# Combining Duplication, Partial Reconfiguration and Software for On-line Error Diagnosis and Recovery in SRAM-based FPGAs

Anargyros Ilias, Kyprianos Papadimitriou and Apostolos Dollas
*Department of Electronic and Computer Engineering*
*Technical University of Crete*
*GR73100 Chania, Crete, Greece*
{*ailias, kpapadim, dollas*}*@mhl.tuc.gr*

***Abstract*—SRAM-based FPGAs are susceptible to Single-Event Upsets (SEUs) in radiation-exposed environments due to their configuration memory. We propose a new scheme for the diagnosis and recovery from upsets that combines i) duplication of the core to be protected, ii) partial reconfiguration to reconfigure the faulty part only, and iii) hardcore processor(s) for deciding when and which part will be reconfigured; executing the application in software instead of hardware during fault handling; and controlling the reconfiguration. A hardcore processor has smaller cross section and it is less susceptible than reconfigurable resources. Thus it can temporarily undertake the execution during upset conditions. Real experiments demonstrate that our approach is feasible and an area reduction of more than 40% over the dominant Triple Modular Redundancy (TMR) solution can be achieved at the cost of a reduction in the processing rate of the input.**

***Keywords*-Single-Event Upsets; Duplication; Partial reconfiguration; Hardcore Processor;**

## I. Introduction

The dominant method for error detection and recovery is the Triple Modular Redundancy (TMR) [1], [2]. It uses three replicas of the system and a voter to identify the correct result amongst the three ones with respect to the majority vote. When combined with partial reconfiguration it is claimed to have zero error rate [3]. TMR is reported to have at least 3.2x resource utilization cost [4], while clock frequencies over 100 MHz in a Virtex-II FPGA are difficult to achieve [5]. This scenario is rather optimistic as in other works even for small filters resource utilization from 4x to 5x has been reported [1]. Another redundancy method with smaller resource overhead is the Duplication With Compare [6], which uses two replicas and a comparator to check their outputs.

Our aim is to form a scheme with near the same performance with TMR by paying less overhead in resources and without sacrificing reliability. It lies between the DWC and TMR, as the hardware cores are duplicated and a third model of the application executes in software. We target real-time domain in which the FPGA needs to sustain the processing of continuous input data. Upon a failure, real-time processing is undertaken by the software processor, which executes with less performance than the hardware counterpart. If the software processing rate is slower than the input rate a FIFO is needed to avoid losing data.

## II. Architectural Concept and Analysis

First, we need to clarify some terms we use throughout the paper. The operation of fault handling is distinguished in the detection phase, i.e. finds that an error occurred, the diagnosis phase, i.e. identifies the faulty part, and the correction or recovery phase, i.e. corrects the fault. Also, we are coining the term "restoration time" being the time elapsed from a fault detection until the hardware core begins processing again the input data.

The method we employed for the detection of faults is the Duplication With Compare (DWC) [6]. Currently, the architecture contains two hardcore processors (A and B) communicating through a shared memory. The scope of the processor A is threefold: checking a XOR gate for fault detection, informing the processor B when a fault occurs, and reconfiguring the faulty part. The main task of the processor B is the execution of the software model of the application upon a failure[1].

For the implementation of our scheme we use the Xilinx PR module-based design flow. The replicas of the core are designed as partially reconfigurable modules (PRM). They are placed into predefined regions of the FPGA called partially reconfigurable regions (PRR). Upon a fault occurrence the bitstream of the corresponding PRM is loaded to reconfigure a PRR. During reconfiguration the PRR is isolated so as to leave intact the remaining part of the FPGA by controlling special communication primitives called bus macros - the I/Os of the PRR - placed at the edge of the boundaries of PRRs.

Two different designs can take effect according to our scheme. The architecture is the same for both designs. Their main difference lies in whether the two replicas are placed in the same PRR or in two separate PRRs. In the first case, the so called 1-PPR design, the two replicas are implemented as one PRM that is loaded into one PRR. In the second case, the so called 2-PRRs design, each replica is created as one PRM each of which is loaded into a separate PRR. Only the operation of fault detection is the same for both designs; the other operations differ:

- *1-PRR design (No diagnosis is made):* Once a fault is detected the processing in the HW cores is halted.

