward and no specific issues arise in the implementation of the solution, provided the hierarchy is guaranteed during the synthesis process. The tree of TRCCs is applied to the module's primary outputs, amounting to a total of $n + 2$ lines, being n the number of IRAs in the monitored FPGA.

B. Self-Checking via Error Detecting Codes: SC

For the self-checking implementation, three classes of elements that constitute point of failures have been identified: i) the states of the FSM, ii) the internal variables of the datapath, and iii) the outputs. The analysis of the number of states of the control FSM and the values stored in the other registers allowed us to select the 1-hot code, to protect all the registers of the module, thus adopting a very well known (and thus easily implementable) encoding. In particular, since one of the monitored IRAs may be faulty at most, based on the adopted working hypotheses, the content of the `faulty_ira` and `last_ira` (from Figure 3) naturally exploits a 1-hot encoding. A similar analysis holds for register `ira`, buffering the input error signals; eventually, it might contain an all 0s configuration, when no fault is detected on the monitored IRA. Therefore, by concatenating the complement of the `error_detected` signal with `ira`, an immediate, cost-free 1-hot encoding is obtained. For the other internal variables (state register and `error_counter`), since the number of states is limited, the code is not too expensive (the typical limitation of this code). The primary outputs are single lines (`fault_identified` and `fault_type`), encoded with a Two-Rail Code, whereas `faulty_ira` is the encoded content of the register.

Table V
AREA OCCUPATION (SLICES AND FFS – IN PARENTHESIS) OF THE FAULT CLASSIFIER IMPLEMENTATIONS AND RELATED OVERHEAD.

Solution	K=2 (n=15)		K=3 (n=10)		K=4 (n=5)		K=5 (n=3)	
	area	over.	area	over.	area	over.	area	over.
Nominal	86 (52)	-	60 (36)	-	40 (22)	-	29 (16)	-
DWC	176 (104)	105%	120 (72)	100%	81 (44)	102%	58 (32)	100%
SC	120 (53)	40%	96 (39)	60%	57 (25)	42%	47 (20)	62%
SC+	133 (53)	55%	107 (39)	78%	67 (25)	67%	58 (20)	100%
X-TMR	406 (201)	372%	283 (138)	372%	186 (81)	365%	141 (57)	386%

Table VI
AREA OCCUPATION (SLICES AND FFS – IN PARENTHESIS) OF THE FAULT CLASSIFIER MODULES FOR THE NOMINAL VERSION AND THE SELF-CHECKING ONE.

Module	K=2 (n=15)		K=3 (n=10)		K=4 (n=5)		K=5 (n=3)	
	Nom.	SC	Nom.	SC	Nom.	SC	Nom.	SC
fsm	9 (4)	14 (6)	9 (4)	14 (6)	9 (4)	14 (6)	9 (4)	14 (6)
error_detected_module	27 (15)	27 (15)	18 (10)	18 (10)	13 (10)	13 (10)	8 (6)	8 (6)
faulty_ira	15 (15)	15 (15)	10 (10)	10 (10)	5 (5)	5 (5)	3 (3)	3 (3)
last_ira	15 (15)	15 (15)	10 (10)	10 (10)	5 (5)	5 (5)	3 (3)	3 (3)
same_ira_module	4 (0)	8 (0)	3 (0)	4 (0)	2 (0)	2 (0)	1 (0)	2 (0)
error_counter_module	5 (3)	3 (2)	5 (3)	5 (3)	5 (3)	6 (4)	5 (3)	8 (5)
max_count_module	1 (0)	1 (0)	1 (0)	1 (0)	1 (0)	1 (0)	1 (0)	2 (0)

C. SC and TMR: SC+

A first fault tolerant implementation has been defined by protecting the most critical registers with TMR, using a feedback loop to propagate the correct value to the protected registers, thus avoiding any glitch in the stored values, especially usefull for the case of non-recoverable faults.

D. X-TMR

As a final alternative, we applied fault tolerance to the overall module with a fine granularity. We implemented the solution achieved by using X-TMR, that applies TMR to the entire system with a fine granularity. Unlike traditional TMR, X-TMR triplicates i) all inputs, including clocks and throughput logic, ii) feedback logic, and iii) all ouputs; it inserts majority voters on feedback paths, and minority voters on outputs to detect and disable incorrect output paths. Such solution is the one usually adopted for reconfigurable FPGAs in high-radiation environments.

