Reliability Engineering and System Safety 151 (2016) 1–9

Contents lists available at ScienceDirect

Reliability Engineering and System Safety

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

Synchronization of faulty processors in coarse-grained TMR protected partially reconfigurable FPGA designs

U. Kretzschmar [a], J. Gomez-Cornejo [a], A. Astarloa [a], U. Bidarte [a,*], J. Del Ser [b,c]

[a] Department of Electronics, University of the Basque Country UPV/EHU, 48013 Bilbao, Bizkaia, Spain
[b] OPTIMA Area, TECNALIA Research & Innovation, 48160 Derio, Bizkaia, Spain
[c] Department of Communications Engineering, University of the Basque Country UPV/EHU, 48013 Bilbao, Bizkaia, Spain

ARTICLE INFO

Article history:
Received 31 August 2015
Received in revised form
27 November 2015
Accepted 23 December 2015
Available online 16 February 2016

Keywords:
Reliability
TMR
FPGA
Synchronization
Fault-recovery
Processor

ABSTRACT

The expansion of FPGA technology in numerous application fields is a fact. Single Event Effects (SEE) are a critical factor for the reliability of FPGA based systems. For this reason, a number of researches have been studying fault tolerance techniques to harden different elements of FPGA designs. Using Partial Reconfiguration (PR) in conjunction with Triple Modular Redundancy (TMR) is an emerging approach in recent publications dealing with the implementation of fault tolerant processors on SRAM-based FPGAs. While these works pay great attention to the repair of erroneous instances by means of reconfiguration, the essential step of synchronizing the repaired processors is insufficiently addressed. In this context, this paper poses four different synchronization approaches for soft core processors, which balance differently the trade-off between synchronization speed and hardware overhead. All approaches are assessed in practice by synchronizing TMR protected PicoBlaze processors implemented on a Virtex-5 FPGA. Nevertheless all methods are of a general nature and can be applied for different processor architectures in a straightforward fashion.

1. Introduction

The great flexibility, high achievable system speeds and the large number of available design resources make SRAM-based FPGAs a good choice for a wide variety of electronic designs. Especially the low non-recurring engineering cost of FPGA based designs, where the high initial expenses of ASICs or ASSPs cannot be compensated by very high production volumes. Such designs realized using FPGAs have been shown to outperform standard CPUs [1–3]. Nevertheless, modern FPGA implementations typically utilize soft- or hard-core processors to enable an efficient implementation of the overall system, as well as to take advantage of existing hardware modules [4,5].

An additional advantage of SRAM-based FPGAs is Dynamic Partial Reconfiguration (DPR), which allows designers to change parts of the implemented design while keeping the overall system operational. This time-multiplexing method of FPGA resources can be used in a variety of ways such as e.g. adapting a cache architecture to specific application requirements [6], implementing a multi-protocol network switch or enabling software defined radio [7].

Due to the advantages that it provides, the number of fields of application that make use of the FPGA technology continues growing. In certain of these fields, in which a faulty operation can jeopardize human life or the integrity of valuable technology, high levels of reliability are required. Railway [8,9], automotive [10,11] and space [12,13] systems are remarkable examples. In such systems robustness is one of the most relevant aspects and the high susceptibility of SRAM-based FPGA technology to Single Event Effects (SEE) becomes a crucial factor. SEE faults can be caused by high energy particles impacting on the FPGA [14,15]. The most critical among all SEEs are Single Event Upsets (SEUs) [16], specially when they affect the configuration memory [17]. Unlike ASICs where the interconnections and the logic elements on the die are fixed, FPGAs use a configuration bitstream for defining the function of the configurable logic elements or the interconnection matrix. Configuration memory upsets can consequently alter the implemented design by changing its elements. SEUs in user memories such as FPGA internal block RAM (BRAM) and flip-flops are not as critical for the overall system, because the SEU occurrence rate per bit of BRAM is in the same order of magnitude as for the configuration memory. By contrast, a FPGA typically has one order of magnitude more configuration memory than BRAM [18–20].

To avoid this issue, methods for mitigating the susceptibility of FPGA designs against SEUs have been thoroughly investigated in the literature by resorting to Error Correction Codes (ECC) [21,22]

* Corresponding author.
E-mail address: unai.bidarte@ehu.eus (U. Bidarte).

http://dx.doi.org/10.1016/j.ress.2015.12.018
0951-8320/

or Duplication With Comparison (DWC) [23,24]. In particular, the so-called Triple Modular Redundancy (TMR) method results to be the most frequently addressed by both industry and academia in diverse technological architectures [25]. The rationale for this trend is threefold: (1) the possibility of fault masking by implementing the process of *voting*; (2) the method of scaling the TMR protection by changing its granularity [26]; and (3) the availability of tools allowing for a completely automated TMR generation [27]. TMR is typically combined with configuration scrubbing [28–30], a process which corrects configuration memory upsets. In this combination TMR enables the design to continue operating correctly in presence of faults, whereas scrubbing avoids the accumulation of multiple faults.

