Low-Overhead Fault-Tolerance Technique for a Dynamically Reconfigurable Softcore Processor

Hung-Manh Pham, *Student Member*, *IEEE*, Sébastien Pillement, *Member*, *IEEE*, and Stanisław J. Piestrak, *Member*, *IEEE*

Abstract—In this paper, we propose a new approach to implement a reliable softcore processor on SRAM-based FPGAs, which can mitigate radiation-induced temporary faults (single-event upsets (SEUs)) at moderate cost. A new Enhanced Lockstep scheme built using a pair of MicroBlaze cores is proposed and implemented on Xilinx Virtex-5 FPGA. Unlike the basic lockstep scheme, ours allows to detect and eliminate its internal temporary configuration upsets without interrupting normal functioning. Faults are detected and eliminated using a Configuration Engine built on the basis of the PicoBlaze core which, to avoid a single point of failure, is implemented as fault-tolerant using triple modular redundancy (TMR). A softcore processor can recover from configuration upsets through partial reconfiguration combined with roll-forward recovery. SEUs affecting logic which are significantly less likely than those affecting configuration are handled by checkpointing and rollback. Finally, to handle permanent faults, the tiling technique is also proposed. The new Enhanced Lockstep scheme requires significantly shorter error recovery time compared to conventional lockstep scheme and uses significantly smaller number of slices compared to known TMR-based design (although at the cost of longer error recovery time). The efficiency of the proposed approach was validated through fault injection experiments.

Index Terms—Error recovery, fault injection, fault-tolerance, FPGA, lockstep, reconfigurable system, single-event upset (SEU), softcore processor

1 Introduction

Modern FPGAs, besides customary reconfigurable resources, offer the designers the possibilities of implementing programmable processors having features of Commercial Off-The-Shelf (COTS) components (no need to modify processor architecture or application software). In particular, Xilinx FPGA devices include two categories of processors: the hardcore embedded processor (PowerPC) and softcore processors (MicroBlaze, PicoBlaze) [1]. Hardcore-embedded processors are hardwired on the FPGA die and their number is limited on each device (1, 2, 4, or no hardcore processor). On the other hand, softcore processors use reconfigurable resources, so their number that can be actually implemented depends on the device size only.

Xilinx FPGAs use SRAM-based technologies which are known to be very susceptible to radiation and electromagnetic noise [2]. The major effects caused by them are known as *Single-Event Upsets (SEUs)* or *soft errors*, because only some logic state(s) of memory element(s) are changed but the circuit/device itself is not permanently damaged. In FPGAs, SEUs may directly corrupt computation results or induce changes to configuration memory; the latter can cause changes in the functionality and performance of the device. Due to their flexibility, FPGAs are attractive for mission-critical embedded applications, but their reliability could be insufficient unless some fault-tolerance techniques capable of mitigating soft errors are used. These techniques should allow for online error detection or/and correction during system operation, very fast fault location, quick recovery from temporary failures, and fast permanent fault repair through reconfiguration.

Here, we are interested in designing and implementing a fault-tolerant (FT) softcore processor using Virtex-5 FPGA. FPGA implementations of various fault-tolerant softcore and hardcore processors can be found in [3], [4], [5], [6], [7], [8], [9], [10]. In [3], lockstep systems using dual hardcore processors PowerPC found in certain FPGAs are constructed. However, a limited number of available PowerPC processors in FPGAs restrict their utilization to build, e.g., a fault-tolerant Multiprocessor System-on-Chip (MPSoC). Therefore, one option to implement larger number of lockstep modules in a single FPGA is to employ softcore processors that can use all available reconfigurable resources of the device.

In [4], the authors claim proposing a lockstep scheme using two hardcore PowerPC processors embedded in Xilinx Virtex II Pro FPGA, which could be considered for application in softcore processors as well. However, this scheme is rather a handshake scheme than a lockstep one because, to perform consistency checks, the processors execute the same program but not simultaneously. As a result, the overall time overhead is relatively large (besides context saving and restoring, if needed), because of the sequential execution of two identical tasks on two different processors, which might be prohibitively long in some real-time applications. Moreover, using checkpointing and rollback for context recovery results in an extra time

- *H.-M. Pham is with VNPT Technology JSC, 124 Hoang Quoc Viet, Cau Giay, Hanoi, Vietnam. E-mail: manph@vnpt-technology.vn.*
- *S. Pillement is with École Polytechnique de l' Université de Nantes, Département Électronique et Technologies Numériques, 44306 Nantes, France. E-mail: Sebastien.Pillement@univ-nantes.fr.*
- *S.J. Piestrak is with Institut Jean Lamour, UMR CNRS 7198, Université de Lorraine, 54506 Vandoeuvre-Lès-Nancy, France. E-mail: stanislaw.piestrak@univ-lorraine.fr.*

Manuscript received 8 Apr. 2011; revised 4 Jan. 2012; accepted 7 Feb. 2012; published online 21 Feb. 2012.
Recommended for acceptance by R. Marculescu.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2011-04-0231.
Digital Object Identifier no. 10.1109/TC.2012.55.

overhead. Hence, it is worthwhile to consider the possibility of context saving and restoring only in case of errors, even at the cost of extra hardware.

The fault-tolerant softcore processor architectures from [5] using triple modular redundancy (TMR) are able to correct single errors in a faulty module through roll-forward error recovery. Although there is no need to regularly save the context when no errors occur, the TMR approach consumes well over 200 percent of extra hardware resources which could limit the possibilities of building a more powerful system like MPSoC.

