Superimposed In-Circuit Fault Mitigation for Dynamically Reconfigurable FPGAs

Alexandra Kourfali, David Merodio Codinachs and Dirk Stroobandt

Abstract—Reassuring fault tolerance in computing systems is the most important problem for mission critical space components. With the rise in interest of commercial SRAM-based FPGAs, it is crucial to provide runtime reconfigurable recovery from a failure. In this paper we propose a superimposed virtual coarse-grained reconfigurable architecture, embedded an on-demand three level fault-mitigation technique tailored for FPGA overlays. The proposed method performs run-time recovery via Microscrubbing. This approach can achieve up to 3× faster runtime recovery with 10.2× less resources in FPGA devices, by providing integrated layers of fault mitigation.

Index Terms—FPGA, Parameterised Configuration, Scrubbing, TMR.

I. INTRODUCTION

FIELD Programmable Gate Arrays (FPGAs) have become one of the core components for space computing systems. FPGAs are commonly used on board spacecrafts, such as in Sentinel-2, where 58% of its integrated on board circuits are reprogrammable. SRAM-based FPGAs specifically are a convenient solution, as they can offer hardware reconfiguration in later design or even mission stages. They are best suited for high-performance on board tasks due to their flexibility and their embedded DSP blocks, compared to single-board computers and DSP processors.

There is a growing demand for Commercial Off The Self (COTS) SRAM-based FPGAs for Low-Earth Orbit satellites, for missions with lower lifetime and for missions with less quality constraints due to their high performance and low cost. However, in contrast to their antifuse counterparts, SRAM-based FPGAs are in principle more susceptible to Single Event Upsets (SEUs), as radiation effects can alter the hardware configuration. Lately, SRAM-based FPGA overlays have gained traction among researchers, because of their potentials to reduce development costs in a range of High-Performance Computing applications. An FPGA overlay is a virtual reconfigurable architecture that overlays on top of the physical FPGA configurable fabric and performs useful computations.

Virtualization of FPGA resources has long been an active area of research since the early days of reconfigurable computing. The COTS FPGAs do not undergo radiation hardening procedure and they are not tested for harsh radiation environments, leaving the system susceptible to radiation-induced permanent and non-permanent faults. Hence, the system must integrate built-in mitigation techniques, in system or architecture level, for run-time recovery from failures. The objective of our research is to introduce mitigation techniques in FPGA overlays, to enhance their fault tolerance.

In this paper we propose a superimposed in-circuit reliability-aware infrastructure for overlay architectures, with an integrated fault mitigation mechanism. We also introduce a custom tool flow that integrates an overlay FPGA with fault tolerance techniques, such as TMR, and custom configuration scrubbing.

II. STATE OF THE ART

An FPGA is an array of programmable logic blocks and a configurable routing network. FPGAs implement combinational ans sequential logic by using K-input LUTs. This functionality is highly dependent on the programming bitstream that is shifted into the FPGA’s SRAM configuration cells. The FPGA’s SRAM configuration cells are extremely prone to radiation that introduces hardware faults in the FPGA configuration memory. The hardware faults can often be mitigated via radiation hardening procedures. However, in Commercial Off The Shelf (COTS) FPGAs, there is no radiation hardening. In that case, hardening by design is used.

Mitigation of radiation-induced hardware faults in COTS FPGAs considers both permanent and non-permanent faults, such as SEUs, Single Event Transients, etc. These faults are caused by high-energy subatomic particles from the cosmic radiation. They can cause extensive damages in the functionality of FPGA components such as logic blocks, I/O blocks and DSP blocks. Hence, it is essential to apply fault tolerance for SEU mitigation. The basic approaches include Triple Modular Redundancy (TMR) and configuration scrubbing.

TMR is the most common type of mitigation in SRAM-based FPGA devices for aerospace applications. It involves triplication of circuitry and majority voter insertion. TMR is applied to the netlist of a circuit using automatic insertion. Several commercial and academic software tools are available, such as the Xilinx TMR (XTMR) [2].