Although the combination of TMR and scrubbing is the fault handling strategy recommended by the FPGA vendors [31], several recent publications have proposed a step beyond this established method. This new strategy is based on coarse-grained TMR where each triplicated instance of the design is partially reconfigurable. Any error in one of such instances can be corrected by reconfiguring the corresponding module [32–34]. By providing distinct implementations of the same instance, this solution can also repair instances where the corresponding FPGA region has suffered a permanent error. Notwithstanding the great extent to which the TMR setup and the partial reconfiguration aspect of this approach have been studied in the related literature, the required synchronization of the reconfigured instance has been either neglected or it has been analyzed in an incomplete manner.

This work addresses this scarcity of investigations around synchronization in TMR systems by proposing, implementing and evaluating four different synchronization methodologies. The four approaches span a broad spectrum of possible alternatives from minimal hardware overhead to completely hardware-based synchronization. This allows balancing the trade-off between implementation cost and synchronization speed, depending on the requirements of the target application at hand. The performance of the proposed techniques is verified and compared to each other on the PicoBlaze [38] processor. However, all four approaches are of general nature and can easily be migrated to other processor architectures. These papers synchronization methods are furthermore not restricted to a set-up implementing TMR and DPR. They are applicable to any TMR protected processor system to recuperate a processor element, which was forced out of sync by a SEE.

The remainder of this work is structured as follows. Section 2 surveys recent advances on fault protection using TMR and DPR. Next, Section 3 proposes the aforementioned four different synchronization approaches, whereas implementation details are outlined in Section 4. Practical results are presented in Section 5, and finally Section 6 ends the paper by drawing some concluding remarks.

2. Fault tolerant systems based on DPR and TMR

The combination of Triple Modular Redundancy and Dynamic Partial Reconfiguration is a very attractive solution for the implementation of fault tolerant systems. Among many different possible implementation forms of TMR, the so-called coarse-grained TMR [26] implements three instances of the same module and a final voter. This method provides a slightly lower protection than fine-grained TMR [26,39], where modules needing protection are broken into smaller parts. Some fine-grained TMR approaches even provide synchronization of the three modules [40,27]. However, coarse-grained TMR is ideal for its combination with DPR as it results in a small amount of reconfigurable partitions. A block diagram of this combination is depicted in Fig. 1.

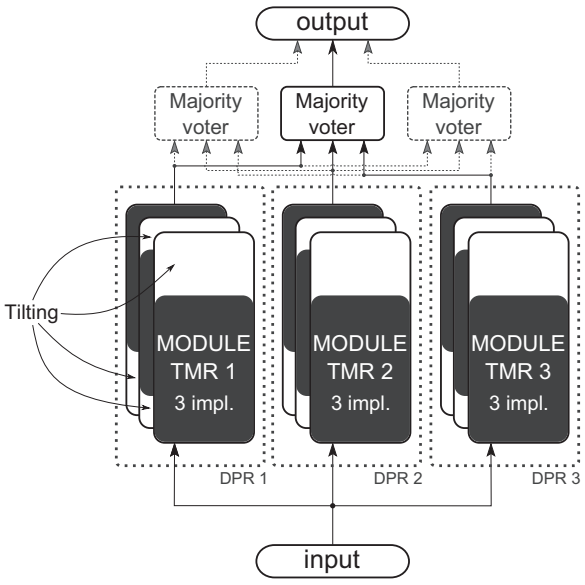

Fig. 1. Typical architecture of the combination of TMR and DPR.

The above figure summarizes the high level architecture proposed in a number of recent publications [32–37]. Three instances of the same module are placed in three partially reconfigurable areas. Configuration errors in one of these modules can consequently be repaired by reloading the bitstream of the faulty module by using partial reconfiguration once a voting step has identified the module providing the incorrect input. This voting step can be implemented either as a single voter or as a triplicated voter as suggested in Fig. 1. In addition to this protection against configuration errors, the architecture in Fig. 1 accommodates the idea of *tilting* [24].

In tilting the three reconfigurable regions are enlarged to enable different implementations of the same logic, leaving selected parts (the white spaces marked by arrows) of the partially reconfigurable area unused. This strategy provides a means to avoid permanent errors: if e.g. one of the reconfigurable regions fails to operate correctly due to a permanent error, reloading the same partial bitstream will not recover this corresponding instance. But if a different tilted implementation of the module is loaded, the region can be repaired if the tilted bitstream does not use the region with the permanent fault.

Nevertheless, in the majority of cases a reconfiguration by itself does not suffice for recovering a faulty TMR instance. If this instance features some kind of internal state, this state needs to be synchronized to the other instances after the reconfiguration. In this line of reasoning, a synchronization method valid for small Finite State Machines (FSM) is proposed in [36] introducing the notion of *state prediction*. State prediction suggests that each FSM has (at least) one state to which the machine always returns after a finite amount of time. Therefore, by setting the FSM of a reconfigured module to this state it is possible to wait for the other two instances to reach this point during their normal operation, and thereafter continue seamlessly operating with all three instances. It should be clear that this method is obviously only applicable for small state machines with a reduced number of possible states.

Synchronization has also been considered in [35], where a fault tolerant MicroBlaze architecture using TMR and DPR was presented, as summarized in Fig. 2.

In this work three MicroBlaze processors sharing peripherals and memory were implemented in partially reconfigurable areas. The peripherals and the shared memory are protected by TMR and ECC, respectively. Sharing one memory between the three processors