An Error-Detection and Self-Repairing Method for Dynamically and Partially Reconfigurable Systems

An Error-Detection and Self-Repairing Method for Dynamically and Partially Reconfigurable Systems

M. Sonza Reorda, *Fellow*, *IEEE*, L. Sterpone, *Member*, *IEEE*,
A. Ullah, *Student Member*, *IEEE*

Abstract— Reconfigurable systems are gaining an increasing interest in the domain of safety-critical applications, for example in the space and avionic domains. In fact, the capability of reconfiguring the system during run-time execution and the high computational power of modern Field Programmable Gate Arrays (FPGAs) make these devices suitable for intensive data processing tasks. Moreover, such systems must also guarantee the abilities of self-awareness, self-diagnosis and self-repair in order to cope with errors due to the harsh conditions typically existing in some environments. In this paper we propose a self-repairing method for partially and dynamically reconfigurable systems applied at a fine-grain granularity level. Our method is able to detect, correct and recover errors using the run-time capabilities offered by modern SRAM-based FPGAs. Fault injection campaigns have been executed on a dynamically reconfigurable system embedding a number of benchmark circuits. Experimental results demonstrate that our method achieves full detection of single and multiple errors, while significantly improving the system availability with respect to traditional error detection and correction methods.

Index Terms— Self-Repair; Partial and Dynamic Reconfiguration; Single Event Upsets (SEUs); Multiple Event Upsets (MEUs)

1 Introduction

TECHNOLOGY scaling in the nano-metric domain and beyond supports the increasing usage of high performance and miniaturized embedded systems. However, the quest for pushing the limits of technology to the ultra-nano scale devices has exacerbated concerns related to power consumption and reliability that have not be envisioned before. In particular, one of the major issues in safety-critical applications (especially in the space and avionic domains) is the run-time mitigation of various radiation-induced fault effects, which may provoke transient and permanent modifications of the electronic circuit's behavior. The problem is widely known and various methods have been developed and proposed in the area during the last decade. The ubiquity of embedded systems for safety-critical applications operating in radiation environments demands continuous and successful operations of the system by autonomously overcoming possible malfunctions. This condition requires the abilities of autonomous error detection, self-diagnosis and self-repair [1].

Among the available technology solutions, the adoption of SRAM-based FPGAs is the most suitable for the realization of dynamically and partially reconfigurable systems; however, when used in harsh environments, SRAM-based FPGAs have to withstand the radiation effects in the form of Single Event Upsets (SEUs) and Multiple Event Upsets (MEUs), especially affecting their configuration memory [2].

The increased probability of MEUs hitting the configuration memory of an FPGA can limit the effectiveness of traditional redundancy-based fault-tolerance approaches [3]. In fact, particles can hit the same logic group of circuit replicas enabling erroneous results to propagate. To cope with this scenario, researchers have recently investigated the fine-grain redundancy and its resilience to MEUs [4][5][6]. However, for proper shielding against high failure rate while minimizing redundancy overhead in terms of area, speed and power consumption, systems should be designed with accurate mixed-grain redundancy and self-repair properties which are not feasible for fine-grain redundancy.

State-of-the-art SRAM-based FPGAs have the technology supporting run-time dynamic and partial reconfiguration (DPR), which can be used for adaptive behavior as well as for fault repairing [7]. A self-repairing system adopting the partial dynamic reconfiguration capabilities of SRAM-based FPGAs is often divided in two parts, called *static region* and *dynamic region*. The logic and routing resources and the corresponding configuration memory frames individuated by means of clock regions and major and minor columns, illustrated in figure 1, are organized in a *static region*, also called *base region*. The static region typically consists of a microprocessor, some memory modules and input/output ports, as described in figure 2. In general, these components are not reconfigured and their full functionality is constantly required for implementing the correct operations of the system; for this reason the static region is often hardened using a traditional redundancy-based approach, such as Triple Modular Redundancy (TMR) [7]. The static region is also responsible for the reconfiguration of the modules placed into the reconfigurable regions. On the contrary, the components in the *dynamic region* correspond to partially reconfigurable resources that can be configured in different ways depending on the system requirements [8]. The dynamically reconfigurable regions idea is extended in this paper so that the system is able to also correct the identified errors by applying internal reconfiguration.

M. Sonza Reorda, L. Sterpone and A. Ullah are with the Dipartimento di Automatica e Informatica (DAUIN), Politecnico di Torino, Torino, Italy. For any information please refer to:
luca.sterpone@polito.it (contact author).

The proposed approach provides significant advantages compared to already developed solutions [9][10], mainly because it increases the error detection and correction capabilities while introducing comparable area and performance overhead.

In order to practically prove the effectiveness of the approach, we developed a complete set of tools for the automatic generation of the constraints used for the partitioning of the dynamic regions. The developed set of tools directly acts at the physical level, automatically inserting a carry chain into the physical net-list and adding comparator check flags into the circuitry; moreover, the tool is able to cleverly place the different partitions of the dynamic region into proper sub-regions, thus allowing SEUs and MEUs correction. The proposed approach drastically improves the solution in [9] which uses the built-in slice carry chain for error detection, only.

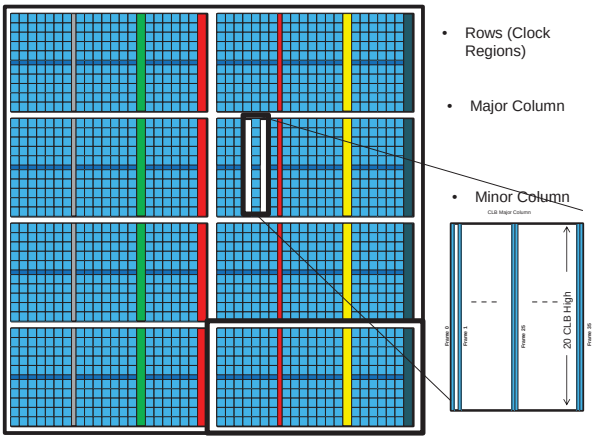

Fig. 1. Resource and Frame Layout of Modern SRAM based FPGAs

Our approach introduces a minimal area overhead, which is strictly dependent upon the number of user-defined partitions. On the average, the overhead introduced by our approach is around 11% with respect to the duplication-based approach; hence, the proposed technique is using far less computational resources if compared to the standard TMR solution. Furthermore, correction is performed on a single reconfigurable frame, which is the smallest amount of reconfigurable information that can be read or written; therefore, we can achieve the highest availability limits offered by the current reconfigurable technology.

The paper is organized as follows. Section 2 gives an overview of the soft error detection and correction methods implemented with modern SRAM-based FPGAs and summarizes the major contributions of this paper. Section 3 describes the proposed method, while the developed design flow is illustrated in Section 4. Experimental results on the selected case study and their analysis are presented in Section 5. Finally, conclusions and future works are described in Section 6.

2 Previous Works

State-of-the-art SRAM-based FPGAs are heterogeneous devices containing several macro blocks, like Digital Signal Processors (DSPs), Block RAMS (BRAMs) and IO Blocks (IOBs), along with Configurable Logic Blocks (CLBs) inside the FPGA reconfiguration fabric. Each of these resource types is arranged in columns that span from top to bottom of the device realizing a column of CLBs, IOBs and BRAM memories interconnected by a mesh of heterogeneous routing resources. Each SRAM-based FPGA chip is organized in a number of rows dependent on the manufacturer families or specific part. The most advanced devices have CLB rows connected to global resources as well as local clock sources [11]. In order to harden circuits implemented on SRAM-based FPGAs, different architecture level techniques have been proposed in the past. However, we can broadly classify them into two main techniques, namely *fault masking* and *fault correction*. In the next part of this section we will present a detailed discussion of the previous research work in each category.