Multiple errors can break TMR. Thus, configuration scrubbing is used as an enhancement to TMR to avoid fault accumulation. Scrubbing is a mitigation technique that corrects SEUs in an FPGA’s configuration memory via reconfiguration. Scrubbing protects against accumulation of SEUs, by writing incorrect bits with correct data while the system is fully operational. It utilises a dedicated area of the FPGA and performs the necessary operations to repair SEUs by reconfiguring a portion of the design while continuously scrubbing the entire FPGA without disrupting its operation [4], [5].
The Xilinx 7-Series FPGA provides a number of fault mitigation features, such as a built-in internal configuration scan and Frame ECC. However, results from radiation testing on the 28-nm Kintex 7 FPGA suggest that intra-frame multi-bit upsets (MBU) account for 9.9% of the events observed [13]. That means that only 90.1% of the bits can be repaired with the existing technology, as scrubbing will not protect TMR against a MBU, where a single strike can hit multiple nodes at one time. Additionally, scrubbing needs re-synchronization of the system, which introduces time overhead. Significant time overhead is also introduced by reconfiguration during scrubbing, which may jeopardize the entire design [1].

An FPGA overlay is a virtual reconfigurable architecture that overlays on top of the physical FPGA configurable fabric and can carry out certain computations. An FPGA Overlay may be designed as a virtual FPGA, a processor, a GPU, or as a (virtual) coarse-grained reconfigurable array (CGRA).

Virtual CGRAs (VCGRAs) are FPGA overlays that can bridge the gap between FPGA implementations and high level application descriptions, as with VCGRAs the time consuming design cycle of the FPGA (synthesis, mapping, place and route) can be moved forward to a pre-compile time step. This is achieved mainly because VCGRAs allow the designer to write the code in a higher abstraction level language, without requiring knowledge of the underlying hardware. Hence, the entire development cycle is reduced, as (re)compilation is avoided for the (re)construction of the VCGRA. VCGRAs have been proposed before, either as optimised architectures, as a solution for long compilation times, or as a facilitator for high level synthesis [3], [11].

VCGRAs consist of a large number of processing elements (PEs), laid out in a grid pattern, and Virtual Channels (VCs), that are a communication network connecting the PEs. Each PE is a coarse grained element and is capable of computing an incoming data and pass on to the next depending on the connection defined by the VC located adjacent to the PE. The VCGRA is implemented using reconfigurable connections, with the parameterised configurations flow [8]. Parameterized configurations for VCGRAs is a methodology used for implementing an application whose input values (called parameters) change infrequently, on an FPGA. The VCGRA needs to be changed/adapted infrequently. Instead of implementing the VCGRA’s inputs as regular inputs, in the parameterized configuration approach these inputs are implemented as constants and the FPGA overlay is optimized for these constants. When these inputs change, the design is re-optimized for the new constant values by reconfiguring the FPGA.

There are currently no dedicated techniques that introduce SEU mitigation for overlay FPGA architectures. The designers can so far employ an existing TMR tool such as XTMR and Frame ECC and then run the conventional design flow, which introduces significant resource overhead (\(\geq 3 \times\)) and reduces the overall performance. Moreover, these tools do not offer any protection against MBUs and fault accumulation.

Alternatively, we propose a technique that integrates a VCGRA with fault mitigation schemes, during the FPGA’s later design and mission stages. In that way less hardware overhead and performance degradation is observed, caused by the integration of the added logic (e.g. majority voters) in the VCGRA and the elimination of the need of (re)configuration and recompilation of the target FPGA. Hence, less LUTs and interconnection FPGA resources are needed and the reliability is improved. The remainder of this paper is organised as follows: In Section III we introduce the proposed technique, that describes the hardening by design technique. This section describes in detail how TMR and scrubbing can be introduced in an overlay design. Section IV describes the tool flow that supports the mitigation functionality. Then, Section V describes a case study and the subsequent steps taken to provide SEU mitigation in a target application.

III. MULTIPLE-LEVEL FAULT MITIGATION

The main contribution of this paper is an on-the-fly adaptable fault mitigation scheme for aerospace applications that use COTS SRAM-based FPGAs, integrated with an overlay architecture. The proposed tool creates an online self-healing solution of the virtual architecture by constructing automatically a tailor-made TMR scheme for the target VCGRA application, that is automatically installed every time the overlay changes. Then, during online time (later design or mission stage), the FPGA can be rapidly repaired from SEUs and MBUs on-the-fly via targeted microscrubbing. A separate step subsequently translates this overlay architecture, together with the application that runs on it and its mitigation scheme, to the physical FPGA. This section describes how the VCGRA can be integrated with SEU mitigation schemes and how it can be adapted and repaired on-the-fly.

In order to build the VCGRA, the first step is to design the PE and VC. Then, the VCGRA grid is constructed. Instead of building the VCGRA grid from scratch, we use a tool presented at [9] that automatically creates the VCGRA’s top-level VHDL description, from a description of the hardware structure. The grid’s structure is described by the number of PEs in each level of the architecture and the elements’ input and output bandwidths. The tool’s output is HDL code that describes the grid.