[1]With slight modifications one processor only can be used thus eliminating the overhead of communication between the processors. Currently our proof-of-concept system operates with two processors.

Table I
MAIN DIFFERENCES BETWEEN THE OPERATIONS CARRIED OUT IN THE 1-PRR AND 2-PRRS DESIGNS UPON A FAILURE.

| | 1-PRR design (the two replicas are placed in one PRR) | 2-PRRs design (one replica per PRR) |
|---|---|---|
| In the correction phase | the entire PRR carrying both HW cores is reconfigured (no diagnosis was made). | only one PRR carrying the corrupted HW core is reconfigured (the SW diagnosed it). |
| The data processing continues | by the SW model in processor B. | by the uncorrupted HW core. |

Then, the PRR carrying both HW cores is reconfigured and at the same time the processor B is instructed to continue processing the input data. After reconfiguration completes, the HW cores undertake again the data processing.

- *2-PRRs design (Diagnosis is made):* Once a fault is detected, the processing in the HW cores is halted. Then, the processor B is instructed to process a specific amount of the next input data and at the same time the two HW cores process the same input data. The results are stored in registers accessible by the processor A, and the latter compares them in order to identify the HW core that disagrees with the SW model. Then, it instructs the uncorrupted HW core to resume its operation and reconfigures the PRR carrying the corrupted HW core. Once reconfiguration is ended, the system enters its normal operation.

Table I summarizes the differences amongst the two designs. Their effectiveness is determined by their "restoration time". The smaller this time is, the faster the system will enable the hardware to undertake again the processing of the input and thus the smaller the FIFO that will be needed. Clearly, the 1-PPR design doesn't allow any HW core to process the input until the reconfiguration completes. Contrarily, the 2-PRRs design releases the uncorrupted HW core once diagnosis is made without waiting for the reconfiguration to complete.

Our approach resembles TMR in that a third model of the application exists; however it is developed in software. The software model offers the following capabilities; i) it is developed in a hard processor which is less possible to get upset and affected by permanent SEUs, ii) it doesn't operate continuously on the incoming data like a third replica in the TMR does, and iii) its resource utilization doesn't scale with the same rate hardware resources would do for large application designs. In particular, the demands of the software model for a larger application design would increase only with respect to the program memory. This is of low concern as compared to the hardware counterpart that besides memory requires more logic and routing resources. Especially for a large design, triplication will result in a dense circuit and in clock degradation [5]. We anticipate that the benefits of the proposed scheme over TMR will be exhibited for large HW cores. This will be possible if a generic software subsystem with relatively fixed amount of resources that won't increase with the application size and with a small FIFO can be built.

It is worth noting that until the system is restored the processing rate decreases. This will disturb the output, either by halting it completely, or by lowering the rate the processed data occur at the output. Also, reliability issues outside the guarded core haven't been taken into account. The same practice is followed in other projects as well [1].

## III. PROOF-OF-CONCEPT

The proposed scheme was implemented in a Virtex-II Pro FPGA that embeds two hardcore PowerPCs (PPCs). The guarded application was a $7^{th}$-order FIR filter operating on 12-bit data. In order to evaluate our scheme for an as much as possible susceptible core, the hardware filter was built with configurable elements only, i.e. LUTs. This practice has been followed by other researchers as well working towards vendor-independent filters that are effectively configured in partially reconfigurable devices [7].

Figure 1 shows the architecture of our scheme in the Virtex-II Pro FPGA. The grey components correspond to the hard resources, while the white components are SRAM-based. Once the PPC0 (processor A of our scheme) detects a fault it informs the PPC1 (processor B of our scheme) to process the data with the software FIR. The rest operation depends on which design amongst the 1-PRR and 2-PRRs is deployed. Each PPC has a program memory implemented with BRAMs along with a memory controller, and a dedicated bus interface. The PPCs communicate through a shared memory implemented with BRAMs. The partial bitstreams are loaded by the PPC0 from an external compact flash memory. The FIFO is implemented with embedded BRAM blocks and its controller is implemented with the TMR technique. The FIFO is connected with the FIR cores and the PPC1. The guarded components implemented in partially reconfigurable regions, i.e. the FIR cores and the TMR FIFO controller, are enclosed in thick-lined boxes.