A fault-tolerant version of an open-source LEON3 processor [6], called LEON3-FT, was proposed in [7]. It is provided with means allowing for detection and correction of errors caused by SEUs in the processing unit and all on-chip RAM memories. However, the FPGA implementation of LEON3-FT is highly resource consuming, because all reconfigurable logic like Configurable Logic Blocks (CLBs), Lookup Tables (LUTs), and flip flops, must be protected using TMR.

The solutions from [8], [9], [10] have the ability to detect and correct errors using low-level approaches capable of detecting bit-flips in the configuration memory, but they are unable to distinguish whether the detected bit-flips really affect the design. Very often, a flipped bit does not belong to any design, so that halting the system to correct this error is pointless.

Finally, the cost of full TMR can be excessive in some applications like space systems, in which power consumption and weight are limited. In particular, the available internal FPGA resources like CLBs, LUTs, and Input/Output Blocks (IOBs) could be insufficient to implement a full TMR version (in some designs, at least inputs must be separated to avoid a single point of failure). Moreover that sample designs show that full TMR versions could require significantly more overhead than 200 percent (e.g., in [11], 430 percent for slices and 320 percent for LUTs). In those cases, alternative fault-tolerance techniques that can provide substantial cost reduction, although at the expense of some increase of the failure rate, could be of some interest. Several such techniques which rely either on partial TMR applied only to the most critical parts of the design and/or are application-dependent have been proposed, e.g., in [11], [12], [13], [14].

In this paper, we propose a new architecture of a fault-tolerant reconfigurable system which can be implemented at a reduced hardware and time cost on any SRAM-based FPGA with integrated softcore processors. Our actual implementation on Xilinx Virtex-5 FPGA relies on using an Enhanced Lockstep scheme built using a pair of MicroBlaze cores. To identify the faulty core, we propose a specially designed fault-tolerant *Configuration Engine* (CE) built using PicoBlaze. Once the exact error location is determined by a specially designed *Scan Motor* and a *Frame Address (FA) Generator*, the error is corrected through partial reconfiguration (PR) combined with roll-forward recovery technique.

The paper is organized as follows: Section 2 details some basic FPGA features that are essential to support fault tolerance in the designs proposed here, presents the fault and error model of SRAM FPGAs, and surveys the fault-tolerance techniques commonly used in FPGA-based systems. Section 3 details the new fault-tolerant architecture.

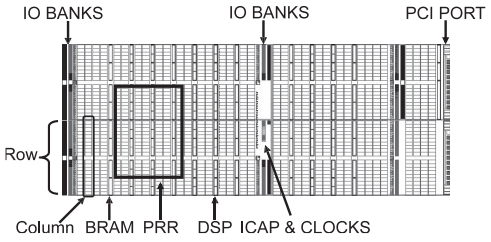

Fig. 1. Dynamically reconfigurable architecture of Virtex-5 FPGA.

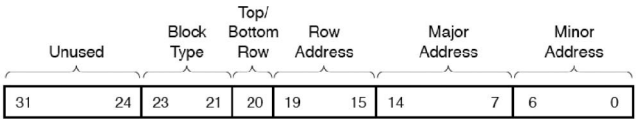

Fig. 2. Division of 32-bit frame address ([15, Figs. 6, 7, 8, 9, and 10]).

The fault mitigation strategy and the state recovery procedure are described in Sections 4 and 5, respectively. FPGA implementation details and comparisons against existing designs are presented in Section 6. Section 7 presents fault injection campaigns which provided statistical data to validate our design. The last section presents some conclusions and gives some perspectives for further research.

2 Preliminaries

In this section, we will present some basic FPGA features that are essential to support fault tolerance in the designs proposed here, the fault and error model of SRAM FPGAs, and the survey of the fault-tolerance techniques commonly used in FPGA-based systems.

2.1 Partial Reconfiguration of FPGA

Currently, the main interest in the granularity of the FPGA programming data is related to the dynamic reconfiguration property provided by some recent FPGAs to perform online programming (dynamic reconfiguration) of a portion of their logic (partial reconfiguration) without affecting the rest of the system.

Fig. 1 shows the internal structure of the Xilinx Virtex-5 FPGA. Virtex FPGAs have configuration memory arranged in frames that are tiled across the device. Frames are the smallest addressable segments of the device configuration memory space. The FPGA fabric is physically divided into different rows, with each row divided into columns. The rows are numbered from 0 (up to 9) in the top and bottom halves of the FPGA, starting from the center (see [15, Figs. 6, 7, 8, 9, 10, and 11]). A column in the device matrix corresponds to a block in the array (CLB, DSP, block RAM (BRAM), IOB, etc.). In Virtex-5, a CLB column is 20 CLBs high by 1 CLB wide ([15, Ch. 6]) and one CLB contains two slices, whereas one BRAM column contains four 4 KB BRAM blocks. The configuration of a column is defined by a set of certain configuration frames whose unique 32-bit address can be determined using five parameters detailed in Fig. 2, stored in the Frame Address Register (FAR). The major addresses are numbered from left to right (physically in the device matrix) starting with 0, and they, respectively, correspond to the positions of the columns inside a row. The addresses with the

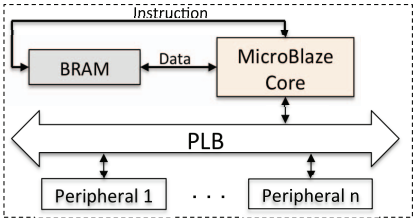

Fig. 3. MicroBlaze softcore processor structure.

same major address and different minor addresses belong to the frames determining the configuration of one column. The numbers of minor addresses inside a column depend on logic type (CLB, DSP, or BRAM interconnect) [15, Table 6-16].