A. First level: Triple Modular Redundancy for VCGRAs

In aerospace applications it is crucial to be able to have a detect mechanism that can assess if a SEU occurs in this
After the construction of the VCGRA at the target FPGA, SEU mitigation can be directly applied. Then, after the VCGRA is changed TMR can be adapted without FPGA interruption. TMR is applied on the overlay infrastructure at a PE level and not at a gate level as in the state of the art. That creates an overlay TMR infrastructure on top of the VCGRA architecture shown in the Superimposed level in Fig. 1. In that way the TMR can be adapted on-the-fly, as soon as the VCGRA changes. Moreover, since it is performed on a higher abstraction level, not all TMR resources are translated directly at real FPGA resources, reducing the resource overhead.

1) TMR: Since the VCGRA is constructed with multiple layers of PEs/VCs, TMR is also applied after each layer of PE/VC, as it is shown in Fig. 2(b). One voter is placed directly at the output of the triplicated PE. The voted output serves as an input for the next level (and not the PE’s output). Hence, the VCGRA is redundant on PE/VC level, as each element is triplicated and voted. Thus, SEU effect can be eliminated via redundancy before it can be propagated to another layer of PE/VC. The VCGRA is constructed in such way, that there are clear levels of PEs and VCs. We leverage this architecture and we insert the mitigation scheme on each individual VCGRA level. In that way we try to simultaneously increase the design’s robustness and eliminate fault propagation.

2) Voters: The voters are a standard mitigation technique. They provide a majority voting functionality that ensures correctness at least at the two out of three copies of the system, in order to remain operational. Thus the majority voter plays a pivotal role in ensuring the correct operation of the system. We extend this logic to provide a repair functionality alongside TMR. In that way we can avoid fault accumulation. A majority voter is installed after each triplicated level of PEs/VCs. Each voter is connected with a circuitry that triggers configuration scrubbing on a frame level. Thus, when the TMR result is different than the original target PE or VC, the voters can enable on-the-fly repair of the current level. In that way, the SEU can be detected and repaired on each level, eliminating the need to scrub the entire VCGRA (or FPGA) periodically. This is visualised in Fig. 2(c).

Algorithm 1 3-level Fault Mitigation

1: procedure SICTA(PE, VC) // SEU detection
2: PE ← {add, sub, mul, eq}
3: VC ← {PE[i...i+3]}
4: for i ≤ grid_length do
5: TMRRPEi ← PEi_a, PEi_b, PEi_c
6: Vtmri ← (PEi_a ∧ PEi_b) ∨ (PEi_a ∧ PEi_c) + (PEi_b ∧ PEi_c)
7: end for
8: while VCGRAen do
9: if PEi ≠ Vtmri, then
10: PEi ← PEscrubbed // SEU → cnt++
11: else
12: continue
13: end if
14: TMRVCE ← VC ← Vtmri[i...i+3]
15: switch MS_en do
16: case cnt = threshold ∧ SEUaccm. ∨ MBU
17: microscrub VCGRA
18: case VCGRA change
19: microreconfigure FPGA // re-apply TMR
20: end while
21: end procedure

B. Second level: Configuration Scrubbing

After the installation of the superimposed TMR, the VCGRA still needs to be scrubbed, to avoid SEUs, MBUs and fault accumulation. Since we propose an online self-healing solution of the virtual architecture, the system will not perform full scrubbing periodically to heal the VCGRA from SEUs, instead it will perform location-targeted Microscrubbing (MS).

1) Microscrubbing: After a subgrid is constructed, during the design stage, the fault mitigation scheme is integrated alongside the VCGRA (TMR and majority voters). Then, during the FPGA’s mission stage, if a voter’s output is different than the current output, a SEU has been detected at a specific PE or VC. The majority voters protect the VCGRA from SEUs, however they can not handle Multiple Bit Upsets (MBUs).

Microscrubbing (MS) repairs periodically the VCGRA’s bits, to deal with MBUs and fault accumulation, that are 10% of the total faults [13]. MS is enabled by the processor after detection of multiple SEUs. Subsequently, MS scrubs only the targeted frames (lowest configuration granularity) under SEU using golden bits (stored in a non-volatile memory). This process is triggered automatically without any user interference. It is performed in order to scrub the MBU infected
areas and reconfigures the targeted frames. Therefore, if SEUs are detected, we are able to reconfigure the VCGRA frames instead of the whole FPGA. This is described in Algorithm 1.