We implemented both the 1-PRR and 2-PRRs. Their operation accords with the operation of the general scheme described in Section II. Table II consolidates the operations that affect the "restoration time" for each design. In the 1-PRR it is influenced by the reconfiguration time, while in the 2-PRRs it mainly depends on the time to identify the corrupted filter.

### A. Resource Utilization

Table III has the resource requirements of our scheme derived from Figure 1. The first column has the component type and the second column reports whether it is SRAM-based (S), or hard IP (H). Amongst the SRAM-based components only the HW cores and the FIFO controllers are guarded, while the others belong to the static part of the system. The fourth column of the Table III has the components needed for the alternative TMR solution.

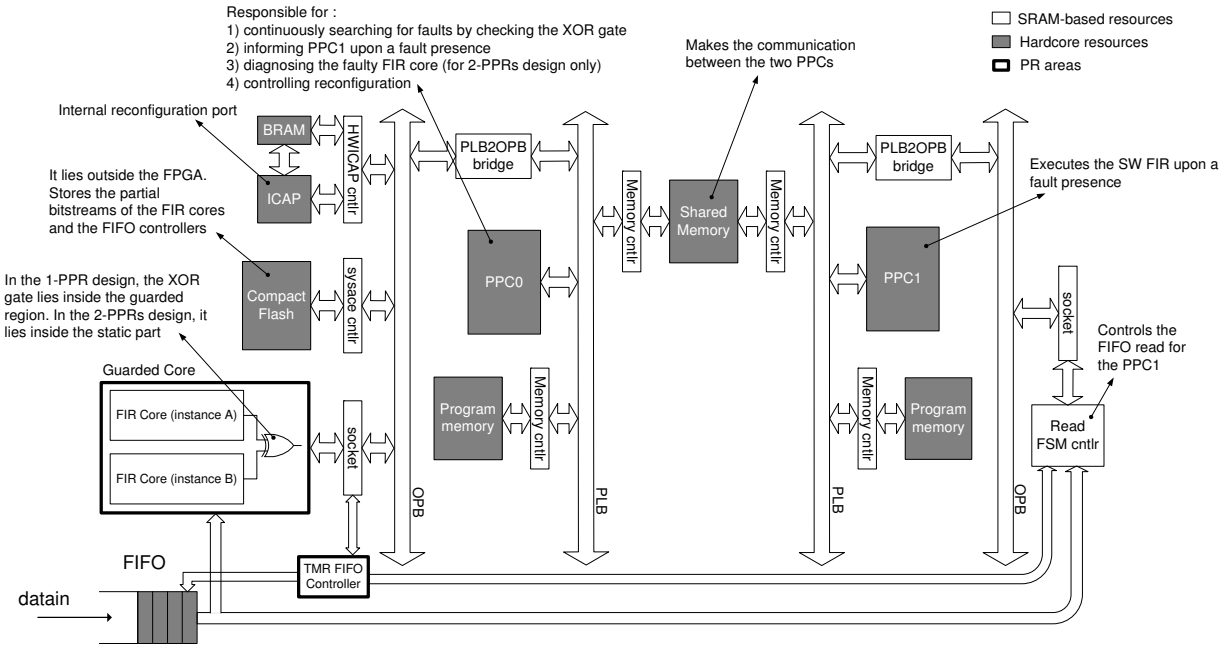


Figure 1. Block diagram of the scheme implemented in a Virtex-II Pro FPGA.