In the recent years, two different mitigation approaches have been proposed to mitigate SEUs affecting the configuration memory of SRAM-based FPGAs. On one side, full hardware redundancy obtained thanks to Triple Modular Redundancy (TMR) is used to identify and correct logic values. This solution presents a large overhead in terms of area, power and especially delay, since it triplicates all the combinational and sequential logic, and the architecture introduces delay penalties for the voter propagation time and the routing congestion. On the other side, redundancy approaches are nowadays combined with *scrubbing*, that consists in periodically reloading the complete content of the FPGA's configuration memory. A more complex system is used to correct the information in the configuration memory by using read-back and partial configuration procedures. Through the read-back process the content of the FPGA's configuration memory is read and compared with the expected value, which is stored in a dedicated memory located outside of the FPGA. With the advent of modern SRAM-based FPGAs this operation may be performed through dynamic reconfiguration. In details, with dynamic reconfiguration, the FPGA configuration memory can be read-back continuously without interfering with the circuit functionality and if any upset is detected it can be selectively re-written with the correct values, thus avoiding the accumulation of radiation-induced errors [12]. However, the main drawback of this technique is the huge detection and correction time that makes it useless for real-time operations and ineffective versus the single point of failure induced by configuration memory bit-flips. Spatial redundancy using Triple Modular Redundancy (TMR) is complementarily used with the read-back and correction techniques: on one side TMR can tolerate faults with the limitation of withstanding a single fault per voting group [13], on the other side read-back and correction avoids the accumulation of errors within the configuration memory. The combination of TMR and self-healing using dynamic partial reconfiguration has been previously used in [14][15]. However, the results achievable with this combined solution are computationally expensive and area hungry.

Reconfiguration at the gate level is used in fine-grain approaches [16] with particular efficiency from the point of view of the area overhead, although it suffers from a complex and not flexible control mechanism. Furthermore, because of the adopted fine granularity, this approach is infeasible for system-level healing. A self-healing partial dynamic reconfigurable design methodology has been proposed in [17]. However, the method inserts control circuitry by partitioning the circuit for error

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2016.2607749, IEEE Transactions on Computers

localization and detection purposes. This requires a significant overhead.

A methodology for fault tolerant architectures using on-line checkers for fault detection and localization was introduced in [18]. On-line checkers for TMR- and duplication-based systems were combined with partial dynamic reconfiguration in [19] in such a way that detection and localization of faults will be performed by the checker, while reconfiguration will recover from the error. The detection and localization of errors is implemented as a partial reconfigurable module which is itself subject to errors. A previous approach based on fine-granularity error masking has been developed in [20]; however, such solution can only be applied to a TMR technique with a majority voter logic scheme. Vice versa, a first overview of recovery architectures for high computational systems based on SRAM-based FPGAs has been presented in [21].

2.1 Main contribution

The main contribution of the present work, which is based on the platform preliminarily presented in [8], is the description of an autonomous recovery approach that can be applied to Partially Reconfigurable Modules (PRMs) when errors are detected inside them. The approach is implemented by the static region providing effective capabilities of error detection and correction of faults within the dynamic region. Our approach allows resilience to MEUs, since we adopt a static region protected with a fine-grain redundancy approach, as described by [3]. In particular, we propose a new fine-grain fault detection mechanism applied to FPGA resources: the approach is based on the comparison of Look-Up Tables (LUTs) outputs by using the logic available to allow carry propagation, which is generally used for fast arithmetic computations and mostly not inferred by design tools, following the approach preliminarily introduced in [9] for fault detection. In details, the proposed method is characterized by the ability of detecting MEUs into the FPGA's configuration memory, as well as to recover any number of faults in the dynamic partition, thus improving previously developed approaches, as presented in [9], that cannot deal with MEUs. Our solution is adaptable to all modern SRAM-based FPGAs equipped with an Internal Configuration Access Port (ICAP) and based on a LUT-slice architecture.

3 The Proposed Method

The proposed method consists of two flows: one applied to the dynamically reconfigurable region for implementing error detection, the other one for instrumenting the circuit mapped on the FPGA so that it supports the execution of the self-repairing method against single and multiple-bit errors.

A dynamically reconfigurable system, from the architectural perspective, is partitioned into static and dynamic regions as illustrated in figure 2. The *static* region consists of a processor with a static-RAM, some general purpose IOs, flash memories, and hardware resources for managing the internal configuration access port connected to the processor local bus. The static region contains the main processor, which is in charge of controlling the partially reconfigurable system operational functionalities: therefore, it is very important to tolerate and recover errors in these modules. In this paper we assume that this region is implemented using Triple Modular Redundancy. By suitably mapping the three copies of the circuit elements on the device the static region can be protected against any single point of failure.

The dynamic region consists of the resources implementing the user's circuit. The proposed approach mainly focuses on the dynamic region, and exploits reconfiguration at the individual frame level for error detection and correction. The dynamic region can be organized into a Single Bit Error (SBE) region, Multi Bit Error (MBE) region and Coarse-Grain Error region. It is first necessary to introduce some definitions related to the major characteristics of current SRAM-based FPGAs: modern FPGAs are row-wise divided into a number of clock regions for dynamic partial reconfiguration, while column-wise are organized in major columns of resources, such as CLBs, DSPs or IOs. Each major column spans the whole height of the device but it is configured in each clock region (row) by a separate reconfigurable frame (RF). Each RF contains a different number of "minor frames", each having a height equal to the clock region (row) and numbered from left to right. For example, in Xilinx Virtex-5 devices [11] a CLB RF consists of 36 minor frames (hereby simply referred to as *frames*), which are responsible for the configuration of LUTs and their routing, while configuration bits for a single LUT are distributed over multiple frames. From the point of view of the circuit architecture, the proposed method is based on the Duplication With Comparison (DWC) technique applied at two different levels of granularity, herein called *Coarse-grained DWC* (C-DWC) and *Fine-grained DWC* (F-DWC).

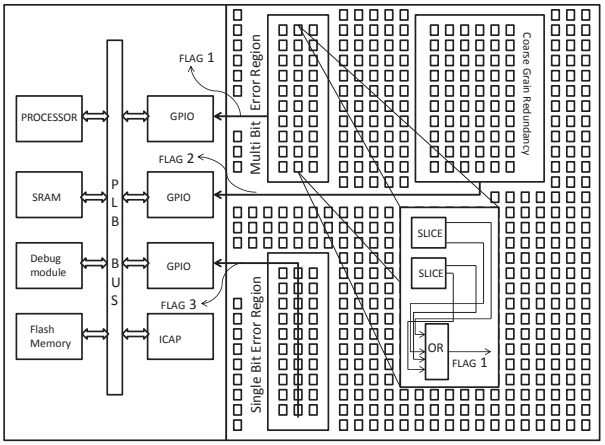

Fig. 2. Placement space division into *Single Bit Error region*, *Multiple Bit Error region* and the *Coarse Grain Error region*.