Frame-based reconfiguration allows to modify a frame—the smallest reconfigurable entity, whereas a *modular* or *module-based reconfigurable system* is one which has at least one entire block dynamically reconfigurable [16]. In a modular reconfigurable system, the FPGA fabric consists of some static logic and one or more partially reconfigurable regions (PRRs). This fabric partitioning enables reconfiguration of a single PRR without system interruption (the static region and other PRRs continue execution while only the reconfigured PRR halts) [17].

The behavior of the FPGA is determined by configuration bitstreams that consist of a sequence of *instructions* and *control signals* data. Downloading this sequence allows to program the FPGA to perform requested design functions. A nonreciprocal relation exists between the design and its bitstream: it is not possible to extract the design structure and implementation on the FPGA from the bitstream. The reconfiguration process itself can be done either completely or partially by sending the related bitstream (full or partial) to the Internal Configuration Access Port (ICAP) ([15, Ch. 4]).

2.2 MicroBlaze Softcore Processor Structure

Besides hardcore embedded processors like PowerPC, that are hardwired in the die, FPGAs can also implement softcore processors offered by manufacturers, like MicroBlaze, PicoBlaze, and Nios, as well as those proposed by the open-source community, like LEON3 [6] and LEON4 [18]. Softcore processors are Intellectual Property (IP) blocks written in hardware description languages (HDL) like VHDL or Verilog, to be implemented using reconfigurable resources of FPGAs. Fig. 3 shows a typical MicroBlaze processor system that consists of a 32-bit MicroBlaze core, the program memory BRAM with its data and instruction buses, and the Processor Local Bus (PLB) [19]—the central bus of the MicroBlaze core with some peripherals connected to it.

2.3 Fault and Error Model of SRAM FPGAs

We assume a commonly used fault model of SRAM FPGAs, which includes temporary faults (SEUs) affecting configuration memory or user memory implementing, e.g., flip flops and registers. Single SEUs will be of our primary concern here, because they are the most likely to occur. Nevertheless, unlike most authors, we will also consider permanent faults affecting configuration memory (in Section 6.1).

2.3.1 Configuration Upsets

SEUs in configuration memory may result in modifications of the functionalities of the application design the FPGA implements. For a given design, all configuration memory bits can be classified as being *sensitive* (whose upset induces errors) and *non-sensitive* [20]. This is because among numerous configuration memory bits only some are actually utilized by the user's design, hence SEUs affecting the configuration bits that are not utilized by a specific design will not affect the behavior of that design. Although of temporary nature, SEUs may have permanent effects until the device is reconfigured, e.g., by readback or scrubbing [21]. In addition to configuration sensitivity, the sensitive bits can be further categorized into two following categories [20]:

- *Nonpersistent bits* are those configuration bits which, when upset, may induce nonpersistent functional errors which disappear once the device is reconfigured, so that the design can return to normal operation. The nonpersistent bits generally involve purely combinational circuitry of the design.
- *Persistent bits* are those configuration bits which, when upset, induce persistent functional errors, which do not disappear even after the device is reconfigured. The persistent bits generally involve any part of the design that contains the sequential circuitry or BRAM. The frame-based reconfiguration followed by the internal reset could eliminate persistent errors but, unfortunately, it cannot deal with corrupted data written to the BRAM. One feasible solution for the latter problem is the following:
  - to reconfigure the whole module (module-based reconfiguration) to eliminate configuration errors and reestablish the startup state of the BRAM; then
  - to perform an internal reset of the registers and flip flops to the initial correct state; and finally,
  - to perform the context recovery.

2.3.2 User-Logic Upsets

The user logic contains memory elements (registers and flip flops) which change their states during normal operation. Obviously, they can also be affected by SEUs which could provoke the so-called *logic errors* due to bit flips and which may have permanent effects until the affected memory element is rewritten. Because their contents is not readily available for inspection, any concurrent error detection is virtually impossible without using fault-tolerance techniques specifically dedicated to handle logic errors (like TMR). Nevertheless, the probability of undetected logic errors seems relatively limited by taking into account the following observation. In modern FPGA devices, the configuration cells occupy more than 98 percent of all memory elements: e.g., Xilinx Virtex-5 SX50T FPGA contains a total number of 20,019,328 configuration bits and only 28,800 flip flops [15], [22]. The occurrence probability of logic upsets three orders of magnitude lower than of configuration upsets is confirmed by the experimental results reported in [23, Figs. 6 and 7]. Moreover, it was reported in [24, Fig. 11] that frequency variation has no evident impact on the number of observed upsets.

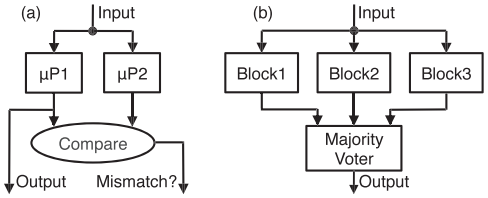

Fig. 4. Hardware redundancy. (a) Duplication with comparison—basic lockstep scheme. (b) Triple modular redundancy.

