J Electron Test (2014) 30:425–442
DOI 10.1007/s10836-014-5463-7

Recovery Time and Fault Tolerance Improvement for Circuits mapped on SRAM-based FPGAs

Anees Ullah · Luca Sterpone

Received: 12 February 2014 / Accepted: 24 June 2014 / Published online: 12 July 2014

Abstract The rapid adoption of FPGA-based systems in space and avionics demands dependability rules from the design to the layout phases to protect against radiation effects. Triple Modular Redundancy is a widely used fault tolerance methodology to protect circuits against radiation-induced Single Event Upsets implemented on SRAM-based FPGAs. The accumulation of SEUs in the configuration memory can cause the TMR replicas to fail, requiring a periodic write-back of the configuration bit-stream. The associated system downtime due to scrubbing and the probability of simultaneous failures of two TMR domains are increasing with growing device densities. We propose a methodology to reduce the recovery time of TMR circuits with increased resilience to Cross-Domain Errors. Our methodology consists of an automated tool-flow for fine-grain error detection, error flags convergence and non-overlapping domain placement. The fine-grain error detection logic identifies the faulty domain using gate-level functions while the error flag convergence logic reduces the overwhelming number of flag signals. The non-overlapping placement enables selective domain reconfiguration and greatly reduces the number of Cross-Domain Errors. Our results demonstrate an evident reduction of the recovery time due to fast error detection time and selective partial reconfiguration of faulty domains. Moreover, the methodology drastically reduces Cross-Domain Errors in Look-Up Tables and routing resources. The improvements in recovery time and fault tolerance are achieved at an area overhead of a single LUT per majority voter in TMR circuits.

Responsible Editor: C.-W. Wu

A. Ullah (✉) · L. Sterpone
Department of Control and Computer Engineering, Politecnico di Torino, Corso Duca degli Abruzzi, Torino 24-10129, Italy
e-mail: anees.ullah@polito.it

L. Sterpone
e-mail: luca.sterpone@polito.it

Keywords Triple Modular Redundancy (TMR) · Partial and Dynamic Reconfiguration · Single Event Upsets (SEUs) · Cross-Domain Errors (CDEs)

1 Introduction

The ubiquity of electronic systems in today's world owes their existence to the aggressive technology scaling of transistor channel lengths. This miniaturization towards the atomic and molecular dimensions has produced computationally powerful and high performance systems. Unfortunately, the quantum physical world that governs the nano-scale geometries has its own challenges that need to be addressed for reliable implementation of computations. In particular, radiation-induced Single Event Effects (SEEs), although known for a long time, have become increasingly relevant for reliability of nano-electronic systems. These events have the potential to result in system failures by producing transient and permanent faults [9]. While the effects of radiations are more pronounced as the altitude from earth increases, nevertheless, the increase in sensitivity to radiation with the shrinking feature sizes is becoming a major concern for applications operating on ground as well [11] [14]. Radiation-induced faults in space, avionics, automotive and bio-medical applications can result in huge loss of life and money. The design and implementation of safety-critical systems requires mitigation strategies from device to system level to increase reliability and dependability.

State-of-the-art SRAM-based FPGAs with low Non-Recurring Engineering (NRE) cost, rapid time to market, high device densities and heterogeneous computing resources (BRAMs, MACs and Soft/Hard processors) are increasingly used in the design of safety-critical systems. Unfortunately, SRAM cells, which hold the configuration bits of the FPGA design, are notoriously sensitive to radiation-induced Single

Event Upsets (SEUs). In order to avoid single point of failures in circuits implemented on SRAM-based FPGAs, Triple Modular Redundancy (TMR) is the main methodology selected by designers [4]. The logical fault masking ability of TMR enables the systems to operate fault-free if an SEU creates a bit flip in one of the TMR domains. However, the persistence of SEU effects in the configuration memory can result in failures of two TMR domains and thus the fault masking can break. This requires periodic (e.g., blind scrubbing) [2] or selective (e.g., single frame read-back and correction [7]) reconfiguration of the bit-stream to stop the accumulation of SEUs in the configuration memory.

While the growing device densities enable the implementation of more functionality per unit area, it has two important consequences for TMR circuits implemented on SRAM-based FPGAs. First, the close proximity of SRAM-cells increases the probability of a single particle to effect two neighboring cells. These single event multiple bit upsets can become critical for TMR circuits when the two neighboring cells are storing the configuration bits of two different TMR domains. These multiple bit upset increases Cross-Domain Errors (CDEs) that result in TMR failures [17]. Second, the growing size of configuration memories and the unimproved speeds of reconfiguration interfaces are causing the reconfiguration times to increase with each new generation of FPGAs. For example, the 28 nm Virtex-7 FPGA has a configuration time of 125 milliseconds [12]. This system downtime can be unaffordable for safety-critical applications with strict real-time requirements. Nowadays, FPGA implementation tools usually tend to optimize the overall circuit performances, however, such kind of optimizations have an opposite effect on dependability of circuits. Since, logic and routing resources tend to have a high congestion, particularly in placing voting resources in the same Configurable Logic Block (CLB), the resultant placement increases the probability of TMR failures. A conservative placement of each domain to separate physical area on the chip may ensure that logic and routing elements from different TMR domains do not increase the TMR failure probability.

A preliminary work presented in [21] instruments each TMR domain with error detection logic for errant domain identification and leverages the non-overlapping domain placement for selective reconfiguration. The present work improves the state-of-the-art techniques with several novelties. The former, consists in an error detection logic introduced within the TMR voting structure. In comparison with the traditional minority-voter based implementation, the proposed method uses less routing segments. This reduces the probability of CDEs in the detection circuitry. Secondly, a new strategy for the individuation of the error location within the FPGA architecture is proposed. The developed scheme adopts carry-chain resources available in each CLB slice improving the SEU detection capability especially considering failures into the FPGA's configuration memory. The latter contribution of the paper is a non-intrusive net-list algorithm analyzer that allows the evaluation of CDEs at the early design phase incorporating static analyzer methodologies [22].

The proposed methodology has been evaluated on different benchmark circuits including microprocessor cores such as the MiniMIPS and Leon-II processors. The experimental results demonstrate a huge reduction in recovery times with respect to previously adopted techniques such as scrubbing. The proposed methodology completely eliminates CDEs in LUTs, while greatly reducing them in the routing interconnects. Moreover, the proposed methodology incurs an area overhead of one LUT per internal majority voter. For all the benchmark circuits that we considered in this work, the error detection logic did not alter the critical path of the circuits. Interestingly, the custom floor-planning of TMR domains enabled the placer to optimize the wire-length more rigorously resulting in performance improvements in some cases. The rest of paper is organized as follows. Section 2 provides background and related work. Section 3 introduces the proposed methodology by highlighting the architectural and layout level changes. Section 4 supports the methodology with experimental results while section 5 concludes the paper.

2 Background And Related Work

State-of-the-art SRAM-based FPGAs are heterogeneous devices containing macro blocks like DSPs, BRAMs, Clock Management Tiles (CMT) and Input and Output Blocks (IOBs) along with the Configurable Logic Blocks (CLBs) inside the FPGA fabric as shown in Fig. 1. The resources within these devices are arranged in columns that span from top to the bottom of the device in each row. The dimensions of columns and rows are related to the device family type [24] [10]. Moreover, the FPGA's configuration memory is divided into Reconfigurable Frames (RF) also called major columns. Depending on the type of resource and the specific FPGA family, the RF may contain a number of frames, or minor frame [23]. The magnified view in Fig. 1. shows the frames in a CLB tile of an FPGA belonging to the Virtex-5 family. This layout of the SRAM-based configuration memory is designed to support run-time dynamic partial reconfiguration. However, because of the sensitivity of SRAM-cell to radiation induced SEUs, the designs need fault tolerance methodologies like Triple Modular Redundancy (TMR).