This process is completed in three steps that include reading the VCGRA’s frames from the configuration memory, replacing the current truth table entries of a PE/VC with the specialized bits (from a copy stored on a non-volatile memory) and writing back the modified frames to the configuration memory. MS can be also used to target specifically the FPGA’s essential or critical bits. A bit is essential if it is used by the VCGRA application, and critical if it affects a resource of the circuit and the effect can be propagated to the output. However, this option is not investigated here, where MS operates on a higher description level and scrubs VCGRA elements and not specific essential/critical bits.

C. Third level: Parameterised scrubbing

Parameterised scrubbing is built on top of the parameterised FPGA configuration technique. This is enabled in the case the FPGA’s overlay architecture needs to be adapted, or a critical error has occurred. When this trigger is enabled, a parameterised reconfiguration is initiated. Then, the proposed tool reconfigures the entire VCGRA, evaluates the new VCGRA and integrates it with the adjacent TMR infrastructure. This is described in Algorithm 1.

IV. SUPPORTING TOOL FLOW

In this work we use low-level FPGA resources, (physical switch blocks and configuration memory) to create an adjacent-to-VCGRA mitigation scheme. Hence, with every different FPGA configuration scheme that describes a different application implemented on the VCGRA, the TMR can adjust accordingly. A supporting tool-flow that implements a fully parameterised VCGRA integrated with the superimposed architecture is shown in Fig. 3. This tool has been designed in order to allow the designer to create the parameterised infrastructure, add/adjust the VCGRA and incrementally add the mitigation scheme. The tool flow that implements a fully parameterised VCGRA integrated with the superimposed in-circuit fault tolerant architecture is shown in Figure 3.

A. Offline VCGRA Setup

On the left hand side at Fig. 3(a), the VCGRA is implemented using parameterised configuration tool flow The application is parameterized when some of its inputs (parameters), are infrequently changing compared to the other inputs. The parameterised inputs are implemented as constants and the design is optimized for these constants. When the parameter values change, the design is re-optimized for the new constant values by reconfiguring the FPGA. Here, the VCGRA’s parameter inputs are mapped in virtual LUTs that allow their lookup entries to be defined as Boolean functions of the parameter inputs instead of static ones and zeros. The truth table entries (boolean values) will be microreconfigured upon every change in the VCGRA. This is completed in Fig. 3(a)). This part of the flow runs once, creating a parameterised configuration. This configuration is a virtual intermediate layer and describes the target application. After the design is synthesized, it is mapped with a custom technology mapping tool, that maps the Boolean network to virtual primitives (called TLUTs and TCONs) [6]. In this step the granularity and the functionality of the VCGRA is defined, alongside all the possible ways the PEs and VCs can be interconnected.

Then, during placement, LUTs are chosen to implement every instance of the LUT (and TLUT) primitives of the mapped netlist, while during the subsequent routing step resources of the routing fabric are chosen to implement the nets (and TCONs). Finally, in the routing step, the tool finds the optimal routing resources taking into account that not every TCON will have to be concurrently realised [12]. Placement and routing for commercial FPGAs is not yet possible using these tools. Experiments were done for a virtual architecture taken from VPR [10]. Then, in the specialisation stage, the Boolean functions in the parameterised configuration are evaluated to create a configuration bitstream of the VCGRA that can be loaded onto an FPGA.

B. VCGRA toolflow

As soon as the architecture of the VCGRA is constructed, the textual settings are extracted from the VCGRA (Fig. 3(b)). Determining the settings of the configurable components of the VCGRA so that a target application is implemented is generally done using a tool flow associated with the specific VCGRA architecture that is used. The specialised FPGA configuration bitstream for the VCGRA settings is determined using the parameterised configurations tool flow and creates a virtual intermediate level, called an FPGA overlay. By having an FPGA overlay that describes the VCGRA, there is no need to reconfigure the FPGA, upon every change on the VCGRA. Hence if a SEU is detected and repaired (or when the VCGRA application design changes), it is faster to generate the new settings values, compared to processing the new design with the standard FPGA flow. Moreover, the utilization of the LUTs is reduced, leaving more flexibility for the installation of the TMR and voting infrastructure for the VCs and PEs.
C. Offline SEU Mitigation Setup

As soon as the architecture of the VCGRA is constructed, the TMR is integrated on top of the VCGRA, creating a two-level FPGA overlay. In order to setup the SEU mitigation scheme, we need to extract the VCGRA settings. The settings needed are the number of PE levels are used in the application and the number of PEs that define each level. Then, the tool triplicates each VCGRA level. This is done by creating two identical versions of each PE on each level. The tool gets as an input the number of levels and the VHDL description of a PE and extracts a VHDL output where each level of PEs is triplicated and connected with a majority voter. The output of the majority voter is the input of the next level of the VCGRA. The voted output is the input of the VC. Subsequently, the VC itself is also triplicated and its output is connected with a majority voter. This process repeats numerous times, until all levels are triplicated. This is depicted in Fig. 3(c). At this point the granularity the functionality and the mitigation scheme of the VCGRA is defined, alongside all the possible ways the triplicated and voted PEs and VCs can be interconnected. Then, the PEs are mapped into virtual PEs of the VCGRA. Next, with the custom router, optimal connections are created between the PEs VCs voted outputs.