2.4 Standard Configuration Data Error Handling Techniques in FPGAs

Configuration data of SRAM FPGAs, containing millions of configuration bits, are particularly vulnerable to SEUs. They can be protected against errors by various standard means provided by FPGAs' manufacturers, like configuration readback, configuration scrubbing, and error detecting or correcting codes [8], [21]. These techniques can be applied continuously in the background of a user design. FRAME_ECC is the Virtex-5 and -6 primitive using single-error-correcting/double-error-detecting (SEC/DED) Hamming error correcting code (ECC) which during configuration readback allows to correct single errors and/or detect double errors in the configuration frame data ([15, Ch. 4]).

2.5 Hardware Redundancy Techniques for FPGAs

The most widely used fault-tolerant methods used to mitigate logic errors in FPGA designs rely on hardware redundancy: 1) duplication with comparison (DWC) for detecting faults (Fig. 4a) and 2) triple modular redundancy (TMR) with majority voter for masking faults (Fig. 4b).

In DWC, the original block to be protected is replicated twice and the results produced by the original and the outputs of replicated blocks are compared to detect faults. The lockstep scheme of Fig. 4a is the implementation of DWC at the processor level, supported by some Xilinx FPGAs [3]. Two identical processors μP1 and μP2 receive the same inputs, simultaneously execute the same instructions, and their results are compared step-by-step at each clock cycle. μP2 generates the reference results to be compared against those of μP1 that provides the system output. Basically, DWC is able to detect but not to correct errors, because it cannot point out the faulty processor. However, it could be capable to tolerate temporary faults, provided that it is supported by some reexecution procedure. In case of FPGA implementation, the system needs also to be reconfigured to recover correct functionalities.

In TMR, the original block is replicated thrice, all three blocks receive the same inputs, and a majority 2-out-of-3 voter is used to determine the correct result. It is also possible to detect inconsistencies in the outputs of the three blocks to identify the faulty one. TMR allows to mask directly both temporary and permanent faults of a single block.

DWC and TMR are conceptually relatively simple and easy to implement in FPGA. Unfortunately, they are also very costly, because they involve, respectively, over 100 and 200 percent hardware overhead. Therefore, they must be used skillfully to guarantee required reliability level at the minimum cost.

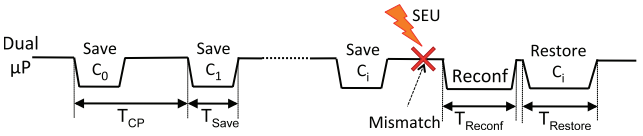

Fig. 5. Checkpointing and rollback for basic lockstep scheme recovery.

2.6 Error Recovery Techniques

Once an error is detected, the next step is error recovery, i.e., the process of removing errors from the system and bringing it back to the error-free state. The *processor context* is the set of information needed to define uniquely the state of the processor at a given moment. It could include the states of the processor registers, the cache, the memory, etc. Saving and restoring all relevant values is essential for effective processor context switching and error recovery.

Several error recovery schemes have been proposed [25], [26], [27] which can be classified as backward and forward. The backward error recovery techniques, which are based on time redundancy, rely on saving the correct processor context and restoring it once the errors are removed, so that the processor could resume correct functioning at the last saved point (checkpoint). Among various context recovery techniques differing in the moment of saving and restoring the context, the most often used is rollback recovery using checkpoints. Checkpointing is the set of operations needed to retrieve and store the system's context, whereas rollback is the operation that returns the processor state saved during last checkpoint before the error occurrence. The major drawback of rollback recovery using checkpoints is the time overhead (regular saving of the processor context with checkpointing frequency growing with the error rate, computations lost from the last checkpoint followed by restoring of the processor context). Fig. 5 shows the scheme of checkpointing and rollback applied for basic lockstep scheme recovery. Because it is not possible to identify the faulty processor without extra diagnosing support, the error recovery in FPGA designs can be achieved through reconfiguration of both processor cores which, however, can be time consuming.

The latter problems are alleviated in roll-forward error recovery which does not need regular context saving. When an error occurs, it is corrected by copying the correct state of the processor from a fault-free mirror processor, provided that there are means to identify it. Because a lockstep scheme using roll forward rather than rollback seems to offer better performance without significant increase of hardware resources, we will consider it in the designs proposed here.

3 New Fault-Tolerant Architecture

Fig. 6 shows the architecture of the fault-tolerant system proposed here, whose two main blocks are: the Enhanced Lockstep scheme and the fault-tolerant Configuration Engine. It relies on using two Xilinx Virtex hardware primitives: the reconfiguration port ICAP and the FRAME_ECC [15] as well as the highly reliable external Golden Memory to store the configurations (whose study is out of the scope of this paper).

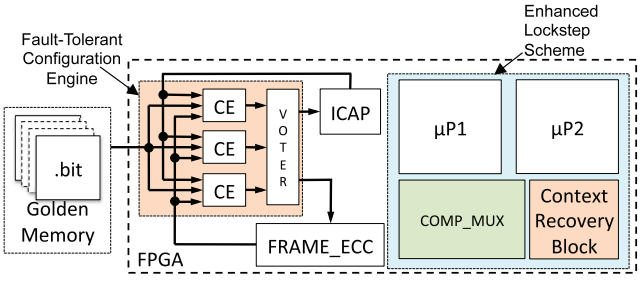

Fig. 6. Block scheme of the fault-tolerant architecture.