In contrast to the Application Specific Integrated circuits (ASICs), where memory elements are particularly protected against SEUs, SRAM-based FPGAs must implement full

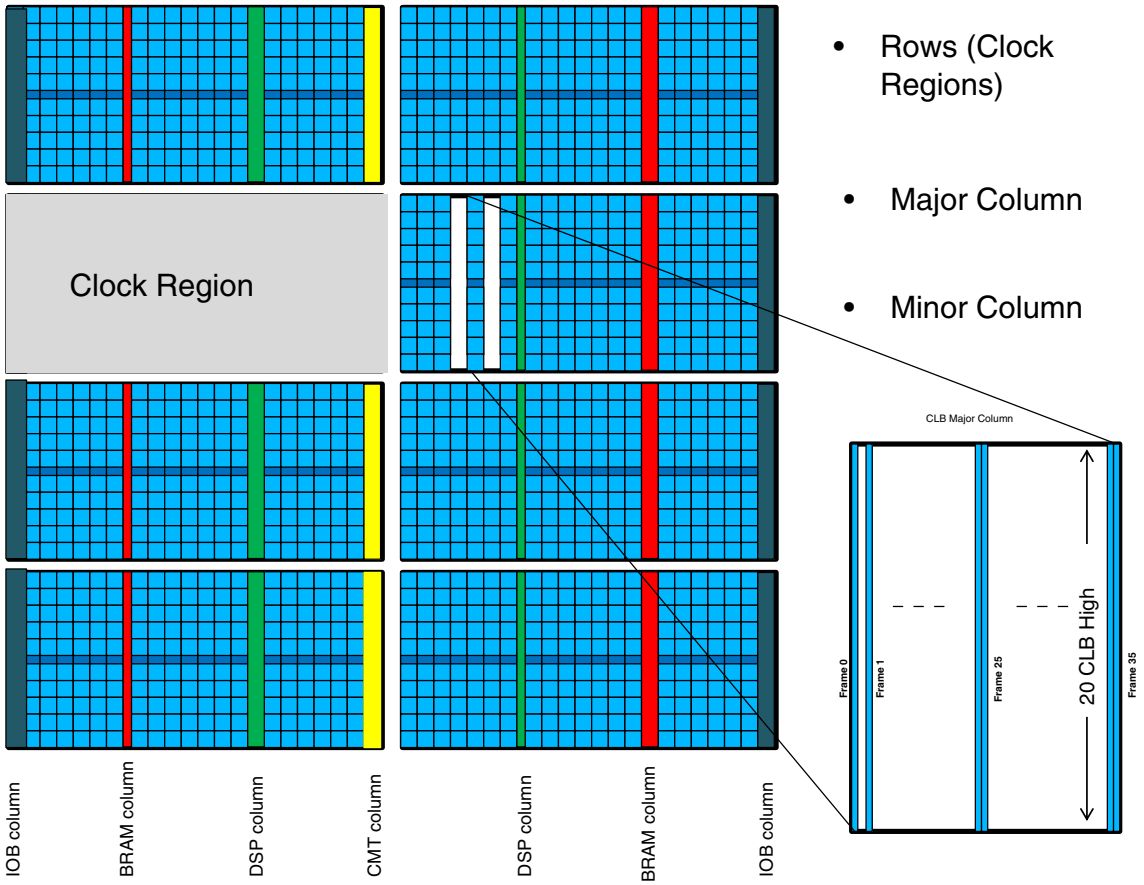

Fig. 1 Resource and configuration memory layout of Virtex-5 SRAM-based FPGA

triplication of all the design elements. Figure 2 shows the typical scenario of circuits implemented with Xilinx TMR tool [4]. It is possible to notice that the architecture consists of partitions and domains, where domains are the replicas of the original circuit, while partitions are the division of the logic resources by the three voter elements.

According to the data structure to be hardened, commercial redundancy tools (e.g., Xilinx TMR tool), handles throughput logic, state-machine logic, I/O logic and special features differently. For example, partition 2 as shown in Fig. 2, represents state-machine logic where the majority voter is inserted on the output of a state register that loops back to the combinational logic in the same domain. This arrangement ensures that an SEU will be automatically corrected by the inherent synchronization due to three voting structures in feedback path with the registers before the next upset occurs. The output logic in "partition 1" in Fig. 2 shows a minority voting based structure to combine the output from the three redundant copies into a single output. In case of a fault in any domain, the corresponding minority voters will tri-state the corresponding output. This helps to avoid signal contention on the output package pins of the FPGA. The TMR architecture shown is Fig. 2 has only one signal group where a signal group refers to the triplet of majority voters that receive the same inputs signals. However, in real circuits each domain consists of multiple signal groups that cross the boundary form one partition to the next partition.

TMR circuits can mask faults when one of the three majority voters of a signal group receives faulty values. However, if more than one input to the majority voters of a signal group is faulty, the fault can propagate to the outputs. Therefore, it is necessary to correct the faulty configuration memory by reloading the configuration bit-stream often called the scrubbing process. TMR with blind scrubbing technique [2] writes back the whole bit-stream after a certain amount of time known as the scrub cycle. The scrub cycle is selected according to the SEU rate and is usually set to ten times the upset rate for the application and device technology at hand [2]. This technique has the advantage of avoiding the accumulation of Multiple Bit Upsets (MBUs) that may cause multiple bit flips in the configuration memory. However, the frequent scrubbing and the comparatively long duration of scrub cycle,

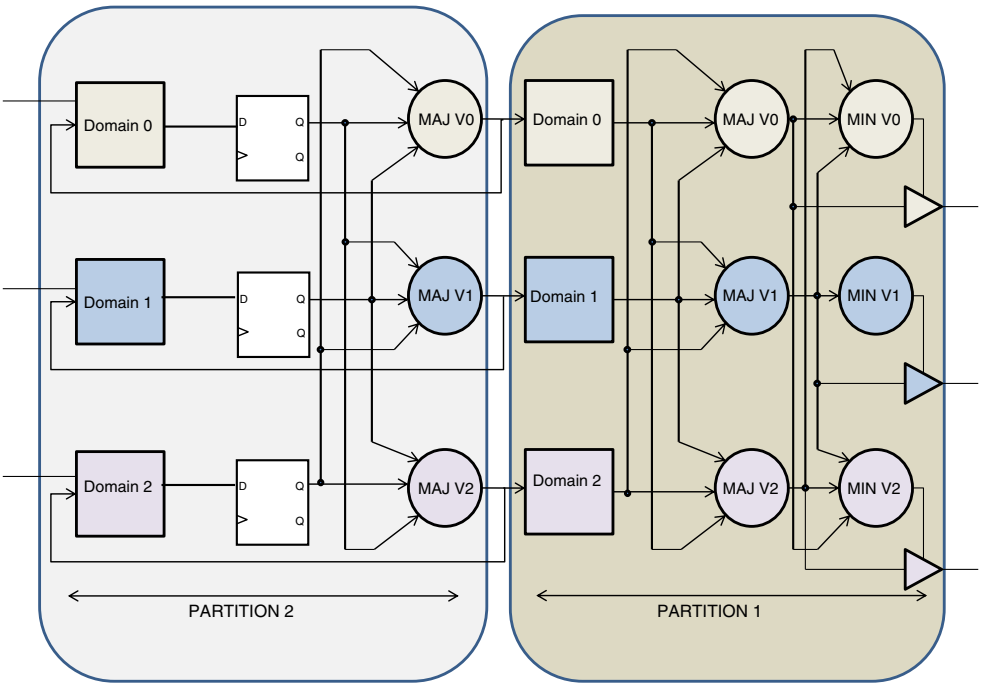

Fig. 2 Xilinx TMR Architecture

exposes this method to increased probability of an SEU corrupting the configuration data during scrubbing. This can lead to catastrophic effects including the corruption of functionality and on-chip contention, which in turn causes high currents large enough to damage device subs-tram.

In comparison with the blind scrubbing, the read-back scrubbing continuously reads back the FPGA configuration bit-stream while the FPGA is in operation, it compares the bit-stream with a stored golden copy and writes it back in case of a mismatch. The most frequently used technique for read-back with scrubbing is the Xilinx SEU macro [7]. This technique operates on the smallest unit of reconfiguration which is a frame. Each frame is protected by error detection and correction codes and during the operational phase it is read out and re-calculated on the frame contents to verify its integrity. Combining the Xilinx TMR with this method can achieve high reliability against SEUs. The advantage of this technique is that the reconfiguration time is very low and the probability of SEU corrupting the configuration data is nullified. However, the configuration engine of the FPGA is unavailable for run-time dynamic partial reconfiguration which is required for adaptive hardware systems. Moreover, the continuous error detection and correction procedure requires energy. A scheduling based technique is proposed by the authors in [19] to deal with the aforementioned issue which introduces its own challenges in terms of implementation and synchronization and can end-up using more resources in hardware for achieving this task. Moreover, read-back scrubbing cannot detect MBUs in multiple frames which can cause functional faults of the system.

Detecting SEU effects by configuration read-back and/or comparison/error detection and correction codes removes every bit-flip from the FPGA's configuration memory. However, every bit is not significant and could not result in circuit level functional fault [5]. Therefore, circuit-level error detection, localization and correction can significantly reduce the overhead related to continuous/regular scrubbing. The authors in [1] [16] proposed a methodology to selectively reconfigure a faulty domain, by combining large-grain TMR with partial reconfiguration. An error-detection logic embedded in the voters enabled the identification of faulty domain. The effects of CDEs were minimized by removing the voters in the internal TMR partition [1]. This leads to synchronization issues after the faulty TMR partition is reconfigured. This is solved by predication based approach which can only be applied to small finite state machines. Another interesting but preliminary work in [6] proposed a dynamic domain-level reconfigurable TMR architecture which has the abilities of error detection, localization, repairing and re-synchronization. The authors adopt a high-level methodology for triplication and insertion of error detection logic at a coarse-grain level. In contrast, this work is based upon post-synthesis modifications in the X-TMR architecture and uses a fine-grain error detection and non-overlapping domain placement for partial reconfiguration.