The C-DWC is applied for slices that use the carry chain for computations such as fast additions or multiplications. In this case, the duplication is performed at the module level and the outputs are compared at the physical level by LUT elements configured to implement XOR combinational functions. Our approach is able to directly modify the circuit physical description in order to use the XOR logic function to compare the module's outputs. In case of error, the software tools running on the reconfigurable system partially rewrite the C-DWC region.

F-DWC is applied at the place and route level, by suitably duplicating each LUT function in two copies that are placed in a single slice using two consecutive LUT positions. The outputs of the two LUTs are then compared with hardwired physical resources built into the slice in

the form of a carry chain by using internal and not programmable resources, such as hardwired MUXCYs and XORCYs.

The outputs generated by the XORCY functions are connected in a chain of OR logic functions in order to provide a single error detection flag for each column. Practically, the F-DWC approach can be adopted by acting at the Hardware Design Language (HDL) level: the combinational functions are duplicated and both copies of the circuit LUTs are placed in a single FPGA slice using two consecutive available LUT positions. Please note that the outputs of any pair of LUTs pass through the carry chain and at each pair position of the XORCY generate a comparison signal called *check flag*. Since we are generating a check flag for each pair of LUTs the number of check flags may drastically increase. This means that a considerable amount of routing resources could be required by the implementation of these check flags because they have to be routed to the *static region* for the detection and correction of possible errors. Moreover, any such scheme will not only have a large overhead, but it will also be fruitless because the smallest unit of reconfiguration is a frame.

In order to have a single check flag for each frame we propose to merge the individual check flags in two different ways. The check flags in the SBE region are merged through the built-in slice carry chain as shown in figure 3 (further details will be provided in section 3.1.1) where the hardwired resources XORCY and MUXCY are labeled as Xi and Mi, respectively. Furthermore, a whole column of slices is connected through the carry chains to produce a single flag for each column of slices (see for example *flag 3* in the SBE region of figure 2).

In this way we achieve a huge reduction in the number of check flags, but we can only detect Single Event Upsets (SEUs) in the SBE region, because multiple LUT pairs are connected together by a long chain of XORs and XNORs and thus an even number of errors will go undetected due to the logic configuration of the detector.

In the Multiple Bit Error (MBE) region each pair of LUTs generates a check flag and thus we have two check flags per slice. The number of check flags can be reduced by OR-ing some of the flags corresponding to the slices in the same slice column, as shown in the magnified MBE region in figure 4. Although some higher overhead is introduced in this way, we have the ability to detect Multiple Event Upsets (MEUs) in the frames mapped on this region; in fact, the individual check flags are not merged along the carry chain passing through multiple XORs, as it happened in the SBE region.

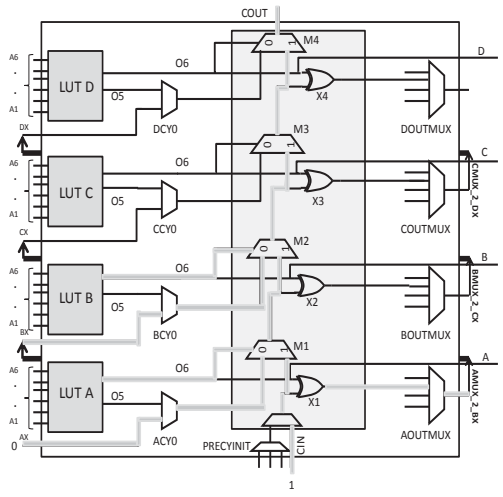

Fig. 3. Single-Bit Error detection scheme implemented in a single slice.

3.1 Error Detection Method

In order to fully explain our proposal, in this section we will specifically refer to the architecture of Xilinx Virtex-5 FPGAs. As described in the previous section, the error detection mechanism implemented in the reconfigurable region is based on LUT-based checkers and carry chains for propagating the check flags. Please note that the LUT checkers are only deployed when the carry chain is unavailable for comparison purposes. This allows reducing the performance degradation of the circuit implemented with our method, although in this case the detection mechanism is implemented at the modular level. In this section, we focus on the method adopted for the error detection using the carry chains for comparison; a more detailed explanation of both the LUT checkers and the carry chains insertion inside the physical place and route description of the circuit will be given in Section 4.3.

3.1.1 *Single-bit error detection*

In order to detect single-bit errors, we propose to duplicate each original LUT function into two identical LUTs. Furthermore, we place the two LUTs in a single FPGA slice, where we set the Carry Input and the generic AX inputs to 1 and 0, respectively, as illustrated in figure 3. Consequently, the hardwired XORCY logic gate in the bottom of the slice is acting as an inverter, while the MUXCY multiplexer in the bottom first position is simply acting as a buffer to pass the value of LUT A.

The multiplexer "M2" receives an inverted (through the AMUX_2_BX hardwired connection) and buffered copy of the LUT A output at its "0" and "1" inputs while the selection line is tied to LUT B (which is the copy of LUT A) thus effectively performing the EX-NOR function. The XOR gate named "X2" receives LUT A and LUT B outputs on its inputs. Similarly, LUT C and LUT D can also be connected with such a scheme by extending the EX-NORs and EX-ORs along the slice. In fact, this scheme can be extended to an entire clock region covering 20 CLBs using the COUT and CIN of slices, thus generating two flags for the even and odd slice columns of the same CLB, respectively. This convergence strategy can only be applied if the CLB column has no empty slices. In case the CLB column contains empty slices the dedicated COUT connection cannot be used to propagate the flag signal upwards along the column. For such a case, an ORing LUT is introduced in the CLB column and placed in an available empty slice. This will be discussed in greater details in Section 4.3. It is interesting to investigate an upper bound on the number of check flags that can be generated for the most complex design. The flag signal is generated per CLB tile columns and is directly related to the device rows and columns. For example, for the Virtex-5 VLX110T device the maximum number of check flags for any design cannot be greater than 1,280 (160x8) [22] [23]. As the FPGA must contain the control processor the actual number will be quite less than 1,280 and will determine the size of the GPIO port that is used by the controller to detect errors. Then, it is possible to pinpoint single bit upsets in any of the four LUTs in any slice column in a clock region. However, errors affecting flip-flops

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2016.2607749, IEEE Transactions on Computers

cannot be directly detected. Instead, they are detected when their effects propagate to the next logic levels of LUTs, but we still cannot pinpoint the configuration frames required for correction and thus a larger area has to be reconfigured to deal with errors in flip-flops (FFs).

3.1.2 Multiple-Bit error detection

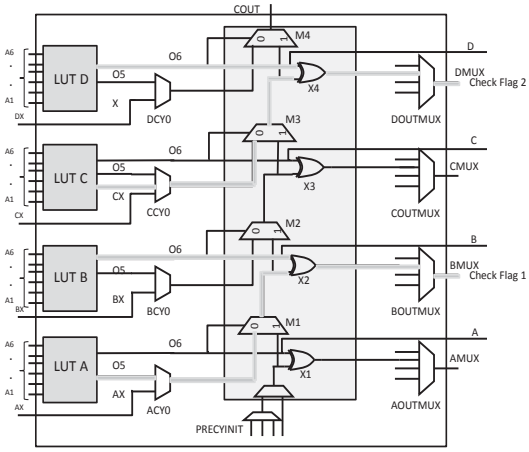

Fig. 4. Multiple-Bit Error detection scheme implemented in a single slice.