3.1 Enhanced Lockstep Scheme

The basic lockstep scheme is the realization of DWC at the processor level. It can be implemented in FPGA using, e.g., a softcore processor MicroBlaze. Unfortunately, it can only detect errors without indicating the faulty module. To alleviate this limitation, we propose here the new Enhanced Lockstep scheme (shown in Fig. 7) which is provided with means to identify the faulty processor, thus allowing to continue execution with the remaining fault-free processor.

Two identical MicroBlaze processors μP1 and μP2 are the heart of the Enhanced Lockstep scheme. Each of them is a complex system consisting of a 32-bit processor core, central bus, etc. (Fig. 3), designed using the procedure proposed in [28]. Their outputs are identical during fault-free functioning, any disagreement indicating error(s). A total of 150 bits of output signals of the PLB bus and the peripheral outputs are compared by the Comparator/Multiplexer (COMP_MUX). These signals were chosen because the PLB is the central bus of a MicroBlaze system and all the communications with the peripherals must go through it: its size and data throughput increase the likeliness to detect as soon as possible any error which could occur during processor operation. To note that each softcore instance should be fixed to a specific location in a reconfigurable zone (i.e., μP1 and μP2 are implemented in separate PRRs). Otherwise, all the resources of the two cores could be blended together, so that it would not be possible to distinguish the faulty one when an error is detected. Also, the location fixation allows for reconfiguration of the faulty core without interrupting the other.

The COMP_MUX consists of two blocks: 1) the Comparator that indicates any mismatch between the outputs Out1 and Out2 of μP1 and μP2 (containing the PLB and final output signals), and 2) the Multiplexer which connects one of the processors to the system output, so that if one of them is reported to be faulty, the other is switched on. The

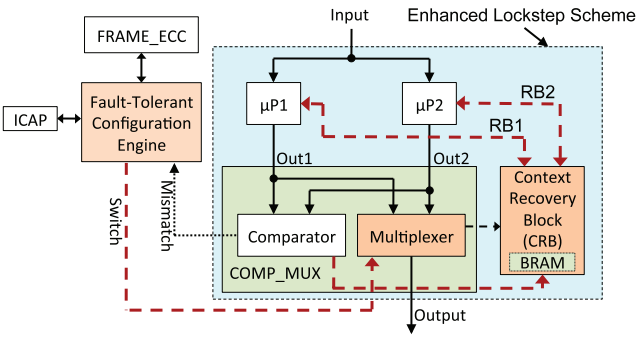

Fig. 7. Enhanced lockstep scheme.

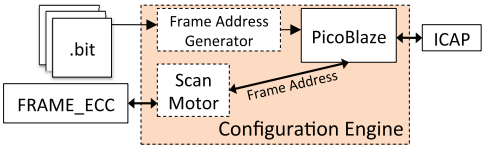

Fig. 8. Configuration engine.

switching is an atomic operation executed in one clock cycle. Once the error is localized by the FT Configuration Engine presented in Section 3.2, the affected processor is reconfigured to eliminate its configuration upset. Then, the two processors need to be synchronized to put the newly reconfigured one to the same state as the correct one, thus enabling them to continue executing the same task in Lockstep again. The recovery process of the Enhanced Lockstep scheme is handled by the Context Recovery Block (CRB). The CRB has a memory shared by both processors to store their context using on-chip BRAM that has dual-port connection providing simultaneous access to two Recovery Buses RB1 and RB2. These buses serve to save and restore the processors' contexts to the BRAM as well as to control the CRB during the context switching process. To note that the COMP_MUX itself is also a dynamic module that can be reconfigured in case of error by the FT Configuration Engine.

3.2 Fault-Tolerant Configuration Engine

The activation of the disagreement signal by the Comparator triggers the procedure of localizing the faulty processor which is executed by the Configuration Engine (Fig. 8). The Configuration Engine consists of a *Scan Motor*, a *Frame Address Generator*, and an 8-bit softcore processor PicoBlaze (chosen because of its small size). Reliable uninterrupted operation of the Configuration Engine is crucial for the correct functioning of the whole Enhanced Lockstep system. Because the Configuration Engine itself is also exposed to configuration upsets, to avoid a single point of failure, we made it fault-tolerant using TMR (i.e., the FT Configuration Engine consists of three Configuration Engines of Fig. 8 and a 2-out-of-3 majority voter). Moreover, if the FT Configuration Engine detects an error in one of its modules, the faulty module will be disconnected and then the FT Configuration Engine will reconfigure it by transferring the correct bitstream from the golden memory through the ICAP.

3.2.1 Scan Motor

The *Scan Motor* continuously works in the background (hence, it does not affect the user design) to point out the physical position of the configuration upset: it cyclically reads the configuration frame by frame using readback through ICAP and checks each frame for errors. The FRAME_ECC primitive is used to identify any changes in the configuration memory due to upsets. If it happens, the Scan Motor localizes and reports the erroneous Frame Addresses to the PicoBlaze, although the Configuration Engine is unable to identify the faulty module. The actual address of the faulty frame of the lockstep module is decoded by the method explained below in Section 3.2.2. The module-based reconfiguration of the faulty block (realized by the Configuration Engine) takes place by sending the appropriate bitstream to the ICAP.

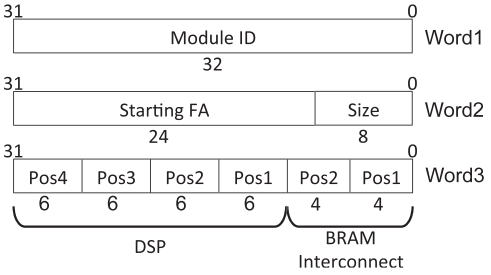

Fig. 9. Three words used to determine frame addresses.