3 The Proposed Methodology

The goal of this section is to present a new TMR fault-tolerance methodology that reduces the recovery time and the number of CDEs affecting circuits implemented on SRAM-based FPGAs. The methodology mainly consists of architectural and layout level changes introduced in the Xilinx TMR architecture [4]. The architectural-level changes include the insertion of error detection and error flag convergence logic while the layout-level changes consist of non-overlapping domain placement. In order to accomplish both these tasks, two alternative algorithms were developed. The first algorithm is based on the modifications in the physical-level circuit description while the second algorithm modifies the post-synthesis triplicated net-list as shown in Fig. 3. The physical-level algorithm inserts the error detection and flag convergence logic in the mapped design. This post-mapping modification makes it impossible (for certain FPGA families like Virtex-5) to use the commercial mapper on the modified design imposing the use of custom placement algorithm which can result in performance degradation. The second algorithm eliminates this problem by inserting the error detection and flag convergence logic in the synthesized and triplicated net-list. As the modifications are done before mapping the design to the FPGA, this algorithm integrates well with the commercial tool-flow and results in better performance. The following sub-sections describe all the phases of the proposed methodology in detail.

3.1 Proposed TMR Architecture

The architectural changes consist of introducing error detection and flag convergence logic at the granularity of TMR domains. The error detection logic uses gate-level functions for comparison, resulting, in a large number of error flags for each domain. As it is fruitless to identify an errant domain with more than one error flags, the flag convergence logic connects together all the flags from a single domain, generating a unique flag per TMR domain. A novel flag convergence logic is introduced in this section to make the realization of fine-grain error detection feasible for TMR circuits. The rest of details are in the following sub-sections.

3.1.1 Error Detection Logic

A fine-grain error detection logic based on logic gates is used to identify faults in a TMR domain. The implementation and insertion of error detection logic in TMR circuits can be accomplished in three different ways mainly based on minority voter and equivalence function shown in Fig. 4. The minority voter has three input signals consisting of a primary and two secondary inputs. The minority voter circuit generates a logic "0" value on the output when the primary agrees with at least one of the secondary inputs otherwise the result is a logic "1" value [4]. Thus, the output of minority voter can be used as an error flag signal. In Fig. 4a, the minority voter is fed by the outputs of majority voters. In case of a fault in one of

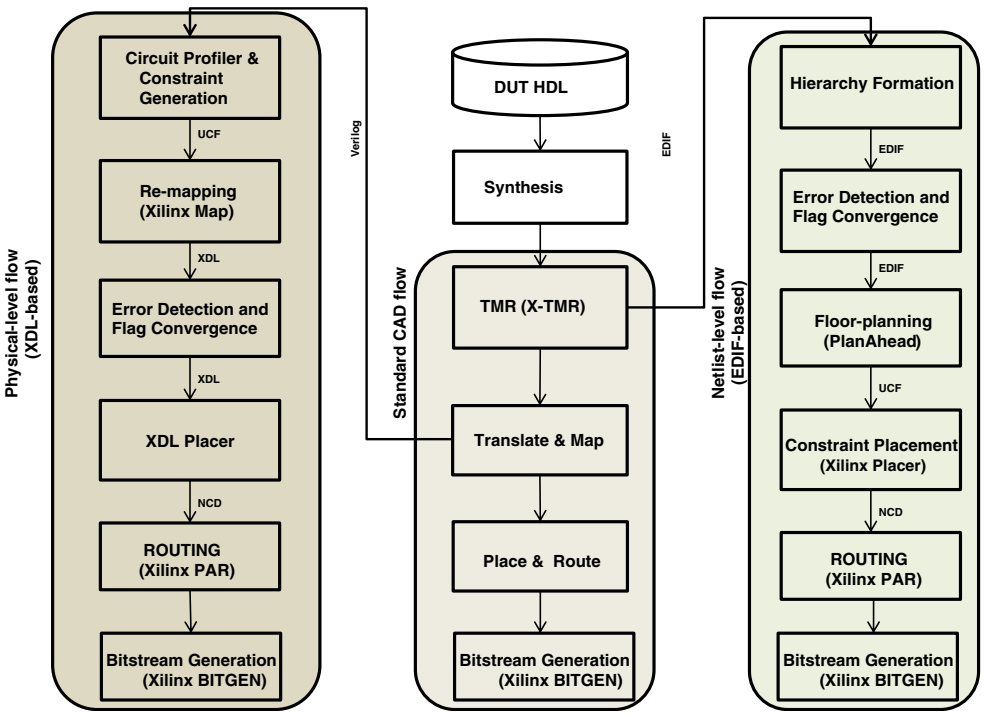

Fig. 3 The proposed flows: physical-level and net-list level flows

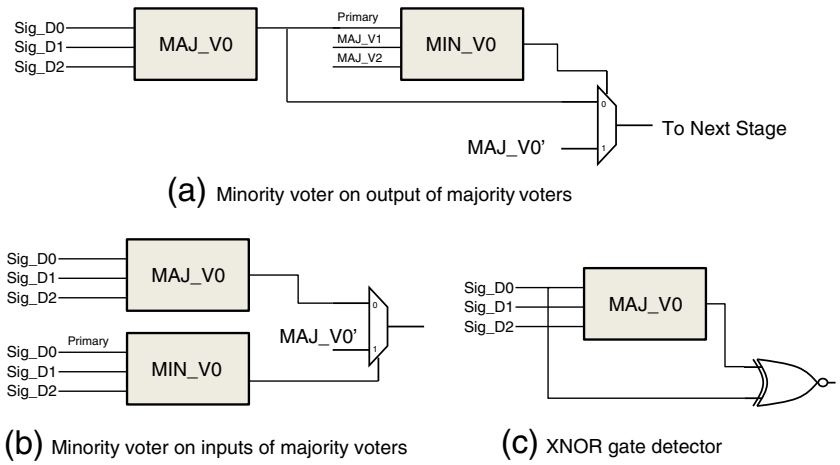

Fig. 4 Gate-level error detection circuits for TMR architecture

the majority voters, the corresponding minority voter will flag an error condition and will disable the propagation of error to the down-stream logic by controlling the select line of the multiplexer. The multiplexer receives inputs from the primary domain and one of the secondary domains. However, a fault in any resource before the majority voter will be masked from reaching the minority voter and it cannot be detected. In Fig. 4b, the minority voter is placed on the signals that feed the majority voters. In this way, the majority voter will mask faults while the minority voter will detect faults. This type of detector has the ability to detect faults in the resources before the majority voters but a fault in majority voter itself cannot be detected using this method. As the minority voter based detectors require input signals from three different domains of TMR, it is subject to the same reliability consideration during the placement as the majority voter. Furthermore, their usage can lead to routing congestion and error un-detection if they are placed in the same CLB with the majority voters.

Considering the propagation of a fault from a domain in a given partition to the next, the majority voters will mask the fault. Similarly, if the next partition is the output stage of the TMR circuit, the minority voters will disable the output to avoid signal contention on the output package pins of the FPGA. Therefore, the steering logic, implemented as multiplexer in minority voter based detectors, can be removed without affecting the functionality of the circuit while the repairing procedure is carried out.

To overcome the limitations of minority voting based error detectors, this work adopts an XNOR based error detector as illustrated in Fig. 4c. As can be noted from the figure, the XNOR gate is fed by the majority voter and the primary signal that feeds that majority voter. The output of XNOR gate produces a logic "1" value when both the inputs are the same. A fault affecting the logic module before the majority voter or a fault in the majority voter itself will result in a logic "0" value. Therefore, this circuit combines the error detection capabilities of the individual minority voter based detectors. Moreover, as compared to the minority voter based detectors, XNOR detector will be more resilient to routing errors because it receives inputs from the same domain logic. As you can notice from the scheme illustrated in Fig. 4, compared to the minority voter based detectors it uses less number of inputs, thus in a highly congested design this saving can be significant in terms of the routability and congestion avoidance.

3.1.2 Flag Convergence Logic

The use of fine-grain error detection logic generates a large number of error flags for each TMR domain. As this work tends to detect errors at the granularity of TMR domains, more than one error flag emerging from each TMR domain is not useful. The purpose of flag convergence logic is to connect them together to have a unique flag per TMR domain. This is achieved by using the built in-slice carry-chains available of fast arithmetic computations in a novel way. As a large number of carry-chains available in each CLB slice on modern FPGAs remains unused because the application at-hand does not require it or because the CAD tools could not infer them, they can be exploited for certain applications [13] [18] . The insertion of error detection and flag convergence logic in a TMR domain is illustrated with a placement-level view in Fig. 5 where SLICE 1 contains the majority voters while SLICE 2 contains the error detection and flag convergence logic. The placement of same domain voters is a mandatory constrain for our methodology. It should be noted that modern FPGAs provide local power supply and ground located near each slice column in

