

# SEU Mitigation for SRAM-Based FPGAs through Dynamic Partial Reconfiguration

Cristiana Bolchini, Davide Quarta, Marco D. Santambrogio

Politecnico di Milano – Dip. Elettronica e Informazione

P.zza L. da Vinci, 32 – 20133 Milano – Italy

`firstname.lastname@polimi.it`

## 1 Introduction

The paper is organized as follows. Section 2 depicts the working scenario by recalling the basic information on the adopted fault model and by giving an overview of other approaches from the literature. The proposed methodology for achieving mitigation capabilities with respect to the considered faults is thoroughly discussed in Section 3, while the experimental results are reported and discussed in Section ???. The last section draws some conclusions.

## 2 Background

### 2.1 Fault Model

The impact of radiation and alpha particles on both combinational and sequential logic causes Single Event Transient (SET) and Single Event Upset (SEU) faults to mine the behavior of the implemented device on both Application Specific Integrated Circuits (ASICs) and FPGAs [?, ?]. The effect of these types of faults is a pulse on a signal, that may get latched in a register thus producing an erroneous value, or a change of the value stored in a SRAM cell (*bit-flip*). In both situations the fault is referred to as *soft error*, and it is a transient fault that does not cause any permanent damage in the circuit.

SRAM-based FPGAs are particularly affected by this class of faults, due to the presence of a high number of SRAM memory cells, storing both user information (the data being processed and manipulated) and configuration information. More precisely, the fault may corrupt the content of a user register storing a data value, thus generating an erroneous result, or the content of a configuration memory

element, thus modifying either the functionality performed by one of the Look-Up Tables (LUTs) or the routing between Combinational Logic Blocks (CLBs) also altering the resulting functionality. While in the former situation a re-computation can solve the problem, a re-configuration of the FPGA is needed in the latter situation, requiring the downloading of the correct bit-stream on the device, to recover from the fault.

## 2.2 Related Work

Several approaches have been developed for detecting faults in FPGAs [?, ?, ?, ?, ?, ?, ?], most of them based on an off-line testing of the device.

For instance, with Built-In Self Test (BIST) [?], the FPGA is loaded with small testing circuit, which is restricted to a specific physical region of the device. This circuit, opportunely configured, can test a portion of the FPGA, therefore, by re-configuring it multiple times, the entire FPGA is cyclically analyzed. However, off-line testing can not be applied in all those scenarios which are highly fault-sensitive and characterized by hard real-time constrains, such as in space and avionic applications. In these cases it is necessary to act promptly, trying to minimize the fault detection latency introduced by an off-line technique, in order to avoid the failure of the whole system (classified as mission-critical).

To solve this problem [?, ?] propose a methodology for on-line testing of FPGAs that allows the detection of those faults which are located in the off-line portion of the device, i. e., in those sections that are not being used while normal operation is being performed by other parts of the FPGA. These methods use Self-Testing Areas (STARs) to test specific off-line sections of the device, then the STARs are moved across the FPGA in order to cover all the off-line sections. The limitation of this class of approaches is their inability to protect the entire FPGA.

Two techniques most commonly adopted to detect and also correct errors caused by transient fault are readback and scrubbing [?, ?], both of them are based on mathematical analyses which try to estimate the upset rate. *Readback* consists of the periodical reading of the configuration memory of the FPGA to detect, through a small bit manipulation approach [?], when an upset of the configuration memory has occurred. *Scrubbing* is a simple and straightforward method to correct SEUs; it performs a periodic reloading (usually with the same frequency of readback) of the entire configuration bitstream, independently of the fact that an error has occurred or not. Scrubbing introduces in the system substantially less overhead than readback, nevertheless both these methods rely on the determination of an appropriate readback/scrub rate, which is a rather difficult parameter to be determined.

In this scenario, a more efficient SEU mitigation strategy is the one that combines Triple Module Redundancy (TMR) and scrubbing [?, ?]. The TMR mitiga-

tion scheme uses three identical functional blocks performing the same operation, whose outputs are checked by a majority voter. In this way if a SEU occurs, TMR can guarantee the correct functionality of the system and activate the scrubbing procedure, requesting a refresh of the configuration memory of the device before more errors accumulate and overcome the TMR tolerance property. Indeed, due to the significant overhead of the overall circuit (consisting of three replicas and the voter) TMR is not an affordable approach for numerous systems; furthermore, it may become difficult to define the optimal design of TMR logic to ensure the appropriate robustness [?].

In this paper we presents a new methodology for SEU detection and correction, which can guarantee through dynamic partial reconfiguration, fast recovery of the system, robustness and less overhead than the TMR one.

### 3 Proposed approach

The basic idea behind the proposed approach is to try to identify a design methodology that can be used to design reliable system using different characteristics, i.e. partial dynamic reconfiguration, of the device, such as FPGAs, used to implement the final solution.

Different techniques have been proposed in literature, as shown in Section 2.2, but the proposed approach differs from them for several aspects:

- 
- *Design partitioning oriented to the reconfiguration:*  
since FPGAs have strong reconfiguration constraints, such the one introduced by 1-D placement for the VirtexII and VirtexIIPro, we proposed a partitioning phase able to explore different systems solution trying to optimize the system *specification* to minimize the reconfiguration time without losing in reliability
- *Single design, multiple techniques:*  
FPGAs have the ability to reconfigure themselves but, on the contrary it is necessary to remember that the designer has to work with a fixed amount of resources. Reconfiguration is used to implement different algorithms and systems on the same chip but different implementations imply different resource usages. This scenario shows that it is impossible to define an *always-true* technique able to guarantee *always the best* reliable design solution. Therefore we propose an approach able to use different techniques to design a reliable system component to obtain the best system solution that can fit into the chosen device.