Table IV
RESOURCE UTILIZATION FOR THE STATIC AND THE PARTIALLY RECONFIGURABLE PARTS OF THE 2-PRRS DESIGN IN A VIRTEX-II PRO.

| Resource type | Available | Static (%) | FIR cores (%) | TMR FIFO Cntlr (%) | Total (%) |
|---|---|---|---|---|---|
| PPCs | 2 | 2 (100%) | - | - | 2 (100%) |
| LUTs | 27392 | 3654 (13%) | 4288 (15%) | 480 (2%) | 8422 (30%) |
| Flip-Flops | 27392 | 3488 (12%) | 4288 (15%) | 480 (2%) | 8256 (29%) |
| Slices | 13696 | 3181 (23%) | 2144 (15%) | 240 (2%) | 5565 (40%) |
| BRAMs | 136 | 50 (36%) | - | - | 50 (36%) |

Table II
OPERATIONS AFFECTING THE RESTORATION TIME. "YES" AND "NO" MEAN THAT THE CORRESPONDING OPERATION TAKES PLACE IN THE DESIGN, BUT IT AFFECTS AND IT DOESN'T AFFECT THE RESTORATION TIME RESPECTIVELY. "N/A" STANDS FOR "NOT APPLICABLE" MEANING THAT THE CORRESPONDING OPERATION DOESN'T TAKE PLACE IN THE RESPECTIVE DESIGN.

| Operation | 1-PRR | 2-PRRs |
|---|---|---|
| Reconfiguration | yes | no |
| Processors' communication | yes | yes |
| FIR initialization | yes | no |
| Execution of SW model (for diagnosis) | n/a | yes |
| Comparison between SW and HW results | n/a | yes |

Table IV has the resource utilization of the 2-PRRs design. It occupies relatively more logic resources than the 1-PPR, thus we will compare the worst case of our scheme in terms of area with the TMR. The BRAMs are used mainly for the software subsystem, i.e. the PPC's program memories and the shared memory. Only one BRAM is used for the configuration cache of the ICAP and one BRAM for a $1024 \times 12$ FIFO. The amount of BRAMs that is allocated for the software subsystem is the minimum that can be set as imposed by the vendor's tools. However, it is much beyond the present memory needs and probably no additional BRAMs would be needed to develop in software a larger application.

### *B. Benefits, Drawbacks and Extensions*

The proposed approach detects and mitigates upsets affecting the HW core at real time. For a larger application the static area will not necessarily increase. But even in this case it is likely that it won't scale as much as a hardware core would do. The FIFO size relates to the restoration time and the input data rate. The faster the restoration time and the smaller the input rate is, the smaller the FIFO that is needed. These factors are dependent on the application.

In present work except for the application core and the FIFO controller we do not guard other components such as the memories, controllers, buses and interfaces of Table III. To increase the safety at system level such concerns should be addressed. For instance, in order to guarantee the operation of memories Error Correction Codes (ECC) can be employed. Alternatively, the SRAM resources of the static part can be placed in a partially reconfigurable region and checked periodically so as to "scrub" it when the output of the software subsystem disagrees with both HW cores' outputs.

Table III
RESOURCE REQUIREMENTS FOR OUR SCHEME AND THE TMR SOLUTION.

| Component | S/H | our scheme | TMR |
|---|---|---|---|
| HW cores | S | 2 | 3 |
| majority voters | S | - | √ |
| minority voters | S | - | √ |
| feedback logic | S | - | √ |
| bus macros | S | √ | √ |
| ICAP port | H | √ | √ |
| HWICAP controller | S | √ | √ |
| ICAP's BRAM | H | √ | √ |
| sysace controller | S | √ | √ |
| I/O interfaces | S | √ | √ |
| hardcore processor | H | 2 | 1 |
| program memory | H | 2 | 1 |
| shared memory | H | 1 | - |
| memory controllers | S | 4 | 1 |
| internal bus (OPB and PLB) | S | √ | √ |
| FIFO | H | √ | - |
| TMR FIFO controller | S | √ | - |
| voter for the TMR FIFO controller | S | √ | - |
| ReadFSM controller | S | √ | - |
| XOR gate | S | √ | - |

Table V
SLICE UTILIZATION OF OUR SCHEME VS. TMR FOR VARIOUS BASE DESIGNS IN THE VIRTEX-II PRO.

| base | our scheme $base \times 2.3 + Static + TMR\ FIFO\ cntlr$ | TMR 5x $base \times 5$ |
|---|---|---|
| 901 | 2144+3181+240=5565 | 4505 |
| 2000 | 4759+3181+240=8180 | 10000 |
| 4000 | 9518+3181+240=12939 | 20000 |
| 6000 | 14277+3181+240=17698 | 30000 |

## IV. COMPARING OUR SCHEME WITH THE TMR

In Table III, we observe that the TMR besides the triplication of the HW core requires a circuit supporting partial reconfiguration in order to "scrub" the corrupted replica. Just like in our system, this circuit comprises the PPC and the ICAP port along with the supporting peripherals. The primary difference between our scheme and TMR is that in the former the third copy of the application runs in software. This allows for better scalability in terms of area.