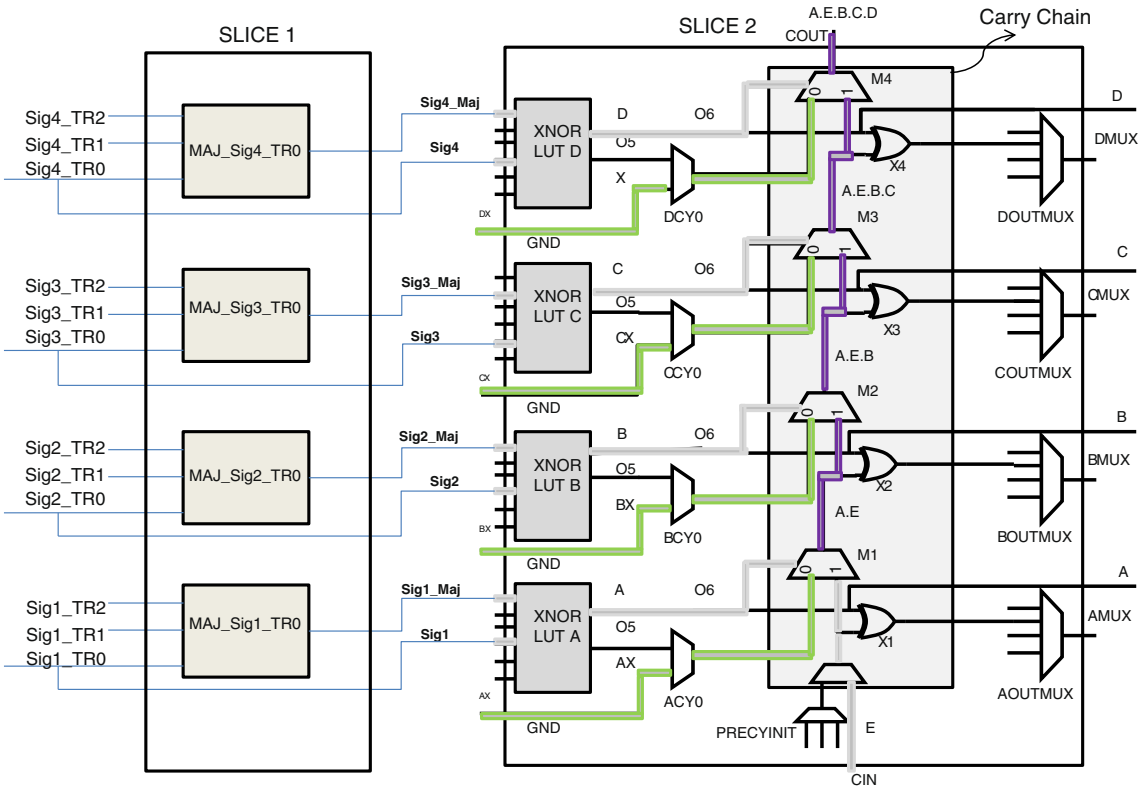

Fig. 5 A device-level view of the insertion of error detection and flag convergence logic in domain

the form of the TIEOFF elements, making it possible to connect the inputs of the hardwired multiplexer in the carry-chain to the fixed logic values like VCC and GND. These elements can be configured to VCC or GND by modifying the physical circuit description in the post-map net-list. Although Fig. 5 shows the flag reduction for four XNORs, this method can be extended to connect multiple slices using the dedicated carry-wires [24].

The usage of carry-wires to connect the hardwired carry-chain resources makes the detectors resilient to errors in the detector itself. Although, the proposed solution efficiently detects error in a TMR domain with a unique flag, however, due to the propagation delay along the carry-chain, the method has associated error detection latency. The error detection latency depends upon the number of majority voters at the boundary of a TMR domain in each partition. The formation of partition depends upon the nature of computations that the circuit implements causing a considerable variance in the size. Consequently, the size of domain within the same partition is constant but it can vary when we move to other partitions. Therefore, the detector length and error detection latency varies in the same TMR circuits for domains of different partitions. This work does not optimize the trade-off between the detector length and the number of error flags; instead the goal is to reduce the number of error flags to the minimum possible for every domain.

3.2 Non-Overlapping Domain Placement

The non-overlapping domain placement is the layout of TMR domains to separate and non-overlapping areas on chip. As the current CAD tools from the vendors are optimized for timing, the stringent requirements on the timing closure may lead to place elements from different domains in the same slice. This results in a critical voter placement scenario as shown in Fig. 6a. It can be noted that the voters belonging to the same signal group, Voter 1, but different domains are placed in the same CLB slice. Each domain is represented with a different color. The corresponding critical voter bitmap is shown in Fig. 6b, where “1”, “2” and “3” represents the configuration bit for the corresponding LUTs. The configuration bit “0” means that the bit is un-used. The criticality of some of the neighboring bits, responsible for the configuration of two different TMR domains is shown with overlapped squares in Fig. 6b. The close

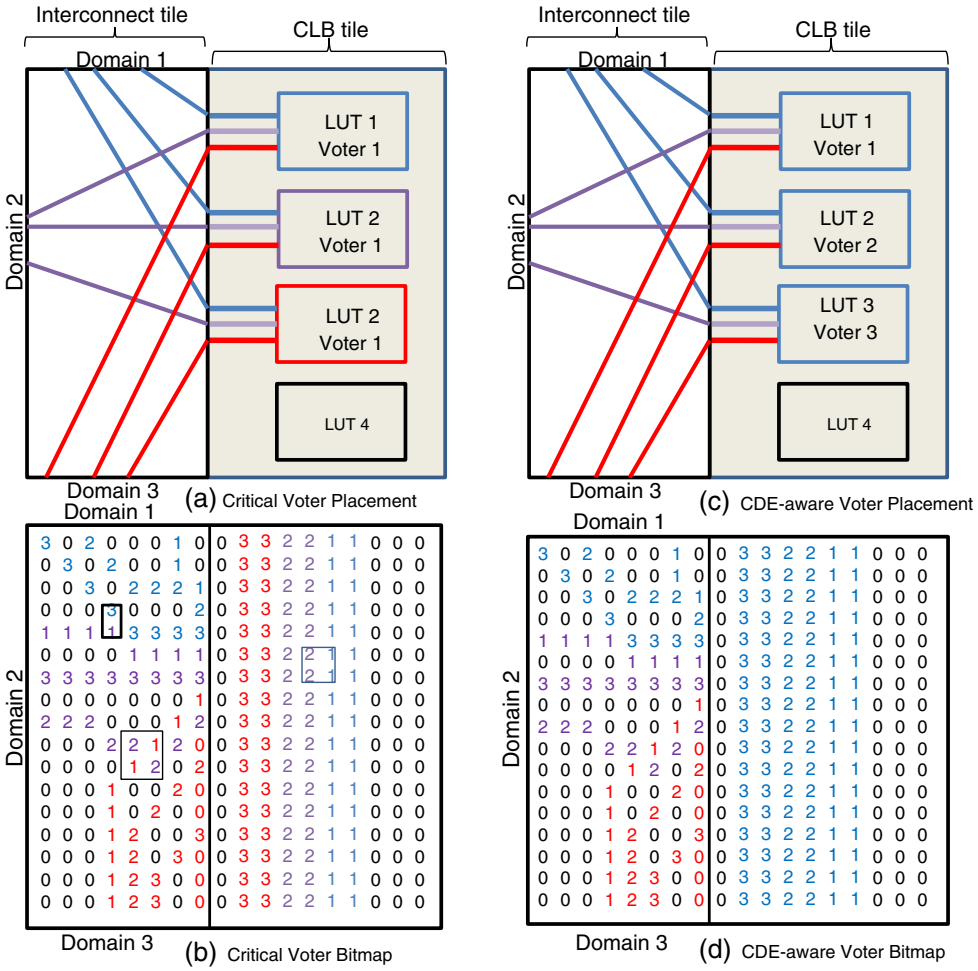

Fig. 6 Effects of placement on cross-domain errors

proximity of bit-stream frames in this scenario results in a large number of CDEs in logic as well as in routing. In particular, the interconnect tile contains nine nets from three different domains that feeds combination logic from three different domains. One example of a CDE-aware voter placement is shown in Fig. 6c where the voters from different signal groups, denoted by Voter 1, Voter 2 and Voter 3, but belonging to the same domain, shown with same color, are packed together in a CLB slice. The corresponding CDE-aware bitmap in Fig. 6d, which follows the same convention, demonstrates a more reliable situation. It can be noted that the number of CDEs in the configuration frames of CLB tile in this case are zero. The constraints on placement not only guarantee that voters are placed in CDE-aware manner but it also places all other logic elements separately from other domain elements. Consequently, most of the routing interconnection reside inside the boundaries of a placed domain and the routing CDEs considerably decreases as well.

As each of the TMR domains is already instrumented with logic that identifies the errant domain, the non-overlapping placement can be used to correct the faulty domain without disturbing the real-time operation of the circuit. This is achievable by exploiting the Dynamic Partial Reconfiguration (DPR) capabilities of modern SRAM-based FPGAs. Therefore, the non-overlapping domains placement accounts for the dramatic decrease in the reconfiguration times of TMR circuits using partial reconfiguration of a faulty domain. The detailed explanation of the implementation of placement, for commercial FPGAs, is presented in the next section.

3.3 Physical-Level CAD Flow

The physical-level flow is based upon the modification of the mapped TMR circuits to the FPGA architecture. The mapping process packs the design LUTs and FFs into slices and assigns them physical locations on the FPGA fabric. The physical flow starts from the mapped design

and applies the proposed architectural and layout-level changes presented in Fig. 3. First, a re-mapping of the TMR circuit with packing constraints is performed to avoid logic elements from different domains to be placed in the same CLB slice (section 3.3.1). This is followed by insertion of the error detection and flag convergence logic (section 3.3.2) and finally the custom placement (section 3.3.3) algorithm performs the non-overlapping domain placement. The following sub-sections explain further details of all the phases of the proposed flow.