In case a further protection level is needed, our approach is able to provide detection against multiple-bit errors in a slice column. However, multiple bit errors can only be detected if the error detecting carry chain is inserted in a specific pattern that we will mention in this section. Furthermore, the logic connectivity pattern required dictates that the LUTs used in this scenario have 5 inputs or less. Therefore, all LUTs in the design having a number of inputs lower than 5 can be designated to this region. It is interesting to know that a recent survey of designs for Xilinx FPGAs suggests that 37% of LUTs inferred by the synthesizer have less than 5 inputs [21]. The scheme implemented in order to achieve multiple-bit error detection is shown in figure 4, where the output of the LUT A is connected to the output O5, while the output of the LUT B is taken from O6. The two signals O5 and O6 are internally hardwired to the inputs of the XOR gate in the second LUT position, which acts as an error detection logic. Please note that we have to tie up the "A6" input of LUT A since we intend to activate both the outputs O6 and O5 and this feature is technically achieved by connecting it to the "TIEOFF" element. In this way every pair of LUTs in the design generates a check flag signal, as shown in figure 3 (check flag 1 and check flag 2). However, the number of flags grows linearly with the number of LUTs in the dynamic region, and so the method becomes unfeasible for large designs. In order to reduce the number of flags we propose the usage of 2 slices (out of the available 20) for merging the check flags by OR-ing them. In fact, two levels of OR gates placed in the bottom two slices of a slice column are used as a flag reduction strategy. As we are producing two flags for each clock region (one for odd and one for even slices) we can have a maximum of 72 LUTs (out of 80 LUTs in an even or odd slice column) configured for computations in any slice column location (even or odd) within a single clock region. Thus, the MBE regions require an overhead of 11.11 % for flag reduction.

3.2 Error Correction Method

The error correction method we propose is based on the assumption that the DUT Dynamic Regions are physically placed at preliminary known positions at the design time and that the original configuration memory frames are stored into an external memory in such a way that they can be retrieved when an error on the corresponding flag from the DUT region is signaled. Data errors affecting combinational logic or Flip-Flops are individuated by the error detection scheme previously described. The error signal can be used to drive the processor interrupt signal: in case an interrupt is triggered from any flag, the main processor controller responds to the interrupt in the following way. First, it determines the clock region (in terms of CLB row) and the slice position (even or odd) the error was triggered by. This is particularly relevant, since the main processor controller needs to selectively reconfigure the faulty frames of the DUT design with the correct copies of the corresponding frames that are stored in the outside memory. Secondly, the clock enabling signals should be de-activated to disable the propagation of errors to the next stages in the design. This is possible since both static and dynamic regions have well-defined interfaces with clock enabling registers. Similarly, the DUT region should also have latched outputs at every stage of the design. Thirdly, the main processor controller accesses the configuration layer of the FPGA and reconfigures the faulty region with the golden copy of each frame. Lastly, the main processor controller enables the clock to re-start the normal operation in the DUT region involved in the correction.

4 Design Flow

In this section we describe the tool flow we developed in order to insert fine-grain duplication with comparison using the built-in slice carry chains. The developed flow is shown in figure 5, and consists of two phases applied at the design mapping process. A *pre-map* step generates a number of constraints for directed packing, placement and sites prohibitions, while a *post-map* step inserts the error detecting carry chains and the convergence logic required to reduce the number of flag signals. This post-map modification is implemented by modifying the XDL file (i.e., the Xilinx interface for interacting with the Xilinx CAD flow). The tool flow has been developed as a C++-based software environment making heavy use of boost library and Tools for Open Source Reconfiguration (TORC).

4.1 Net-list Extraction

The flow starts by parsing the net-list description of the circuit implemented into the dynamic region, which was duplicated at the Hardware Description Level (HDL). It is important that both instances of the design should be labeled with "inst1" and "inst2" so that each synthesized element contains the hierarchical information of the top level instance to which it belongs. However, global reset/clock signals are not duplicated at the module-level, as it will be explained in Section 4.2. After synthesizing the design, a post-synthesis simulation Verilog file is generated using the Xilinx NETGEN utility. The post-synthesis Verilog file contains the circuit net-list using the Xilinx primitive cell library elements. The parser module

traverses the post-synthesis Verilog file and extracts the netlist of the circuits in form of a graph composed of edges and nodes, where edges represent netlist interconnections, while each node represents a logic behavior element (i.e., a logic gate or a flip-flop). In details, each node of the graph corresponds to a data structure with a number of fields including: functional string, instance name, inputs vector, outputs vector and type of primitive element (LUT or FF). When a new type of elements is encountered on the current line a new node is dynamically created and parsing the consequent lines populates all parameters of this node. After all the nodes are properly initialized we iterate over the nodes of the graph connecting one node's output with the inputs of the proper nodes.

4.2 DUT Regions Formation and Constraints Generation

Once the circuit net-list is created in the form of a graph, it is necessary to generate user constraints, represented within the User Constraints File (UCF) in order to perform the DUT physical space division into regions and for packing the primitive cells into slices. The information obtained from the net-list is parsed in a hierarchical manner individuating the couples of LUTs that have been already duplicated at the HDL level. The hierarchical organization consists of three categories of components. Firstly, all the components that use MUXs and XORs are grouped together for modular duplication without carry chain usage. Secondly, LUTs with less than 5 inputs are grouped together to form multiple error regions. Thirdly, LUTs with 6 inputs are grouped to form single bit error detection regions. The division of flip-flops into groups is dependent upon the LUTs to which each flip-flop is connected. It is possible to identify two cases: in case a LUT output is connected directly to a FF input it is possible to individuate a LUT-FF pair; vice versa, if a LUT output is connected to another LUT input a LUT-LUT pair is considered.

Considering this classification, each LUT-FF pair identified within the netlist is stored in a way that each FF is indexed by the corresponding LUT within the same pair. Similarly, a LUT-LUT pair is created each time a LUT used in the design is related to a LUT in another instance. These LUT-LUT pairs are necessary for fine-grain comparison and need to be packed together in the same slice. A slice object is modeled to house the packing of LUTs and FFs. The slice object model is an abstraction of the actual slice on the corresponding FPGA architectures and considers the key attribute, for example the number of LUTs, FFs and the unique clock and reset signals at the input of the slice. The clock and reset signals (along with Clock enable, Set/Reset signals) are important constraints for the application of our method because each slice can use a unique clock/reset signal that should be used by the FFs residing in the slice. For this reason the global clock and reset signals were not duplicated due to the architectural limitation of state-of-the-art FPGA devices. Currently, the approach picks up a LUT-LUT pair and its corresponding FFs and checks the legality for packing them in a single slice object and packs them together if the legality constraints are fulfilled. The legality constraints mandate that the FFs should use the same clock/reset signal, although the global clock/reset signals are unique.

For the purpose of this work, the synthesizer is not constrained in any manner for the duplication performed

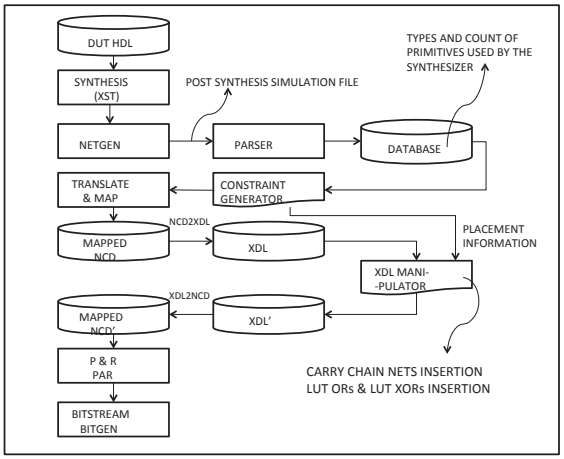

Fig. 5. The developed design flow.

at the HDL level; different duplication strategies applied at the synthesis level may allow the FSM to use unique control signals.