In Table IV it is shown that the TMR FIFO controller consumes much less resources as compared to the HW core. Even if the HW core increases, the area occupied by the three FIFO controllers would remain about the same. In the TMR solution the third HW core would scale with the application size. Apart from it self the third core will incur an additional overhead due to the bus macros and the I/O interfaces, which is not paid in our scheme. In the latter the needs in program memory would scale less, thus the problem translates mostly to whether the FIFO size can be kept small.

Table V projects the amount or slices needed for a larger design than the filter we used, according to the scalability factors derived from Table IV for our scheme, and from the literature for the TMR [1]. The base (static) design of the filter occupies 901 slices of the Virtex-II Pro. For the largest base design our scheme achieves an area reduction of 41% over the TMR.

Table VI
RESTORATION TIME AND MAXIMUM (TESTED) INPUT RATE WITHOUT OVERFLOWING THE $1024 \times 12$ FIFO DURING FAULT HANDLING.

| | 1-PRR | 2-PRRs |
|---|---|---|
| Restoration time | $907.67\ ms$ | $23.33 \times 10^{-3}\ ms$ |
| Maximum input rate | 4.4 Mbps | 480 Mbps |

## V. EXPERIMENTAL RESULTS

To test our scheme we created an artificial way of injecting faults. A streaming environment was emulated with a data generator connected to the FIFO input. Table VI has the restoration time and the maximum input rate that each design can serve without loosing data for a $1024 \times 12$ FIFO. In the 1-PPR design the restoration time is dominated by the reconfiguration time, which presently suffers from considerable delays such as the communication of the PPC with the compact flash and the ICAP port [8].

## REFERENCES


[1] C. Bolchini, A. Miele, and M. D. Santambrogio, "TMR and Partial Dynamic Reconfiguration to mitigate SEU faults in FPGAs," in *Proceedings of the IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT)*, 2007, pp. 87–95.

[2] M. Caffrey, K. Morgan, D. Roussel-Dupre, S. Robinson, A. Nelson, A. Salazar, M. Wirthlin, W. Howes, and D. Richins, "On-Orbit Flight Results from the Reconfigurable Cibola Flight Experiment Satellite (CFESat)," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2009, pp. 3–10.

[3] C. CarMichael, "Triple Module Redundancy Design Techniques for Virtex FPGAs," Xilinx Inc., Application Note XAPP197, Jul. 2006. [Online]. Available: http://www.xilinx.com

[4] Xilinx Inc. (2005) Radiation Effects and Mitigation Overview. Presentation. [Online]. Available: http://www.xilinx.com/esp/aerospace.htm

[5] B. Bridgford, C. CarMichael, and C. W. Tseng, "Single-Event Upset Mitigation Selection Guide," Xilinx Inc., Application Note XAPP987, Mar. 2008. [Online]. Available: http://www.xilinx.com

[6] J. Johnson, W. Howes, M. Wirthlin, D. McMurtrey, M. Caffrey, P. Graham, and K. Morgan, "Using Duplication with Compare for On-line Error Detection in FPGA-based Designs," in *IEEE Aerospace Conference*, 2008, pp. 1–11.

[7] A. H. Gholamipour, H. Eslami, A. Eltawil, and F. Kurdahi, "Size-Reconfiguration Delay Tradeoffs for a Class of DSP Blocks in Multi-mode Communication Systems," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2009, pp. 71–78.

[8] K. Papadimitriou, A. Anyfantis, and A. Dollas, "An Effective Framework to Evaluate Dynamic Partial Reconfiguration in FPGA Systems," *IEEE Transactions on Instrumentation and Measurement*, Apr. 2010.