3.3.1 Circuit Profiling and Re-Packing Constraints Generation

The preliminary phase of the flow consists of creating a Directed Acyclic Graph (DAG) of the TMR circuit by parsing the post-mapped simulation Verilog file (utilizing [3]). The nodes of the graph are modeled as an object with a number of fields such as functional string, instance name, inputs nets, output nets, type of primitive element (LUT or FF) and the corresponding domain and partition. After the net-list is created, each node should be labeled with the corresponding domain, partition and signal group. It should be noted that each node of the graph has a name containing a string referred to as TR0, TR1 and TR2 for the corresponding domain; however, in order to find partitions, majority voters should be identified in the circuit net-list because majority voters are located at the boundary of partitions. Majority voters are identified with the functional strings and three inputs each one from different domains. Furthermore, partitions are created starting from the circuit's primary output pins and traversing the graph in Breadth First Search (BFS) manner towards the circuit primary inputs. Starting from a boundary of majority voters, a unique label ID for the partitions is assigned to all the elements of the graph until the next majority voter boundary is encountered. The next majority voter boundary will be assigned an incremented label ID and the partition label ID will be propagated in a similar manner as before. During the process of identification of partitions, signal groups are also formed by finding majority voter triplets which receives the same nets as inputs. It should be noted that the signal group ID is assigned in a manner that they increment linearly in the same partition but are set to zero and incremented again for the next partition. Once this process finishes, all the nodes of the graph will have the corresponding partition and signal group assigned.

This information is mandatory for generating the circuit profile which is a statistics of the division of elements in domains, partitions and signal groups. The circuit profiler is used to identify the slices which are in packing violations using the criteria that all the LUT/FFs in a slice are related to the same domain. Once the elements violating the dependability rules are identified, a new constraint file is generated using the packing constraints [8] and fed to the mapper to generate a mapped design. This mapped design will drive the subsequent phases of the flow by identifying the insertion locations for error detection and flag convergence logic.

3.3.2 Error Detection and Flag Convergence Logic Insertion

A physical-level tool was developed for inserting error detection and flag convergence logic in the slice-level FPGA design net-list. This tool is based upon the software primitives defined in [20]. The developed "Physical-level Manipulator" presented in Algorithm 1 requires two input files. The former, a voter's list file generated by the circuit profiler which has the partition, domain and signal group information for every majority voter in the design. The latter, a physical-level file that describes the mapped design constrained as presented in section 3.3.1. During the initialization phase, the voter's list file is read and the corresponding domain and partition for each voter is extracted and stored in PartitionDomainMap. Using the names of each voter, the corresponding slice and LUT pointers are extracted and stored in a SliceLUTMap. After identifying the voters in the slice-level net-list, the nets that will feed the XNOR gates need to be identified. For each voter, the nets on the primary inputs and outputs of the majority voters are extracted and stored in a VoterNetsMap data-structure. It is worth mentioning that as the physical-level description is a slice-level net-list while the error detection logic is in form of LUTs, therefore, it needs to be packed together into slices. This is achieved by creating abstract LUTs objects and connecting them to every voter in the VoterNetsMap. In order to pack these created LUT objects, the slice and carry-chain insertion phase combines from the same domain in a partition four XNOR LUTs into a newly created slice instance. A carry-chain object is then inserted into the newly created slice instance and is connected to the previous slice in the same domain and partition pair. This process of interleaving the chains continues until all the LUT objects of a domain in a partition are connected. The insertion of error detection logic and flag signal in the post-map net-list follows a specific naming convention, identifying each flag with its corresponding domain and partition. When all the LUT objects of a certain domain are connected, the output

from the final carry-chain is the flag signal. A new net is created at this point and the output of the carry-chain is added as a source. The created nets are stored in XDLNetsMap for all the partitions and domains. In the controller connection phase, for every flag net in the XDLNetsMap, a target port on the reconfiguration controller is specified. The target port of a flag signal can be an external or internal reconfiguration controller. In case of external controller the destination is an IOB site connecting to a PAD, while in case of internal reconfiguration controller the destination can be a port of a processor or input of a hardware reconfiguration manager. It is important to note that we have added new nets in the mapped design; therefore, it is possible to carry out the routing phase using the commercial place and route tool.

```
Input:
              a. Voter's List file
              b. XDL file
Output:
              Modified XDL file with error detection and reduction logic
// ************* Initialization Phase ************************
1: PartitionDomainMap[Voter] = extract_partition_domain(Voter);
2: SliceLutMap[Voter]        = extract_slicePtr_lutPtr(Voter);
3: VoterNetsMap[Voter]       = extract_inputs_for_XNOR(Voter);
// ************* XNOR LUTs Creation  ************************
4:  foreach Voter "v" in VoterNetsMap do
5:      XNOR_inputs=VoterNetsMap[v];
6:      NewXNORLUTObjectMap[v]=CreateNewXNORLUTObjects(XNOR_inputs);
7: end
// ************ Slices and Carry-Chains Insertion ************
// Notation: p "partition" and d "domain"
8: foreach voters_group with same "p,d" do
// Note that voters_group contains at most four voters
9:      XNORs_cluster = NewXNORLUTObjectMap[voters_group];
10:     NewSliceInstace=CreateNewSliceInstance(XNORs_cluster);
11:     InsertCarryChain(NewSliceInstance);
12:     NewSlicesMap.insert(NewSliceInstance);
13:     Connect2PreviousSliceinNewSliceMap(NewSliceInstance);
14:     if XNORs_cluster contains the last voter in current "p,d" do
15:             flagNet = extract_flagNet_from_slice(NewSliceInstance);
16:             NewNetName = extract_name_with_partition_domain(flagNet);
17:             XDLNet = CreateNewXDLNet(NewNetName);
18:             XDLNet.source(flagNet);
19:             XDLNetMap.insert(XDLNet);
20:     end //if
21: end //forevery
// *********** Controller Connection ************************
22: foreach XDLNet in XDLNetMap do
23:     XDLNet.target(ReturnReconfigurationControllerPort());
24: end
```

Algorithm 1. Physical-level Manipulator

3.3.3 Placement

The independent non-overlapping placement for domains is carried out on the physical-level net-list, which already packed together LUTs and FFs into slices. The physical-level placer which uses a simulated annealing algorithm [20] is modified for independent placement of domains in different physical areas as presented in Algorithm 2. As a pre-processing step, the physical-level description file is analyzed to extract a resource vector representing the computational nodes that will be required for each domain in every partition. An area range on the chip is selected for each resource vector that can house it. More formally, the placement problem formulation can be laid out in the following manner.

Given a TMR partition set represented by $P=\{p_1,p_2,\ldots,p_n\}$ where each partition p_i contains a 3-tuple of domains

$D=\{d_1,d_2,d_3\}$. Let for each domain d_j belonging to partition p_i denoted by $d_{i,j}$ a 3-tuple $rv_{i,j}=\{n_{clb}, n_{bram}, n_{dsp}\}$ represents a resource vector. Moreover, a set of resource vectors $RV=\{rv_{(1,1)}, rv_{(1,2)}, rv_{(1,3)},\ldots.., rv_{(n,1)}, rv_{(n,2)}, rv_{(n,3)}\}$ represents the resource requirement for all the partitions and domains. Similarly, a rectangular region denoted by $rr_{(i,j)}$ is the corresponding area on the FPGA that satisfies the resource vector $rv_{i,j}$ for a domain $d_{i,j}$. Each rectangular region is a 4-tuple $\{x_0, y_0, x_1, y_1\}$ where the bottom left corner is represented by (x_0, y_0) and the top right corner is represented by (x_1, y_1). The floor-plan for the whole design can be represented by $RR=\{rr_{(1,1)}, rr_{(1,2)}, rr_{(1,3)},—, rr_{(n,3)}\}$ which is a vector of rectangular regions on the FPGA fabric. Moreover, $PL=\{P_{(1,1)}, P_{(1,2)}, P_{(1,3)},—, P_{(n,3)}\}$ is the random initial placement of all the domains according to the selected floor-plan. The detailed placement algorithm is shown in the following XDL-level simulated annealing placer.

```
Input:
       Unplaced XDL file
Output:
       Placed XDL file
// ************ Initialization phase ************************************
1: RV{rv(1,1), rv(1,2), rv(1,3), ---,rv(n,3) } = extract_resource_vector(DAGTMR);
2: RR{rr(1,1), rr(1,2), rr(1,3),---, rr(n,3) } = floorplan_area_on_chip(RV);
3: PL{P(1,1), P(1,2), P(1,2),---, P(n,3) } = random_initial_placement(RV,RR);
// ************ Simulated annealing engine ********************************
4: foreach PL(i,j) belonging to PL
5: T = Initial Temperature
6: while ( T > Final Temperature){
7:      PL(i,j)new = swap_two_randomly_selected_elements(PL(i,j))
8:      ΔC  = Cost(PL(i,j)new) - Cost(PL(i,j)) // Half-Perimeter Wirelength Cost
9:      if(ΔC < 0)
10:            PL(i,j)) = PL(i,j)new;
11:     else if(PICK_FROM_UNIFORM_RANDOM_DIST(0,1) < exp(-ΔC/T))
12:           PL(i,j)) = PL(i,j)new;
13:     T = update(T) ;
14: }
```

Algorithm 2. Simulated-annealing based physical-level placer

In the initialization phase, the TMR's DAG is traversed to divide the logic resources into the corresponding domains and partitions. The information extracted from the circuit profiler phase can also be used to locate the LUTs/FFs according to domains and partitions. Based on the required resource a feasible area for each domain is selected. It should be noted the floor-planning step is not optimized in any way. This is followed by an initial random placement of domains according to the selected floor-plan. The simulated annealing algorithm starts by initializing the temperature "T" to a very large value. For each domain, in each iteration, two randomly selected logic elements are swapped to check if the half-perimeter wire-length decrease. Each good move is accepted, however, a bad move can also be selected according to a probability. The probability of accepting a bad move is high in the beginning to enable the algorithm to avoid local minima but decreases with time for convergence towards global minima. The physical-level placement algorithm combined with the un-optimized packing results in a decreased performance for this flow which can be vastly improved by the non-intrusive semi-custom flow presented in the EDIF-based flow in the following section.