Once the graph is fully annotated with the resource division information, a constraint file is generated that uses the Xilinx packing constraints (*XBLKNM*) for packing and for forming area groups (*AREA_GROUP*). It is important to note that each XBLKNM constraint uses a unique slice name by concatenating the region identifier with an identification number. For example, slices in the single bit region use names like "SBESlice1", "SBESlice2" and so on. This naming convention is necessary for forming the area groups and also for identification at the XDL level, as it will be discussed in Section 4.3. After the completion of this packing step, each slice is included in an area group of either a SBE region or a MBE region, depending on the slice name assigned to it during the packing process. Furthermore, each area group is floorplanned by selecting a slice range constraint to which it will be mapped. The numbers of slices required are calculated from the number of graph nodes that fall in a certain group. For multiple bit error regions, a number of CONFIG PROHIBIT constraints are generated for each slice tile. These constraints are generated in such a manner that the top two slices of each tile column are left empty. It is important to note that both the packing and the floorplanning steps have not been optimized in our case and this can have considerable effects on the circuit operating frequency. Once the constraints are generated, the translation and mapping processes are executed and produce the mapping of LUT copies in the same slice, as illustrated in figures 3 and 4. The routing and check flag signal insertion are performed in the following phase.

The algorithm illustrated in figure 6 performs the generation of the constraints used for the floorplanning of the circuit including the mapping of the SBE and MBE regions. It consists in three phases. The first one, *netlist-partitioning*, elaborates the Directed Acyclic Graph (DAG) of the circuits identifying the type of LUT resources and allocating them into the SBE and MBE groups. The second phase, *slice formation*, composes the slice resources connecting LUTs and FFs. The final phase, *constraints generation*, creates different mapping groups that will be included in the User Constraints File (UCF). All the groups are generated in the form of relative area constraints, thus referring to resource map not placed on the FPGA array,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2016.2607749, IEEE Transactions on Computers

besides, a prohibit constraints group is generated in order to avoid overlapping placement between SBE and MBE regions.

```
1:  DAG = parser(Verilog_netlist);
//  ****************    Netlist-partitioning   ********************//
2: foreach node ‘n’ in DAG.nodes
3:          if(‘n’ is a LUT node)
4:            if(child node of ‘n’ is a carry chain element)
5:             Modular_LUTs.add(‘n’);
6:            if(‘n’ is a 6 input LUT) // Single Bit Error Region
7:             SBE_LUTs.add(make_pairs(‘n’, find(duplicate LUT in DAG.nodes))
8:            if(‘n’ is 5 or less input LUT)
9:             MBE_LUTs.add(make_pairs(‘n’, find(duplicate LUT in DAG.nodes))
10:         endif
11: endforeach
//   ***************   Slice Formation     *************************//
12: foreach LUT pair ‘p1’ in SBE_LUTs/MBE_LUTs
13:            p2  = random_pick_up_LUT_pair (SBE_LUTs/MBE_LUTs – p1)
14:            ff_pair1 = findconnectedFFs(p1)
15:     ff_pair2 =  findconnectedFFs(p2)
16:            slice_object = create_slice_instance(p1,ff_pair1,p2,ff_pair2)
17:            if(check_legality(slice_object))
18:                                 sbe_slices/mbe_slices.add(slice_object)
19:            else go to step 21
20: endforeach
//  ***************   constraints generation ***********************//
21: ucf_file = generate_packing_constraints(sbe_slices/mbe_slics);
22: sbe/mbe_clb_resources = calculate(sbe_slices/mbe_slices)
23: ucf_file = generate_area_group_constraints(sbe_clb_resources)
24: ucf_file = generate_area_group_contraints(mbe_clb_resources)
25: ucf_file = generate_site_prohibit_constraints(mbe_clb_resources)
```

Fig. 6. The flow of the constraint generation algorithm.

4.3 Low-level Manipulations

Once the mapping is performed, the insertion of the carry chain and the definition of the comparator resources are implemented by modifying the physical place and route description of the circuit in order to properly use the hardwired combinational gates. This process is executed in the following distinct steps.

The carry chain insertion step is applied to the slices where the carry chain primitives are not used for fast arithmetic computation. The insertion mechanism of the carry chain is the same for the single- and multiple-bit error detection scheme; however, they differ in multiplexer's settings and wiring details. Each inserted carry chain is labeled with a unique reference to differentiate it with respect to the ones used for arithmetic computation. Each carry chain that spans multiple slices along a column forms a *Relatively Placed Macro* (RPM) identified by a macro number and an individual carry chain number (for example, Shape_0:0,0 and Shape_0:0,1 carry two carry chains that belong to RPM Shape_0, each one identified by its position in the RPM chain). First, the XDL file should be checked to generate a unique RPM identification name for the next carry chain to be inserted. Single bit and multiple bit regions use two different kinds of error detectors, since the flag reduction strategy differs for both cases. For single bit regions, the flags are merged using the dedicated COUT line that runs from a slice to the next slice. Each XDL instance that belongs to a single bit error region is augmented with a carry chain, each one using a unique name, as discussed previously. Once the carry chains are generated, the multiplexer's settings for each single bit error region is performed according to figure 3.

Now that each single bit error region slice contains the carry chain for error detection it is necessary to connect the carry chains along the column to converge the error flags. In fact, at this stage the number of error detection flags is linearly proportional to the number of slices used in the single bit error region (which is typically huge) and it is impractical to route them all to the control processor. The placement was constrained in the previous step with a generated UCF file; however, the placer stills tries to optimize for timing and does not consider the fact the each slice column should be filled to the maximum extent possible resulting in a placement such that the single bit region slices columns have empty slices positioned in the middle of slices that use carry chain-based detectors. This is a serious problem for the flag convergence because the dedicated COUT line can only be used if the slices are positioned in consecutive positions above and below each other. In order to solve this problem, multiple carry chain detectors were combined using an OR LUT generating a single flag signal per CLB tile column. It is also interesting to note that for each OR LUT an automatic procedure searches for an empty slice in the same CLB column and picks up the nearest one in terms of the slice site distance for the OR LUT placement. OR LUT placement is a necessary step because for the Virtex-5 FPGA architecture the placement occurs as a part of the MAP process and should be completed before the PAR routes the design. In this way, the single bit error region flags are converged resulting in error detection carry chains of varying lengths. The higher the length of a carry chain, the larger the error detection latency will be, as it will be discussed in the experimental results Section. For multiple bit error regions, the carry chains used for error detection use OR LUTs in a manner similar to single bit regions. However, the number of flags to be merged is quite large compared to the single bit region as each slice generates two flags. Therefore, the slice sites that were prohibited from usage by the constraints in Section 4.1 are utilized for the placement of OR LUTs. However, in some cases it is possible that there are some empty slices that are not utilized by the placer and those can be used for OR LUTs based on the metric of close proximity. The area overhead introduced by the OR LUTs in the design utilizes the post-placement empty slices and can be reduced if the slice tiles are filled to the maximum extent possible. However, this situation can cause a routing congestion and resultantly the routing time will increase considerably.