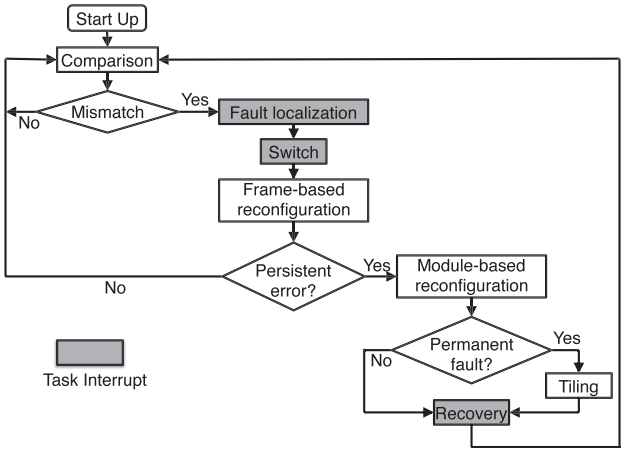

Fig. 10. Fault mitigation strategy for the enhanced lockstep scheme.

In a partially reconfigurable design, any reconfigurable module is physically assigned to a specific region. The addresses of the frames of that region which are occupied by a given module can be enumerated using RapidSmith [29]. In our design, once determined, they are compressed and stored in the internal memory of the PicoBlaze. If during scan process, an upset is detected in a frame which does not belong to any module, it is very likely that it is located in the interconnection between modules and then it can be removed by the frame-based reconfiguration.

3.2.2 Determination of Frame Addresses

The frame addresses of any reconfigurable module to be scanned (two softcores and the COMP_MUX for each entity of the Enhanced Lockstep scheme) can be determined from three 32-bit words (stored in the internal memory of the Configuration Engine) whose contents shows Fig. 9. Word1 contains the Module ID which is the identifier of any reconfigurable module, whose addressing space is sufficiently large to identify practically any number of modules which would fit into a single FPGA. Word2 contains the number of columns (Size) and the 24-bit Starting frame address needed to determine all other addresses of the frame of this module. To ensure that all columns (major addresses) are consecutive, each module spans only one whole row. Because there are the maximum of 25 columns inside a row in the selected FPGA and the largest reconfigurable module in our system (a softcore) requires 21 columns, eight bits suffice to determine the size of any module. Finally, because there are different numbers of minor addresses inside a major address (a column) depending on logic type (CLB, DSP, or BRAM interconnect), all relative positions of the special columns (BRAM interconnect and DSP) are stored inside the PRR of a module in 32-bit Word3.

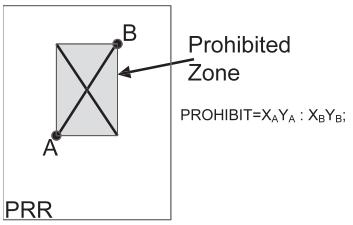

Fig. 11. Tiling technique using PROHIBIT constraint.

4 Fault Mitigation Strategy

Fig. 10 illustrates the fault mitigation strategy applied to deal with errors occurring in the Enhanced Lockstep scheme. During normal operation, the COMP_MUX continuously supervises the dual-core lockstep module, while the FT Configuration Engine scans all the reconfigurable blocks in the background. If the COMP_MUX triggers a mismatch signal, its cause can be a fault either in one of two softcore processors or in the COMP_MUX itself. The COMP_MUX sends a mismatch signal to the FT Configuration Engine to request for scanning of the Enhanced Lockstep scheme. The FT Configuration Engine starts to scan the COMP_MUX immediately and, if a fault is in the COMP_MUX, the FT Configuration Engine will reconfigure it. If the COMP_MUX is fault-free, the error must be due to a fault in one of the two cores, so the FT Configuration Engine only needs to scan one core to identify the source of error. The scan itself does not affect the core's task, although during this special scan, the Enhanced Lockstep scheme needs to be paused to prevent any catastrophic results.

If the error persists in a frame despite that configuration upsets have been recently corrected in it, a permanent configuration fault is declared. Depending on the fault duration, different reconfiguration techniques are selected: normal partial reconfiguration for a temporary fault or tiling technique [30], [31] for a permanent fault.

The principle of tiling avoids usage of the faulty zone of the FPGA by precompiling the same design with various implementations, each of which has the following two properties. First, it has a prohibited zone, so that a permanent fault can be masked by charging the appropriate configuration of the implementation in which the prohibited zone overlaps the faulty area. Second, it has its own bitstream, so that the fault masking process is done by downloading the appropriate bitstream through ICAP by executing the partial reconfiguration procedure. In our system, the bitstream generation for various implementation is done using a basic placement constraint PROHIBIT [32] which is assigned through the user placement constraint file provided to the synthesis tool. The prohibited rectangle zone is defined by the coordinates X and Y of two points (top-right and down-left). By varying these coordinates, the prohibited zone can be displaced inside the PRR.

5 State Recovery Procedure for the Enhanced Lockstep Scheme

Recall that the comparator can signal a mismatch caused by any upset in the dual processor core, without the capability of indicating the faulty core. However, the fault diagnostic

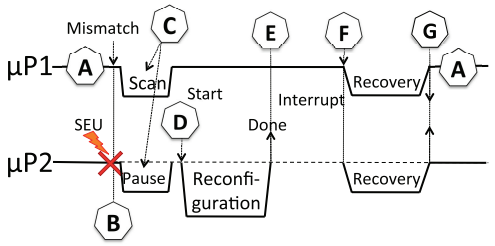

Fig. 12. Roll-forward state recovery procedure for the enhanced lockstep scheme.