3.4 Net-List Level CAD Flow

The net-list described in EDIF grammar is designed with powerful expressive capabilities for all the levels of abstractions related to the electronic design and automation of circuits and systems. It is the industry wide standard for interchanging designs from net-list description to layout phase. Therefore, an alternative net-list based CAD flow is developed to automatically harden circuits according to our proposed methodology to show viability for other vendors. The net-list level CAD flow

(based upon [20]) is shown in Fig. 3 along with the physical-level flow. The flow starts from the triplicated net-list generated by the TMR tool and applies several modifications including hierarchy formation, error detection and flag convergence logic insertion, floor-planning and placing. The rest of the details are in the following sub-sections.

3.4.1 Hierarchy Formation

The input to the hierarchy formation block is a triplicated net-list containing instances from the macro block synthesis library containing cells that are used by TMR tool for hardening the circuits [4]. The functionality of this block is like the circuit profiler block from section 3.3.1. Like the circuit profiler block the purpose is to label each element with a corresponding domain and partition. The net-list uses elements from a library that have special cells for voter structures and the output buffer elements. In contrast to the post-map simulation Verilog file used in section 3.3.1, it is easy to identify them and divide the circuit into partitions. In a similar fashion to the circuit profiler, a BFS algorithm starting from the circuit primary output towards the circuit primary inputs labels each element with a corresponding domains and partitions.

The information extraction from the profiling is used to convert the flat net-list generated by the TMR tool into a domain-level hierarchical net-list. In order to achieve this task, each logic element from every domain and partition is annotated with the corresponding partition and domain. The instances of each cell are renamed by including it in a domain and partition hierarchy. This kind of hierarchy formation is important for the non-overlapping domain placement which is a central theme of this work. The hierarchical net-list is used for constraining the floor-planning and placement using commercial tool-chain and will be in described in section 3.4.3.

3.4.2 Error Detection and Flag Convergence Logic Insertion

The physical-level description file used in section 3.3.2 is a "slice" level circuit description, which already packed LUT/FFs and carry-chains and assigns them physical locations on the FPGA. However, the net-list level flow uses a cell-level net-list utilizing basic elements that comprises slices. Therefore, it is possible to insert the gate-level detectors directly as LUTs and connect them together using instances of carry-chain mux cell. The developed "Net-list Manipulator" presented in Algorithm 3 shows the implementation details for the insertion of error detection and flag convergence logic as illustrated in Fig. 5. In the XNOR gate insertion phase, a new LUT instance is created for the XNOR gate. For each voter, the primary input net and the voter output nets are extracted and are connected to the input of created XNOR LUT object. All the created XNORs are stored in a map data-structure XNORMap indexed by the voter instance. It should be noted that the insertion of XNOR gates follows a naming convention that label them according to the corresponding voter's partition and domain. The output from the last XNOR gate is stored in the FlagNets vector. In the last phase of top-level port creation, an EDIF vector port is created equal to the size of FlagNets. Each of the flag net is connected to a unique bit index of the vector port. This completes the Netlist manipulation algorithm. This methodology is completely compatible with the commercial tool-chain and can be easily adopted for external or internal reconfiguration controller.

3.4.3 Floor-Planning and Placement

The modified hierarchical net-list is used for floor-planning and placement using commercial tools. The floor-planning and placement tools are directed using constraints described in [8] to respect the non-overlapping domain placement required for improving the dependability and recovery time. These constraints are specified for each partition and domain in the design. This helps the floor-planning tool to correctly recognize the hierarchy and extracts the corresponding modules from it. Then, each module is manually floor-planned by selecting a feasible rectangular region on the FPGA fabric. It is also important that during the floor-planning process, the resource requirements for each domain should be satisfied. The use of special resource type like hardwired processors and memories should also be planned accordingly for each domain. For very large designs that may require a large percentage of the device resources, the use of very long detectors can result in infeasible floor-planning, which can be elevated by increasing the number of flag signals per domain. The completed floor-planning information is added as new constraints into the previously created constraint file which is then used by the placer.

The placed design is then routed using the commercial router. The usage of commercial tools for placing the design in the net-list level flow as compared to custom placer in the physical-level flow, results in better performance as presented in the following experimental results section.

```
// Netlist Manipulator
Input:
       EDIF net-list
Output:
       Modified EDIF netlist
// ******************* XNOR gate insertion ************************
1: foreach voter "v" which is not an output voter do {
2:    [primary_net,voter_out_net]=getPrimaryNetandOutputNet(v);
3:    XNORMap[v] = InsertXNORInstance(primary_net,voter_out_net);
4: }
// ******************* flag convergence logic **********************
5: foreach partition "p" belonging to TMR partitions do {
6:   foreach domain "d" belonging to partition "p" do {
7:    foreach XNOR gate "x" in XNORMap do {
8:     if(domain[x]==d && partition[x]==p) {
9:       if(first element in the chain)
          // Implement MUXCY "M1" in Fig 5
10:          Y = InsertMuxCyInstance(GND,output(x),VCC);
11:       else
12:          Y = InsertMuxCyInstance(GND,output(x),prv_O);
13:       } //end if
14:      prv_O = output(Y);
15:      if(last element in the chain)
16:           FlagNets.insert(prv_O);
17:   }// ends iteration over XNORMap
18:  }// ends iteration over domains
19:}// ends iteration over partitions
// ****************** Adding Top-Level EDIF flag Ports ***************
20: FlagPort = CreateTopLevelEDIFPort(FlagNets.size());
21: foreach flag belonging to FlagNets do {
22:    connectflag2portindex(flag,port,index);
23:    increment_index;
24: }
```

Algorithm 3. Netlist Manipulator

4 Experimental Results

To evaluate the proposed approach, the whole system was implemented for Xilinx Virtex-5 LX110T FPGA. A number of test-bench circuits from the processors category of OpenCores [15] repository were used for experiments. All the circuits were hardened using the X-TMR tool and modified according to the methodology presented in Section 3. The following sub-sections emphasize the main results comparing Xilinx TMR with minority-voter [21] and XNOR detectors based TMR methodologies.

4.1 Circuit Profile Statistics

The circuit profile statistics is the division of resource in the TMR circuits according to partitions as shown in Table 1. Partitions are formed by the TMR tools according to the nature of computations the circuit implements. Majority voters in the circuit mark the boundary of TMR partitions. In general, the greater the number of feedback registers in design the more will be the partitions formed. It is important to note that there is a considerable variance among the sizes of TMR partitions. The number of signal groups refers to the triplets of voters in the design and depends upon the number of primary outputs of the circuit.

The number of LUTs and FFs shown here is the sum of all the resources that are spread in the all the TMR partitions. Table 1 also shows the special type of resources in form of DSPs and BRAMs that are required by each circuit. The circuit profile statistics drives the variance in area overhead of the proposed methodology and will be explained in more detail in section 4.4.