The last phase of the low-level manipulation consists of inserting nets with the source and sinks added to them. Nets are added for all the components that have been previously inserted in the form of carry chain-based detectors or in the form of OR LUTs. The routing implementation will be performed by the Xilinx PAR tool that automatically routes all the nets between the inserted components and adds the precise interconnection segments that will be used for routing. It is important to note that if the placement is confined too much the router will face congestion problems and it is possible that the router may take a very long time or in the worst case will be unable to complete the routing of the design. Therefore, the placement should be such that an optimal balance between the usage of OR LUTs for flag convergence and the routing congestion is achieved.

5 Experimental Results

We implemented the proposed method targeting a Xilinx Virtex-5 LX110T SRAM-based FPGA. We designed a dynamically reconfigurable system where a Xilinx Microblaze processor core was mapped into the static region, while the design under test was mapped into the dynamic region. Although other controller solutions exist for managing the reconfiguration (e.g., based on an ad-hoc hardware unit), we adopted the Microblaze processor since it represents a state-of-the-art solution for a dynamically and partially reconfigurable system based on static and dynamic regions [4]. For design validation and fault injec-

tion, the same Microblaze processor is used to apply the test patterns to the DUT and to read the outputs from the DUT through the GPIO port. Moreover, another GPIO port connected to the flags stemming from the DUT region and configured in interrupt mode is responsible for informing the Microblaze in case of errors. After the bit-stream is downloaded to the FPGA the Microblaze memory needs to be initialized with a golden copy of the DUT bit-stream by reading the configuration area of the DUT with the Xilinx hardwired Internal Configuration Access Port (X-HW-ICAP).

As the placement of the different regions was made at design time the Microblaze processor uses that information to build up a bit-stream database for different error regions, as discussed in Section 3. For F-DWC regions, the bit-stream is stored in such a way that if an error is detected by a given flag, the frame can be recalled to reconfigure the affected area. However, the bit-stream for the C-DWC region is stored as a partial bit-stream by reading it with the ICAP from the start address to the end address. In the following sections, we present several results mainly related to the ability of quick error detection, localization and repairing. Besides, a measure of the cost in terms of area overhead and speed is also presented.

5.1 Area Overhead

We selected a set of circuits as benchmarks and proof-of-concept for our approach. The circuits include some relevant ITC'99 benchmark circuits with various complexity, two implementations of the CORDIC arithmetic processor, a miniMIPS processor, a lightweight 8080 SoC, an RS-Decoder and a DCT core from the opencores repository [24] [25].

In Table I, we reported the number of LUTs and FFs of each circuit. We compared the amount of resources used by our approach with the resources used by the original circuits implemented without any error detection or mitigation techniques and with a detection mechanism based on the duplication with comparison using Double Modular Redundancy (DMR) and with a detection and mitigation mechanism based on Triple Modular Redundancy (TMR). Please note that we did not include the amount of resources related to the static region within the area count since the static region remains the same in any Dynamically Reconfigurable system, no matter the adopted solution. Furthermore, please note that DMR or TMR implementations do not directly lead to a duplicated or triplicated resource count, since the redundancy is applied at the pre-synthesis level.

The resource usage figures show that our approach is far better than TMR, since on the average TMR requires 3.64 times more hardware resources than the original circuit, while our approach requires 2.10 times more resources, only. If compared with DMR, our approach requires 10% more resources on the average; however, DMR cannot correct errors, while our approach corrects errors and reduces the probability of single points of failure thanks to the developed fine-grain combinational logic infrastructure.

We underline that the area comparison has been performed directly on the basis of LUTs and FFs counts; if comparison is made considering the number of FPGA slices, the ratio may by slightly different due to stringent packing and placement requirements adopted for the fine-grain redundancy with comparison logic. In particular, slices are used as a route-through and FFs may be placed in separate slices, since the FFs require different control signals that could not be packed together with LUTs.

5.2 Error Detection Latency

The measurement of the error detection latency is the key factor for making a proper self-repairing system able to autonomously repair itself obeying to real-time constraints. The results we obtained are illustrated in Table II, where it is shown the maximum error detection latency for SBE and MBE regions. In detail, the table reports the length of the carry chain detector, the delay latency with routing and logic contributions of the SBE region, as well as the distance from the detector and the delay latency for the MBE region. It is notable that the SBE region latency is larger than for the MBE region because all the carry chains in each CLB that resides in the same column have been connected in a unique CLB column.

Table I. Characteristics of the implemented circuits

	Original		DMR		TMR		Our Approach	
Circuit	LUTs [#]	FFs [#]	LUTs [#]	FFs [#]	LUTs [#]	FFs [#]	LUTs [#]	FFs [#]
B03	39	35	78	69	180	111	85	69
B04	115	67	213	131	449	201	250	131
B05	163	42	326	83	792	135	390	83
B07	98	50	196	99	406	153	18	16
B08	20	21	40	42	89	63	49	42
B09	49	28	98	56	117	84	108	56
B10	37	24	72	47	157	72	80	47
B14	1,161	217	2,322	434	6,417	669	2,552	434
B15	1,900	425	3,480	849	8,358	1,275	4,311	849
Cordic rp	1,024	1,019	2,048	2,038	3,658	3,387	2,329	2,038
Cordic pr	689	690	1,378	1,380	2,259	2,259	1,446	1,380
miniMIPS	3,200	1,883	6,439	3,764	5,649	5,649	6,789	3,764
L80SoC	261	237	522	473	1524	726	609	473
RS Decoder	4191	2801	8178	5600	18452	8403	9056	5620
DCT	1254	1935	2508	3866	5608	4755	2811	3866

The analysis of the obtained data demonstrated that two aspects affect the error detection latency: the routing congestion and the detector location. For the SBE region error detectors the contribution of latency due to logic delay for a fully connected column is around 22.94 ns (almost the 40%) compared to the routing delay contribution, which amounts to 35.53 ns (almost 60%).

Detailing the delay analysis, we note that the logic delay is proportional across a column of slices, since the number of carry chain detectors provides a proportional delay contribution. Vice versa, the routing delay has a higher variability due to the routing congestion induced by the adjacency of slices making the interconnection tiles less available for longer routing tracks. This aspect does not only increase the overall latency but introduces severe constraints on the routing usage in terms of design working frequency. Two alternatives have been used in order to reduce the routing delay time. The former one is oriented to relax the placement constraints of the overall design logic resources, thus leaving some empty slices in the SBE region; this permits placing the carry chain flag interconnections only on a single CLB column and avoiding carry chain flag interconnections convergence across two or more slice columns. The latter solution corresponds to the usage of LUT's configured as OR gates in order to converge flag signals. This introduces a trade-off between the area overhead and the router completion time.

It is worthwhile to mention that while the area overhead increases, the maximum error detection latency decreases. Similarly, the MBE region uses LUT's configured as OR gates for the reduction of the interconnection flags. As a result, the error detection latency for the MBE region has a logic delay contribution of 0.19 ns, while the routing delay is 2.92 ns, which corresponds to almost 6% and 94% of the overall logic delay, respectively. These asymmetrical values are due to the fact that in the MBE region the carry chain detector is one slice long and the error detection latency directly depends on the placement location of the OR-LUTs and their distances from the flag generation node.