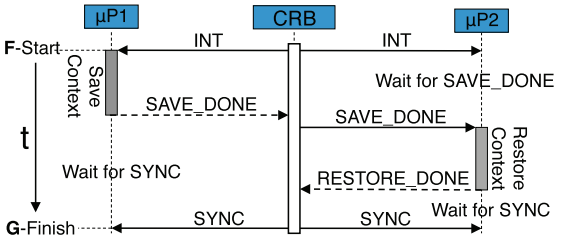

Fig. 13. Recovery process detail for the enhanced lockstep scheme.

process running in background (configuration scan and frame address determination) make possible to diagnose the faulty core. It is even likely that it would report an error despite the comparator indicates no mismatch, which would mean that a nonsensitive bit was flipped. To avoid accumulation of configuration upsets, it is corrected anyway using frame-based reconfiguration (which does not affect the processor operation). In the worst case, even if no errors were observed because of the dormant effect, they could still be detected by checkpointing afterward. Once a faulty processor is found, it is reconfigured to bring it back to its initial state as at the startup, which is followed by loading in it the same state as the other processor, so that its state recovery process can be launched.

Fig. 12 shows the scheme of the roll-forward procedure used to recover and synchronize the states of the two softcore MicroBlaze processors μP1 and μP2. The context of a softcore MicroBlaze processor is represented by the 32-bit values of 31 General Purpose Registers and two Special Registers: the Program Counter (PC) and the Machine Status Register (MSR) [33]. In the present version of our design, the context exchange between MicroBlaze processors is only applied to a monotask system without caches. Nevertheless, if necessary, the same context exchange strategy could be also applied to a more complex software (multitask and/or with caches), although it would involve longer recovery duration (the same problem would occur in case of TMR). During normal execution of both processors (**A**), if a mismatch signal is triggered (**B**), the scan process (**C**) is launched immediately to localize the error (in this example, only the COMP_MUX and μP1 are scanned while the μP2 is paused) and the reconfiguration process corrects the error (**D**). Once it is completed (**E**), μP2 signals its ready status for the state recovery procedure to the CRB. The moment of recovery is decided by μP1 (e.g., it can wait until the end of a critical task that should not be interrupted). To start the recovery process, μP1 signals the CRB inside the COMP_MUX to start recovery via RB1 bus. The CRB sends immediately an interrupt signal via buses RB1 and RB2 to announce the two processors to begin recovery (**F**). Once the recovery process completes, the two processors resynchronize themselves (**G**) and continue executing correctly the task (return to **A** state). During the period between **D** and **F**, only one processor is operational which could be an issue, should another fault appear in it. Nevertheless, this duration is very short (about 4 ms) and the probability of occurrence of two consecutive upsets in two processors within 4 ms in many applications seems rather low. According to [34], the nominal failure rate of Virtex-5 FPGA equals to 151 FIT/Mb with 95 percent confidence range in laboratory conditions (one FIT is one failure in 10^9 hours). Therefore, the probability of occurrence of one sensitive SEU during 4 ms in another processor (while one is being reconfigured) is about 2×10^{-17}. To deal with this highly unlikely event, the Configuration Engine can be programmed to scan simultaneously the unprotected processor to signal any SEUs. Recall also that partial protection against faults was considered an acceptable solution in various TMR-based designs as well [11], [12], [13], [14].

The recovery process is detailed in Fig. 13. The CRB launches an interrupt (**INT**) to force both processors to enter the recovery routine (**Start**). At the beginning, μP1 saves its context in the shared memory BRAM of the CRB, while μP2 is waiting for the context saving process to finish (**Wait for SAVE_DONE**). Then, μP2 starts restoring the context (**Restore Context**), while μP1 waits for the synchronization signal from the CRB. Once μP2 finishes restoring (**RESTORE_DONE**), it signals its ready status to the CRB (**Wait for SYNC**), so that the CRB could instantly send the synchronization signal (**SYNC**), and both processors would be liberated from recovery to continue tasks execution synchronously.

To avoid undetected errors in BRAMs of each processor, they were designed using Xilinx Design Suite v13.2, so that all single errors in BRAMs can be detected and corrected using Error Detection and Correction (EDAC) implementation in LMB BRAMs [35]. The ECC used is a (32,7) Hamming code, i.e., the ECC codeword has a total of 39 bits. Because BRAM is organized in 8-bit blocks, one additional BRAM block must be used to store 7 check bits, which accounts for 25 percent BRAM overhead. The EDAC circuitry occupies 92 slices which are all under the scan by the Configuration Engine as well.

Depending on the reliability level required, logic upsets could be treated accordingly. In the minimalist case, before each use, the whole CRB (including its CRB memory contents and control logic) is refreshed by partial reconfiguration, accompanied by internal reset to eliminate all undetected persistent errors. If error detection and correction is required, the system can perform checkpointing and rollback, provided that a golden copy of the context is stored in the CRB memory which could be also protected by ECC (as suggested in [36]) and would not require partial reconfiguration of its contents.

6 Implementation Details and Comparisons

We have implemented the system of Fig. 6 using Xilinx Virtex-5 XC5VSX50T FPGA and Xilinx Design Suite v13.2.