Table 1 Circuit profile statistics

Circuit	Partitions	Signal groups [#]	LUTs	FFs	DSPs	BRAMs
RISC5x	3	192	1,805	495	0	0
CORDICR2P	1	40	4,311	3,387	0	0
CORDICP2R	1	32	2,890	2,259	0	0
MINIMIPS	3	528	14,490	5,649	12	0
LEON II	4	748	32,290	8,046	3	39
L80SOC	10	137	1,548	726	0	6
HP16	2	134	3,533	876	0	0
RS_DEC	5	376	3,919	2,130	0	12
FPU	2	136	26,702	14,727	36	0
DCT	3	40	6,576	4,755	0	126

4.2 Recovery Time

The recovery time comparison of the proposed approach with the standard X-TMR is presented in Table 2. The detector length expressed in terms of CLBs is the length of the flag convergence logic implemented with cascaded carry-chains along a column of CLBs. It directly depends upon the number of majority voters at the boundary of a partition. The detector length reported in Table 2 is for the partition with maximum number of majority voters at the boundary. Nevertheless, for other partitions, the detector length and corresponding delay can be less than those maximum values shown in Table 2. The detector delay is the corresponding maximum delay. It is interesting to note that the delay of the longest detector of 192 CLBs, for LEON II, is 27.625 ns. Consider that the device we used in these experiments has 160 CLBs from top to bottom along the CLB column. This means that even detectors which span more than one device resource columns have extremely fast error detection times. This is due to the dedicated communication channel connecting the CLBs along a column, especially designed for fast arithmetic additions. However, such long detector lengths have to be floor-planned by partitioning the chip vertically to accommodate the design constraints imposed by our methodology. The recovery time of XNOR based detector is the sum of error detection latency and the reconfiguration time. The comparison clearly shows that the methodologies based on non-overlapping domain placement hugely reduce the recovery time. The difference between recovery times of minority voter based TMR and XNOR based TMR is not significant. The reasons are the same floor-plan and the fast error detection latency. For Xilinx TMR, the partial reconfiguration of a faulty domain is not possible and therefore in case of a fault the whole FPGA bit-stream should be written back. For a Virtex-5 device that is used in this experimental study the total bit-stream download time is 25 milliseconds. However, the Table 2 represents the values for Xilinx TMR considering the used frames only, in order to have a fair comparison with our approach. Although, the probability of un-detection is quite low due to the low cross-sectional area of the detector and the

Table 2 Recovery time comparison

Circuits	XNOR Detector Based TMR (proposed approach)			Minority-Voter TMR [21] Recovery Time (usec)	Xilinx TMR Recovery Time (usec)
	Detector Length (CLBs)	Detector Delay (ns)	Recovery Time (usec)		
RISC5x	22	11.032	74.061	74.05	666.05
CORDICR2P	0	0	530.25	530.25	1,590.76
CORDICP2R	0	0	355.47	355.47	1,066.41
MINIMIPS	72	18.402	594.11	594.09	5,346.81
LEON II	192	27.625	992.95	992.92	11,915.01
L80SOC	14	9.928	17.32	17.31	571.21
HP16	47	15.693	217.30	217.27	1,303.67
RS_DEC	78	23.479	96.43	96.41	1,446.11
FPU	17	12.274	1,642.19	1,642.17	9,853.04
DCT	6	9.402	269.63	269.61	2,426.54

Table 3 Critical SEUs-induced cross domains errors

Circuits	X-TMR		Minority-voter based TMR [21]		XNOR-based TMR (proposed approach)	
	CLB	Routing	CLB	Routing	CLB	Routing
RISC5x	22	32	0	13	0	6
CORDICR2P	20	84	0	22	0	12
CORDICP2R	32	932	0	76	0	42
MINIMIPS	12	342	0	24	0	12
LEON II	52	1,230	0	88	0	47
L80SOC	31	86	0	10	0	0
HP16	28	109	0	21	0	14
RS_DEC	32	118	0	30	0	18
FPU	19	273	0	54	0	32
DCT	10	120	0	19	0	12

hardwired nature of the carry-chains, however, in order to avoid the accumulation of upsets in the configuration memory, the proposed methodology can be augmented with a full scrubbing at a considerably low scrub rate compared to the traditional TMR methodologies.

4.3 Cross-Domain Errors

The reduction in the number of CDEs is a consequence of the non-overlapping placement of TMR domains in different reconfigurable regions. These results were obtained by using the STAR tool [22] on the physical-level description format. The STAR tool uses an analytical approach to estimate the effects of SEUs based upon a cross-correlation of the bit-stream to its resource mapping. Table 3 represents the SEU sensitivity analysis for the standard Xilinx TMR with non-overlapping domain placement based methodologies. It can be noted that the number of CDEs are significantly reduced by our methodologies compared to the X-TMR approach. Mainly, two reasons contribute towards the significant reduction of the number of CDEs. First, the non-overlapping domain placement and second the frequent run-time partial repairing. The non-overlapping domain placement nullifies the CDEs for the combination logic in form of LUTs as discussed in section 3.2. The frequent selective run-time repairing greatly reduces the probability of accumulation of SEUs in the configuration memory.

It can be noted that the CDEs induced by routing failures for the Minority-Voter based TMR are more than that for the XNOR-based TMR. This is due to the nature of minority voter which requires three inputs from three different domains. In fact this strong likelihood in terms of interconnections between the majority voters and the minority voters can result in a placement in the same CLB. Although, we constrained the placement in such a way that logic from different domains do not mix up, however, in a domain the placement of majority voter and minority voter is not constrained. This increases the probability that a CDE in the routing can result in an error being undetected when the minority voter is affected. This can cause a fault to linger longer in the TMR circuit resulting in the accumulation of faults and can lead to a functional failure. The use of XNOR gate reduces the probability of a fault being un-detected due to the fact that it uses two inputs from the same domain. Moreover, the carry-chain based detectors are non-programmable and are not controlled by SRAM cells thus enhancing the dependability.

4.4 Effects on Area and Frequency

The area comparison of the proposed approaches with X-TMR is shown in Fig. 7. For each benchmark circuit shown on the horizontal axis, the vertical axis plots the number of LUTs utilized by the design using a logarithmic scale. Consider the case of the CORDIC cores: both circuits have zero area overhead, while the percentage timing improvement is 67 %. This is because both circuits have a single partition and the overhead in this case is just the one stemming from the flag signals taken from the output of minority voters. For the case of two partitions, HP16 has more area overhead compared to FPU because the number of signals is comparable; however, the logic resources in FPU are far more than the logic resources of HP16. This means that for the same number of signal groups and partitions, for any two different designs our method will result in less area overhead for the larger design. Moreover, for two designs with roughly the same area (e.g., RISC5x and L80SOC) our approach will result in a larger area overhead for the design with the larger number of partitions. Concretely, the area overhead is dependent directly on the number of signal groups and on the number of partitions. Also the logic distribution of

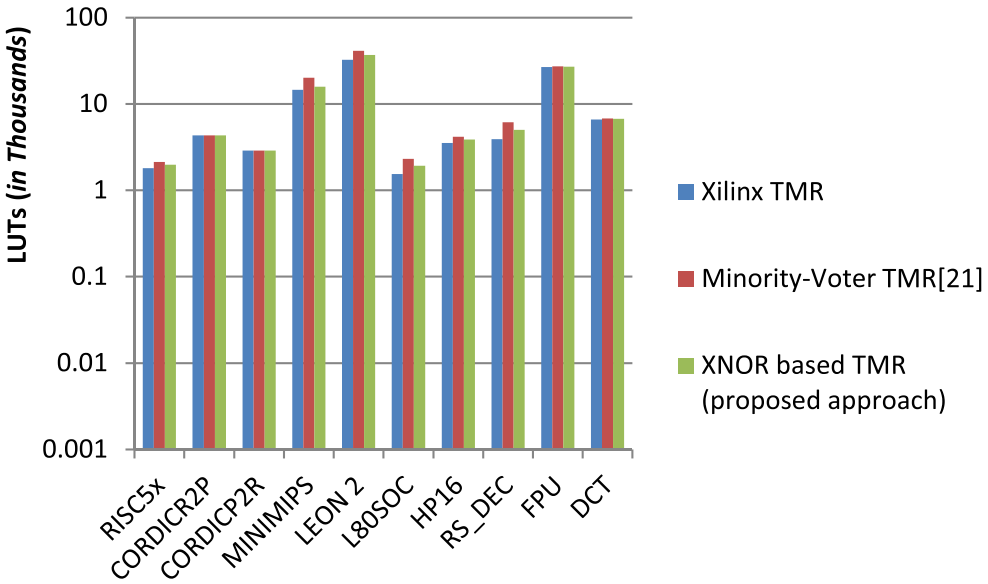

Fig. 7 Area comparison of proposed TMR architectures with X-TMR

resources and area of the original circuits have impact on the area overhead of our method. As discussed in detail in section 3.1.1, the single SEU fault assumption, enable us to remove the steering logic in the minority voter based schemes, therefore, the proposed approach has a less area overhead than the approach proposed in [21].