Table II. Error Detection Latency for carry chain detectors

Circuit	SBE region				MBE region	
	Length [#]	Latency [ns]	Logic Delay [ns]	Routing Delay [ns]	Distance [#]	Latency [ns]
B03	4	8.81	5.73	3.09	4	1.25
B04	8	19.62	9.77	9.85	8	1.52
B05	14	42.61	16.03	26.58	14	2.07
B07	10	22.87	11.67	11.22	10	2.76
B08	3	6.63	4.63	2	2	1.15
B09	2	3.8	3.47	0.33	14	2.82
B10	4	11.98	4.59	7.39	2	1.023
B14	20	55.81	22.93	32.88	18	3.10
B15	20	51.26	22.10	29.16	19	3.42
Cordic_rp	0	0	0	0	3	1.99
Cordic_pr	0	0	0	0	2	1.85
miniMIPS	20	52.43	22.98	28.46	19	3.99
L80SoC	19	40.50	21.86	18.64	16	2.44
RS decod	20	44.80	22.98	21.82	19	3.86
DCT	13	23.84	14.97	8.86	17	2.99

5.3 Error Correction and Detection

The effectiveness of the proposed approach concerning the error correction and detection capabilities have been evaluated through the execution of a number of fault injection campaigns. The experiments have been performed on the Xilinx Virtex-5 LX110T SRAM-based FPGAs by injecting transient faults into the FPGA's configuration memory and evaluating the circuit's response through the execution of circuit specific workloads. Please note that the faulty bitstreams are generated by corrupting the FPGA's configuration memory bits belonging to the dynamic region, while the static region was kept fault free.

Table III shows the fault injection results, where for each circuit 10,000 Single Event Upsets (SEUs) have been randomly injected into the whole FPGA configuration memory bits related to the reconfigurable region. All the circuits have been emulated at 50 MHz and SEUs are practically injected by downloading the corrupted bitstreams into the FPGA configuration memory.

Table III. Fault injection campaign experimental results

Circuit	SEUs		MEUs	
	Wrong Answer [#]	Corrected [%]	Wrong Answer [#]	Corrected [%]
B03	2,448	98.7	2,944	97.7
B04	584	99.8	632	98.3
B05	782	99.9	942	94.9
B07	4,762	97.4	5,682	95.6
B08	1,425	99.9	1,704	98.4
B09	1,784	99.6	8,938	99.0
B10	1,903	99.4	5,986	99.2
B14	5,121	98.4	5,443	97.9
B15	6,930	99.3	7,240	98.4
Cordic_rp	4,932	99.7	5,230	98.4
Cordic_pr	3,142	98.3	3,350	97.6
miniMIPS	8,903	99.9	9,104	98.4
L80SoC	1,238	97.8	1,469	97.3
RS decod	9,236	99.4	9,491	98.8
DCT	5,232	98.9	5,523	97.3

In details, the *Wrong Answer* reports the number of SEUs and MEUs provoking a wrong answer on the circuit outputs; the *Corrected* column reports the number of SEUs and MEUs properly corrected by our approach. Please note that the MEU effect considered in our experiments always occurs in different slice columns involving the modification of two configuration memory bits.

Table IV. Recovery Time comparison (worst case)

Circuit	DMR [µs]	TMR [µs]	Our Approach [µs]
B03	383.7	1,033.2	119.3
B04	678.9	1,239.8	237.7
B05	1,269.4	3,070.1	238.2
B07	531.4	1,594.1	238.9
B08	88.6	856.1	119.2
B09	206.6	501.8	120.9
B10	501.8	974.2	237.2
B14	2,922.5	5,667.8	829.7
B15	5,077.4	8,796.9	2,010.8
Cordic_rp	3,483.4	4,693.7	120.1
Cordic_pr	1,416.9	4,073.8	119.9
miniMIPS	8,029.4	11,335.7	2,011.4
L80SoC	2,154.9	2,892.9	474.8
RS decoder	11,040.5	17,062.6	2,129.3
DCT	4,250.9	19,128.9	2,364.6

This corresponds to the worst-case scenario; in fact, correction of SEUs and MEUs in a single slice column takes always the same amount of time; vice versa, errors in different columns require a longer computational time to be corrected. The obtained results demonstrate the ef-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2016.2607749, IEEE Transactions on Computers

fectiveness of our approach, which is able to correct more than 98% of the injected errors provoking wrong answers for all the considered circuits.

We also measured the recovery time; in Table IV we reported the worst recovery time measured for all the circuits during the execution of the fault injection campaigns.

Moreover, we measured the average recovery time for the tested circuits, which correspond to 76.3 μsec and 158.9 μsec for SEUs and MEUs, respectively. We also computed the recovery time required by the redundancy approaches, such as TMR and DMR, using active configuration memory scrubbing of all the reconfigurable region area, which is about 1.2 ms; our approach shows an improvement of more than one order of magnitude, and the advantage provided by our approach is extremely large on all the considered circuits.

5.4 Timing Analysis

Finally, we evaluated the impact on the circuit maximal working frequency on all the benchmark circuits comparing our approach with the DMR and TMR redundancy based techniques. In order to elaborate the timing data we used the static timing analysis tool provided by the Xilinx ISE environment. This tool is capable to provide the maximum delay for each net as well as the maximum delay for each logic cone, thus calculating the circuit maximum working frequency.

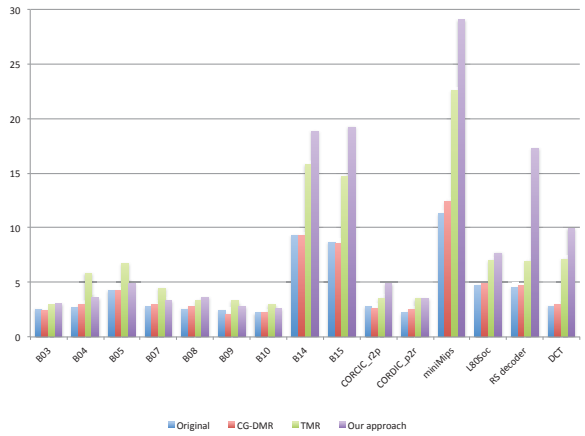

Fig. 7. Clock period comparison for the considered circuits using different error detection and correction approaches.

The results are illustrated in figure 7, which reports the maximum clock period of our benchmark set comparing the figures related to the approaches. As it is possible to notice, our approach has a reasonable behavior for medium complexity circuits, while it is slower (up to the 22% of the nominal working frequency) for larger circuits. This phenomenon is due to the unconventional block placement of logic resources on slice columns for different circuit regions. This aspect affects the timing of the circuit because our technique does not include an optimal floorplan implementation of the different circuit regions. Please note in the reported data, the CG-DMR circuits have been constrained in order to have a justified comparison with respect to our approach. Circuits having less complex routing requirements, such as B03, B09, B14, B15 and CORDIC, may result in a smaller clock period rather then the original ones.

The circuit clock period can be further optimized using floorplan oriented placement algorithms by acting on the Fine-grain logic packing. These logic resources can be packed per slice columns in order to tightly place the fine grain duplication with comparison LUTs resources in the optimal way.