D. Online SEU Mitigation

By using the proposed flow the two-level VCGRA gives a new set of triplicated components. The voter examines the new VCGRA grid, until the voter’s output doesn’t match with the original. When it doesn’t pass, the new settings for the components are merged with the Specialization Stage. This results into a new specialised configuration automatically. This new reconfiguration contains the repaired components. In case a SEU is detected, the process is interrupted immediately in order to perform MS in the VCGRA. This is shown in the lower part of Fig. 3(d).

V. EXPERIMENTS

For the experiments we set up a VCGRA grid used for a HPC image processing (parameterised) application, as our first virtual level overlay, and integrate with SEU mitigation components, such as TMR and configuration scrubbing.

A. First Virtual Level: VCGRA

The components constructing the VCGRA grid has been implemented using the parameterized configuration tool flow [7].

1) Processing Element: The Processing Element has been designed in VHDL and compared as a conventional and a parameterized application. The logic resources (in terms of LUTs) can be optimized up to 24% and the wire length is decreased up to 25% for a parameterised application, with a reconfiguration time costs from 3.4\text{ms} up to 88.5\text{ms}.

2) Virtual Channel: A major part of the virtual channel doesn’t need LUTs to make them reconfigurable, due to parameterization. Hence, with the parameterised configuration, up to 82% of its logic is mapped on the reconfigurable physical switches instead of physical LUTs and multiplexers (as per the conventional implementation). The wire length is decreased by 76%, due to the fact that the minimum channel width is reduced by 42%. This optimization can be achieved at the cost of a reconfiguration time of 4.6\text{ms}.

B. Second Virtual Level: SICTA

1) TMR for a PE: The components that add fault mitigation were synthesised placed and routed, with the proposed CAD tool. Since we virtualise the connections between VCGRA and the TMR infrastructure, SICTA operates on PE level and consumes 10.2x less additional LUTs than non-parameterised TMR. Since overlay architectures don’t always translate in real FPGA resources, installing the proposed architecture has less impact on physical LUTs for PEs, VCs and the complete grid.

2) TMR for a VCGRA grid: To create a fault tolerant VCGRA grid we apply TMR on each layer of PE/VC. Thus, each PE or VC is triplicated and voted, as it is shown in Fig. 2(b). In that way, the fault cannot be propagated in the next level of PEs as its input is always a voted output of the previous level. TMR is always optimised alongside the VCGRA, reducing the actual FPGA resources. Moreover, since it is also integrated with the use of Parameterised Configurations tool, we notice an overall reduction of up to 90% of the total TMR resources compared to the conventional tools.

3) Configuration Scrubbing: MS targets the lowest granularity of configuration for 7-series FPGAs, which is the configuration frame.\footnote{For the 7-series FPGA, each frame is 101 words of 32-bits each (3,232 bits per frame)} In a conventional implementation, the HWICAP is used as a reconfiguration controller. However, MS is realized using a custom reconfiguration controller. For the HPC target application, the reconfiguration speed is improved by 3x over the HWICAP. MS can periodically reload the reconfiguration bits from a golden instance. Since there is no need for VCGRA reconfiguration, only Boolean function evaluation [6], there is a significant decrease in time (5 orders of magnitude less generation time). Finally, in the case the VCGRA needs to be adapted, a full parameterised configuration is needed. This is 3 orders of magnitude faster compared to a full conventional reconfiguration.

VI. CONCLUSION

We have presented a superimposed in-circuit fault-tolerant scheme for future low-cost space missions. We have proposed a technique that applies TMR and microscrubbing at the FPGA’s overlay. This work provides 10x faster scrubbing with 10.2x less FPGA resources and it aims at building an integrated fault mitigation scheme that enhances the reliability of COTS FPGAs.

ACKNOWLEDGMENT

The first author is sponsored by a Ph.D. grant of the Flemish Fund for Scientific Research (FWO). This work was supported by the European Commission in the context of the H2020 FETHPC EXTRA project (#671653).
REFERENCES