The effects of the proposed methodology on the operating frequency of the benchmark circuits are shown in Fig. 8. These results were obtained by analyzing the post place and route net-list with the Xilinx timing analyzer tool. For each benchmark circuit the figure shows the clock period for Xilinx TMR, XNOR based mixed floor-planned, XNOR based non-overlapping and Minority-Voter based non-overlapping versions. The plot shows, on the vertical axis the clock period (in nanoseconds) and the corresponding benchmark circuit on the horizontal axis. The Xilinx TMR version is implemented with standard tool-flow and is not constrained in any way. The mixed floor-planned version has the XNOR based error detectors inside but the domains overlap. The minority voter version utilized the physical-level flow and uses a simulated annealing placement algorithm. The custom simulated annealing placer has a poor performance compared to the Xilinx placer utilized by the other three versions. In contrary to our

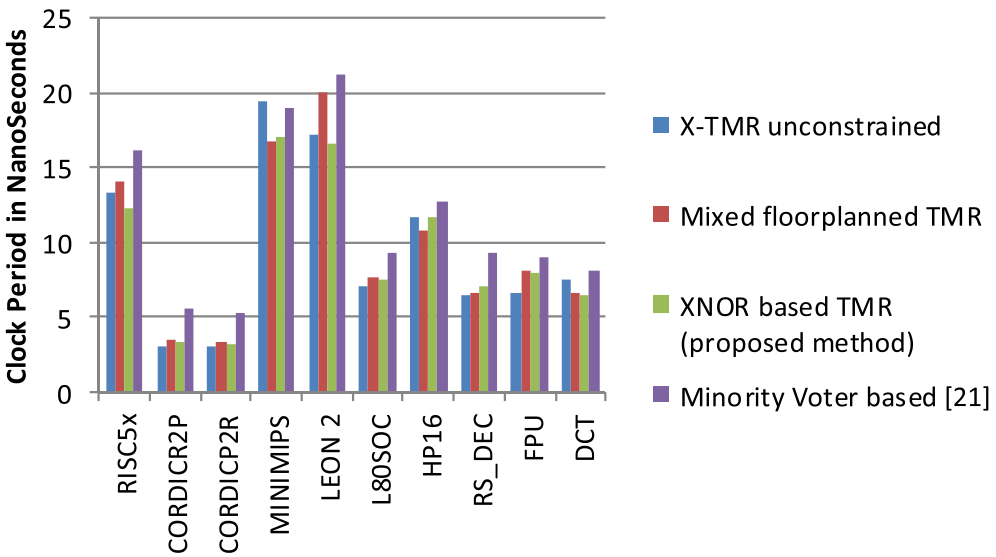

Fig. 8 Effects of the proposed methodology on clock period

intuition, the floor-planning of domains has enhanced the performance of some of the benchmark circuits. The reason for this performance improvement can be contributed to the fact that the manual floor-plan, done in an optimal manner for most benchmark circuits, enables the placer to more rigorously optimize the wire-length due to compact area constraints. It is also very interesting to note that the insertion of very long error detectors in the TMR circuit does not affect the circuit operating frequency. This is due to the usage of dedicated hardwired carry-chains connections, designed for fast arithmetic computations, that never changed the critical path of the circuits.

5 Conclusion And Future Work

A modified TMR architecture is presented in this work in order to improve the recovery time and fault tolerance of TMR circuits implemented on SRAM-based FPGAs without interrupting the real-time operation. Two key modifications are proposed in the TMR architecture and in the placement phase. First, the TMR architecture is instrumented with custom logic at the granularity of individual domains to detect, localize and repair faulty domains. The huge number of comparator check flags generated are merged using the on-chip in-slice carry-chain resources. Second, a novel CAD flow is used for insertion of custom logic and non-overlapping placement of TMR domains. The results show a huge reduction in recovery time and number of Cross-Domain Errors (CDEs) compared to Xilinx TMR with full scrubbing. The recovery time is bounded by the size of maximum partition in a TMR circuit and is orders of magnitude less that the full scrubbing. The area overhead depends on the number of signals crossing the boundaries between any two partitions, on the number of partitions, on the number of logic resources in the circuit, and on its distribution across partitions. Interestingly, the method does not incur performance overheads due to the insertion of error detection logic. An analytical reliability evaluation method is employed for early design phase evaluation in this work; however, in the future it would be interesting to investigate radiation effects under particle accelerator.

References

1. Azambuja JR, Sousa F, Rosa L, Kastensmidt FL (2009) “Evaluating large grain TMR and selective partial reconfiguration for soft error mitigation in SRAM-based FPGAs,”. In On-Line Testing Symposium, Sesimbra, pp 101–106
2. Berg M, Poivey C, Petrick D et al (2008) "Effectiveness of internal versus external SEU scrubbing mitigation strategies in a Xilinx FPGA: Design, test, and analysis,". IEEE Trans on Nucl Sci 55(4): 2259–2266
3. Boost C++ libraries. [Online]. www.boost.org
4. Carmichael C (2001) “Triple module redundancy design techniques for Virtex FPGAs,” Xilinx Inc., XAPP197 (V1.0), November
5. Carmichael C, Caffrey M, Salazar A (2000) Correcting single event upsets through virtex partial reconfiguration XAPP216 v1.0
6. Cetin E, Diessel O (2012) "Guaranteed Fault Recovery Time for FPGA- based TMR Circuits Employing Partial Reconfiguration," in 2nd International Workshop on Computing in Heterogeneous. Autonomous ‘N’ Goal-oriented Environments, San Francisco
7. Champman K, Jones L (2009) SEU strategies for Virtex-5 devices, Xilinx Inc., XAPP864
8. Constraints Guide. Xilinx Inc. [Online]. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_/cgd.pdf
9. Fleetwood DM, Peter S, Winokurb, Doddb PE (2000) “An overview of radiation effects on electronics in the space telecommunications environment,”. Microelectron Reliab 40(1):17–26
10. Iturbe X, Benkrid K, Torrego R, Ebrahim A, Arslan T (2012) “Online clock routing in Xilinx FPGAs for High performance and reliability,”. In IEEE Adaptive Hardware Systems, Erlangen, pp 85–91
11. Jedec Standard (2006) “Measurement and reporting of alpha particle and terrestrial cosmi ray-induced soft errors in semiconductor devices,” Tech. Rep JESD89A, [Online]. http://www.jedec.org/sites/default/?les/docs/jesd89a.pdf
12. Nazar GL (2013) “Fine-Grain Error Detection Techniques for Fast Repair of FPGAs”, Phd Dissertation, UFRGS, [Online]. http://www.lume.ufrgs.br/bitstream/handle/10183/77746/000897120.pdf?sequence=1
13. Nazar GL, Carro L (2012) “Exploiting Modified Placement and Hardwired Rescources to Provide High Reliability in FPGAs,”. In 20th International Symposium on Field-Programmable Custom Computing Machines (FCCM), Toronto, pp 149–152
14. Nicolaidis M (2011) Soft Errors in Modern Electronic Systems. Springer, US
15. Open Cores Repository. [Online]. opencores.org
16. Pilotto C, Azambuja JR, Kastensmidt LF (2008) “Synchronizing triple modular redundant designs in dynamic partial reconfiguration applications,”. In ACM 21st annual symposium on Integrated circuits and system design (SBCCI’08), Gramado, pp 199–204
17. Quinn H, Morgan K, Graham P et al (2007) Domain Crossing Errors: Limitations on Single Device Triple Modular Redundancy Circuits in Xilinx FPGAs. IEEE Trans Nucl Sci 54(6):2037–2043
18. Reorda MS, Sterpone L, Ullah A (2013) “An error-detection and self-repairing for dynamically and partially reconfigurable systems,”. In IEEE Europen Testing Symposium, Avignon, pp 1–7
19. Sellers B, Wirthlin M, Kalb J (2009) “FPGA partial reconfiguration via configuration scrubbing,”. In Field Programmable Logic and Applications, Prague, pp 99–104
20. Steiner N, Wood A, Shojaei H et al (2011) “Torc: Towards an Open-Source Tool Flow,”. In Proceeding of 19th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, pp 41–44
21. Sterpone L, Ullah A (2013) "On the optimal reconfiguration times of TMR circuits on SRAM based FPGAs,". In NASA/ESA Adaptive Hardware Systems, Torino, pp 9–14
22. Sterpone L, Violante M (December 2005) A new analytical approach to estimate the effects of SEU in TMR architectures implemented through SRAM based FPGAs. IEEE Trans Nucl Sci 52(6):2217–2223
23. Virtex-5 configuration user guide. Xilinx Inc. [Online]. http://www.xilinx.com/support/documentation/user_guides/ug191.pdf
24. Virtex-5 FPGA User Guide. (2012) Xilinx Inc. [Online]. http://www.xilinx.com/support/documentation/user_guides/ug190.pdf

Anees Ullah has received BSc and MSc degrees in Electrical Engineering with majors in Communications and Electronics in 2009 and 2011 respectively from University of Engineering and Technology, Peshawar, Pakistan. Since 2012, he is pursuing PhD degree in Department of Control and Computer Engineering, Politecnico di Torino, Italy. His research interests include fault-tolerant digital systems, reconfigurable computing and computer aided design tools for synthesis, packing, placement and routing. His current works are focused on fast error detection, error localization and repairing against soft errors due to radiation effects on SRAM-based FPGAs. He is a student member of IEEE.

Luca Sterpone received MS and PhD degrees in computer engineering from Politecnico di Torino, Italy, in 2003 and 2007, respectively, where he is currently an assistant professor with the Department of Computer Engineering. His research activities focus on the design, validation, and test of safety-critical devices with particular emphasis on computer aided design tool for layout, synthesis and place and route. He has been research intern at Boeing Satellite Systems (El Segundo, California) and at EADS Innovation Works, Suresnes, France, during 2006 and 2007. He is the author of more than 85 papers including several IEEE and ACM Transactions, he published two books on Fault Tolerance Design on Reconfigurable Devices (2010 and 2012). He received several awards for his research activity such as the Best Paper award at the IEEE European Test Symposium (2005) and the EDAA Outstanding Dissertation Award in 2007. He has been a general chair of the HiPEAC Reconfigurable Computing Workshop in 2013, the program chair of the same workshop in 2012 and he serves as member within the program committee of several events such as RADECS, NSREC, ISIE, and DATE. He is a member of the IEEE.