•

### 3.1 Partial Dynamic Reconfiguration

There are different ways that can be used to design a reconfigurable architecture but, since the reconfiguration introduce considerable time overhead, it is necessary to try to identify a design techniques able to minimize as much as possible the reconfiguration time.

The reconfiguration time is proportional to the dimension of the bitstream used to configure the device, that means that reducing the amount of data used to define the partial reconfiguration bitstream implies less time for the reconfiguration. In this scenario the system design plays a key role in the definition of the best components identification as briefly introduce in the *design partitioning oriented to the reconfiguration* described in Section 3. On one side we would like to have components as small as possible to obtain smaller reconfiguration bitstream but on the other side, remembering that the FPGAs as *area* reconfiguration constraints, we have to try to use in the best way the area that we have to use to design a reconfigurable component. A small components, during the system design, might not imply a smaller area compared with a bigger one because of the area constraints imposed by the reconfiguration.

### 3.2 Solution Space Exploration

## 4 Experimental Results

In this section a set of experimental results where the proposed methodology has been applied are presented.

### 4.1 Working scenario

The test case used to evaluate the efficiency of the proposed approach is the *Noekeon* algorithm ([?]). This cryptographic system is a block cipher with a block and key length of 128 bits. It is an iterated cipher defined using an iterated computation of a simple round transformation which is carried out by four functional elements: **Theta**, **Gamma**, **Pi1**, **Pi2**. The identification of the macro-modules, representing the TSC-areas, has followed this natural partitioning. As shown in Figure ??, we have identified three TSC-areas each of the ones differs from the others for the specific transformation preformed, whatever it useful to observe how our methodology is highly flexible as regards the identification of the TSC-areas and the definition of the appropriate technique of concurrent error detection. In

our tests we decide to use a duplication and comparison technique supported by a dual-rail encoding in **Theta** and **Gamma**, while we developed an ad-hoc checker for **Pi** which exploits the duality of the functions performed by **Pi1** and **Pi2**.

Figure 1: .

In order to build up a fault-tolerant system which follows our methods, we exploited the Ucf Builder and Analyzer (UBA) software as the tools for the automatic generation of area constrains of a module-based system. Moreover we used Xilinx Floorplanner [ ] for the graphical representation of the obtained results.

One of the most important parameters which has been analyzed it is the resources optimization index. This data is defined as the ratio between the number of requested slices to implement a specific component into the system and the number of used slices to constrain the same component. Once that specific area constrains with UBA has been defined, a number of additional resources are reserved for each TSC-area in order to satisfied those constraints. For this reason is useful to find out how many additional resources need to be used to guarantee the correctness of the modular-based design. We also estimated the recovery timing of each module and of the whole system by adopting a linear cost model as to the dimension of the configuration bitstream. The combination of these two values allows us to say how much faster is the configuration compared to a pure scrubbing with no CED, and how many resources we saved compared to TMR approach.

## 4.2 Discussion

We tested our model on two different FPGAs: the Virtex-II Pro VP30 and the Virtex-II Pro VP7; results are reported in Tables ??, ??, ?? and ??.

Virtex-II Pro VP7 resources optimization index			
Module	Requested Slices ( $S_R$ )	Occupied Slices ( $S_O$ )	$S_R/S_O$
ThetaBlk	447	640	0.698437
GammaBlk	255	640	0.398437
PiBlk	83	320	0.25937
<b>Tot.</b>	2225	3466	0.6419

Table 1: .

It can be noticed that a smaller dimension of a TSC-block, cause a worsening of the resources optimization index value. That is why the definition of particular area constrains reserves additional spare resources, which are in percentage more

Virtex-II Pro VP7 time analysis	
Module	Recovery Timing ( $T_s =$ scrubbing time)
ThetaBlk	$0.5614 \cdot T_s$
GammaBlk	$0.5614 \cdot T_s$
PiBlk	$0.2807 \cdot T_s$
<b>Tot.</b>	$3.0403 \cdot T_s$

Table 2: .

Virtex-II Pro VP30 resources optimization index			
Module	Requested Slices ( $S_R$ )	Occupied Slices ( $S_O$ )	$S_R/S_O$
ThetaBlk	447	640	0.698437
GammaBlk	255	640	0.398437
PiBlk	83	640	0.129687
<b>Tot.</b>	2225	2962	0.7512

Table 3: .

Virtex-II Pro VP30 time analysis	
Module	Recovery Timing ( $T_s =$ scrubbing time)
ThetaBlk	$0.5614 \cdot T_s$
GammaBlk	$0.5614 \cdot T_s$
PiBlk	$0.5614 \cdot T_s$
<b>Tot.</b>	$2.5982 \cdot T_s$

Table 4: .

significant wherever the dimension of a TSC-block is significantly smaller than the minimum reservedly area (which is a four slice rectangle).

If we focus our attention on the recovery timing of each TSC-block, we can claim that however the initialization of the device is 2.6 up to 3 times slower than the non redundant one (but anyway faster than the TMR approach), the recovery of one of the TSC-block is significantly faster than a classical scrubbing method.

Comparing the results of VP7 and VP30 architecture, we can also notice that in spite of the resources optimization index improvement, there is no collateral evidence in the recovery timing of the TSC-block, but actually a worsening of a factor 2 of the Pi - block recovery time. This can be attributed to a definitely low value of the relative resources optimization index. As stated in . . .

## **5 Conclusions and future work**