In order to estimate the throughput of the user circuit within the dynamic region when the error detection latency is considered, we evaluated the individual contribution of each phase of the design flow on the timing results of the considered benchmark circuits. In figure 8, we illustrated the obtained results showing the percentage contribution of each design phase constraints on the overall circuit delay: LUT blocks, SBE region, MBE region and Detectors. As it is possible to observe, most of the delay is due to the LUT blocks, thus it is related to the circuit complexity. Considering the contributions of our approach, the majority of the penalization is introduced by the SBE and MBE region constraints.

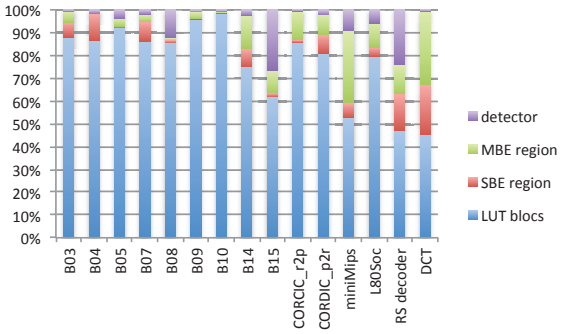

Fig. 8. Percentage of influence of the approach implementation phases on the circuit dynamic region.

6 Conclusions and Future Works

In this paper we propose a fine-grain fault detection and correction mechanism, which can be applied to dynamically reconfigurable systems implemented by FPGA devices. The approach is characterized by the capabilities of detecting and correcting errors induced by single and multiple upsets affecting the FPGA configuration memory. The approach exploits the available carry chains and the hardwired extra logic to perform the error detection; moreover, it is able to recover and correct errors using run-time partial reconfiguration. The effectiveness of our approach has been evaluated with fault injection campaigns demonstrating that our approach is able to detect and correct more than 98% of the bit-flips, showing an improvement of more than 1 order of magnitude in terms of recovery time with respect to traditional redundancy-based approaches. Moreover, our approach has a more limited area overhead than TMR in terms of required circuit resources. As future researches, we plan to improve the approach on two fronts: on one side, we aim at reducing the area overhead by optimizing the error flag propagation in order to be transparent with respect to the reconfiguration process. On the other side, we plan to further reduce the recovery time by optimizing the resource placement and selectively freezing the unused slices during the error correction process.

7 REFERENCES

[1] M. G. Gericota et al., "On-Line Self-Healing of Circuits Implemented on Reconfigurable FPGAs", 13th IEEE International On-Line Testing Symposium (IOLTS 2007), pp. 217-222, 2007

[2] L. Sterpone, M. Violante, "A New Algorithm for the Analysis of the MCUs Sensitiveness of TMR Architectures in SRAM-Based FPGAs", IEEE Trans. on Nuclear Science, Vol. 55, Issue 4, Part 1, pp.2019-2027, 2008

[3] N. Rollins et al., "Evaluating TMR techniques in the presence of single event upsets", . Conf. on Military and Aerospace Programmable Logic Devices (MAPLD), pp. P63–P63, 2003.

[4] M. Niknahad, O. Sander, and J. Becker, "Fgtmr – fine grain redundancy method for reconfigurable architectures under high failure rates," NASNIT 2011.

[5] M. Niknahad, O. Sander, and J. Becker, "A study on fine granular fault tolerance methodologies for FPGAs,", 6th International Workshop on in Reconfigurable Communication-centric Systems-on-Chip, (ReCoSoC), 2011 june 2011, pp. 1 –5.

[6] M. Niknahad, O. Sander, and J. Becker, "Fine Grain Fault Tolerance – A Key to High Reliability for FPGAs in Space", IEEE Aerospace Conference, 2012.

[7] M. Koester et al., "Design optimization for Tiled Partially Reconfigurable Systems", IEEE Transaction on Very Large Scale Integration (VLSI) Systems, Volume:19, Issue 6, pp. 1048-1061, 2010.

[8] M. Sonza Reorda, L. Sterpone, A. Ullah, "An error-detection and self-repairing method for dynamically and partially reconfigurable systems", 18th IEEE European Test Symposium, 2013, pp. 1 – 7

[9] G. L. Nazar and L. Carro, "Exploiting Modified Placement and Hard-wired Rescources to Provide High Reliability in FPGAs," 20th International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 49-152, 2012

[10] X. She and P. Samudrala, "Selective Triple Modular Redundancy for Single Event Upset (SEU) Mitigation," NASA/ESA Conf. on Adaptive Hardware and Systems, 2009 (AHS 2009), pp. 344–350, 2009.

[11] A. Cristo, K. Fisher, A. Gualtieri, R. M. Pérez, P. Martinez, "Optimization of Processor-to-Hardware Module Communications on Spaceborn Hybrid FPGA-based Architectures", IEEE Embedded Systems Letters, Vol. 5, No. 4, December 2013.

[12] M. Gokhale et al., "Dynamic reconfiguration for management of radiation-induced faults in FPGAs", Proc. 18th Intl. Parallel and Distributed Proc. Symp, pp. 145-150, 2004.

[13] F. Kastensmidt et al., "On the Optimal Design of Triple Modular Redundancy Logic for SRAM-Based FPGAs", Proc. Design, Automation and Test in Europe, pp. 1290-1295, 2005.

[14] M. G. Gericota et al., "A Self-Healing Real-Time System Based on Run-Time Self-Reconfiguration," IEEE International Conference on Emerging Technologies and Factory Automation, 4 pp. – 1042, , Sept. 2005

[15] M. G. Gericota et al.,"Robust Configurable System Design with Built-In Self-Healing", Conference on Design of Circuits and Integrated Systems, November, 2005.

[16] V. V. Kumar and J. Lach, "Fine-Grained Self-Healing Hardware for Large-Scale Autonomic Systems", 14th International Workshop on Database and Expert Systems Applications (DEXA'03), pp. 707-712, 2003.

[17] Sandeep K. Venishetti, Ali Akoglu and Rahul Kalra, "Hierarchical Built-in Self-testing and FPGA Based HealingMethodology for System-on-a-Chip", Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS), pp. 717 - 724 , 2007.

[18] M. Straka, Z. Kotasek, and J. Winter, "Digital systems architectures based on on-line checkers," 11th EUROMICRO Conference on Digital System Design, IEEE Computer Society, pp. 81–87, 2008.

[19] M. Straka, J. Kastil, and Z. Kotasek, "Fault Tolerant Structure for SRAM-based FPGA via Partial Dynamic Reconfiguration," 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, pp. 365 - 372, 2010.

[20] B. Pratt, M. Caffrey, J. F. Carroll, P. Graham, K. Morgan, M. Wirthlin, "Fine-Grain SEU Mitigation for FPGAs Using Partial TMR", IEEE Transactions on Nuclear Science, pp. 2274 – 2280, Vol. 55, August 2008.

[21] U. Legat, A. Biasizzo, F. Novak, "SEU Recovery Mechanism for SRAM-based FPGAs", IEEE Transactions on Nuclear Science, Vol. 59, pp. 2562 – 2571, 2012.

[22] Xilinx Product Specification, "Xilinx Virtex-5 Family Overview", DS100, August 2015, pp. 15.

[23] Xilinx Product Specification, "Xilinx Virtex-5 FPGA Data Sheet: DC and Switching Characteristics", DS202, May 5, 2010.

[24] "ITC'99 Benchmarks (2nd release)", 1999. [Online] Available at http://www.cad.polito.it/downloads/tools/itc99.html

[25] "miniMIPS Overview", opencores.org, 2009 [Online]. Available: http://opencores.org/project, minimips.