E. Cost Analysis

The proposed implementations are compared in Table V with respect to their overheads, in terms of area occupation (number of used slices and flip flops); the selected device is a Xilinx Virtex-II XC2V1000. Each implementation has been analyzed by varying K and n as in the previous section, to derive a trend in the implementation costs. As expected, as the number of monitored IRAs n increases, the size of the controller increases, in all nominal and hardened versions. The most expensive solution is the one obtained by using X-TMR, entailing overheads around 370%, as expected. For the other implementations, the incidence of the fault detection/tolerance added functionality is within the foreseen, typical bounds, keeping the final hardened implementation within acceptable costs, that is less than 2% of the entire FPGA resources. In particular, overheads range from 40% and about 100%, based on the different solutions, allowing the designer to select the solutions s/he deems more interesting.

The self-checking implementation allows us to have a competitive solution, characterized by acceptable costs and benefits in terms of reliability. It has been compared to the nominal solution in Table VI, in terms of area occupation of the modules composing the Fault Classifier (see Figure 3); not all modules entail an increment in cost, in particular `error_detected_module` and the registers `faulty_ira` and `last_ira`, that do not require modifications in the self-checking version. Alternative solutions, with acceptable but higher costs, do not entail significative benefits in terms of additional reliability.

VII. CONCLUSION

In this paper, an algorithm for the distinction between recoverable and non-recoverable faults has been designed, and its integration into an overall reliable system has been proposed. Experimental results prove the effectiveness of the algorithm, also with respect to the related work. Moreover, different reliable implementations of the algorithm have been analyzed.

Future development includes a fault injection campaign to experimentally evaluate the achieved reliability, and the integration of the module in the Reconfiguration Controller, exploiting the fault classification information to apply the suitable recovery strategy.

REFERENCES

[1] C. Bolchini, L. Fossati, D. M. Codinachs, A. Miele, and C. Sandionigi, “A reliable reconfiguration controller for fault-tolerant embedded systems on multi-FPGA platforms,” in *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 2010, pp. 191–199.

[2] C. Bolchini and C. Sandionigi, “Fault classification for SRAM-based FPGAs in the space environment for fault mitigation,” *IEEE Embedded Systems Letters*, vol. 2, no. 4, pp. 107–111, 2010.

[3] S. Pontarelli, M. Ottavi, V. Vankamamidi, G. C. Cardarilli, F. Lombardi, and A. Salsano, “Analysis and evaluations of reliability of reconfigurable FPGAs,” *J. Electronic Testing*, vol. 24, no. 1-3, pp. 105–116, 2008.

[4] F. A. Bower, D. J. Sorin, and S. Ozev, “A mechanism for online diagnosis of hard faults in microprocessors,” in *Int. Symp. on Microarchitecture*, 2005, pp. 197 – 208.

[5] K. S. Morgan, “SEU-induced persistent error propagation in FPGAs,” Ph.D. dissertation, Brigham Young University, 2006.

[6] C. Bolchini and A. Miele, “Design space exploration for the design of reliable SRAM-based FPGA systems,” in *Proc. IEEE Int. Symp. on Defect and Fault-Tolerance in VLSI Systems*, 2008, pp. 332 – 340.

[7] R. H. Maurer, M. E. Fraeman, M. N. Martin, and D. R. Roth, “Harsh environments: space radiation environment, effects and mitigation,” *Johns Hopkins APL Technical Digest*, vol. 28, no. 1, 2008.

[8] V. Bocci, M. Carletti, G. Chiodi, E. Gennari, E. Petrolo, A. Salamon, R. Vari, and S. Veneziano, “Radiation test and application of FPGAs in the ATLAS level 1 trigger,” in *Workshop on Electronics for LHC Experiments*, 2001.

[9] S. Srinivasan, R. Krishnan, P. Mangalagiri, Y. Xie, V. Narayanan, M. J. Irwin, and K. Sarpatwari, “Toward increasing FPGA lifetime,” *IEEE Trans. Dependable and Secure Computing*, vol. 5, no. 2, pp. 115 – 127, 2008.

[10] D. Nikolos, “Self-testing embedded two-rail checkers,” *J. Electronic Testing, Theory and Applications*, vol. 12, no. 1-2, pp. 69–79, 1998.

[11] Xilinx Inc, *Xilinx TMRTool*, 2006. [Online]. Available: http://www.xilinx.com/esp/mil_aero/collateral/tmrtool_sellsheet_wr.pdf