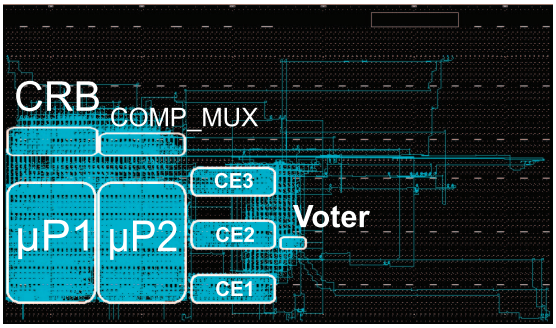

Fig. 14. A 90 degree-rotated layout view of the enhanced lockstep scheme.

The 90 degree-rotated layout view of the system is shown in Fig. 14. The dual lockstep cores operate at 125 MHz (i.e., one cycle takes 8 ns), whereas each of three PicoBlazes inside the FT Configuration Engine runs at 60 MHz (i.e., one cycle takes 16.7 ns).

6.1 Tiling Implementation

Because configuration scrubbing techniques cannot handle permanent faults, to deal with the latter, we have applied the *tiling* technique to the two processors (μP1 and μP2) and the COMP_MUX. Fig. 15 shows different implementations using the PROHIBIT macro (presented in Section 4) which allow to avoid using a zone inside the PRR of μP1. Notice that in each implementation, the corresponding prohibited zone has a different location.

The following tradeoffs are observed between the tiling granularity and the capacity of external memory which stocks tiling configuration bitstreams. The finer is the granularity (hence, the prohibited zone) of tiling, the more configurations are required to cover the whole PRR and the more external memory space is needed to store them. If a prohibited zone corresponds to a column, the number of needed configurations equals the number of columns occupied by the concerned block. The cost of stocking the bitstreams can be reduced considerably by generating only the differential bitstream between two adjacent columns, although it requires to solve two following major issues:

1. For two configurations, all frames of the other columns must be the same, except for the two prohibited zones (prohibited columns). So that only two related columns need to be modified when switching from one configuration to another.
2. The existence of persistent error(s) inside the frame containing a permanent fault within a column could be a major problem. If some configuration bits of the zone containing a permanent fault cause persistent error(s), the whole processor (including BRAM contents) must be reset in one of two ways applied on different levels:
   a. by the module-based reconfiguration of the whole processor followed by the reset signal controlled by the FT Configuration Engine, or
   b. by the global reset of the entire platform.

However, finding a procedure leading to determining the optimal solutions (related to both 1 and 2) is beyond the scope of this paper and will be considered in the future.

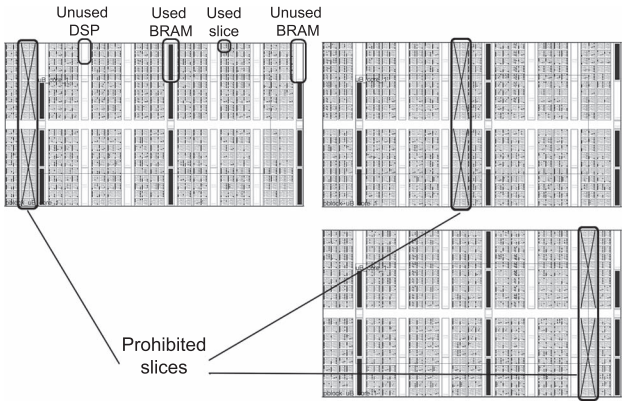

Fig. 15. Three sample tiling implementations of μP1.

TABLE 1
FPGA Resources Needed by Basic Blocks of the Enhanced Lockstep Scheme

Module	Required [slices]	BRAM [KB]	PRR Size [slices]
Configuration Engine	99		120
Voter	24		
MicroBlaze Processor	524	40	560
COMP_MUX	51		80
CRB	72	4	80
XC5VSX50T	8160	594	

6.2 Complexity Evaluation

Table 1 characterizes the complexity of the basic blocks of the system proposed. The FT Configuration Engine that supervises the whole FPGA device uses $3 \times 120 + 24 = 384$ slices (three Configuration Engines and the voter). This constant overhead appears only once in the FPGA device, independently on the number of lockstepped pairs of softcore processors in the device. The overhead involved in each item of the Enhanced Lockstep scheme present on the FPGA is one extra MicroBlaze Processor (560 slices and 40 KB BRAM), one COMP_MUX (80 slices), and one CRB (80 slices). The CRB contains 4 KB BRAM (the smallest amount available in the device matrix) that is shared by the two cores for the state recovery process, despite that 1 KB would be sufficient. Each Configuration Engine, the dual core lockstep block, and the COMP_MUX (except for the majority voter that drives the output signals of the FT Configuration Engine) are dynamically reconfigurable to be able to deal with configuration memory upsets. The only exception is the voter, which however occupies only 24 slices, so its upset probability is negligibly small compared to that of the whole system. If necessary, this small voter can be implemented using external hardened components or using Xilinx TMR solution XTMR [37], [38] wherein the voter is also triplicated. Finally, because one BRAM column contains 16 KB and one MicroBlaze Processor requires 40 KB of BRAM including ECC, to put it into one PRR, 48 KB BRAM must be used, which corresponds to three BRAM columns. For the same reason, CRB uses 16 KB of BRAM, even that 4 KB could be sufficient.

TABLE 2
Partial Reconfiguration Overhead and Bitstreams Needed for the System (without Tiling)

Module	PR overhead [slices]	Bitstream size [KB]	Reconf. time [μs]
3 × CE	3 × 21	3 × 44	752
2 × μP	2 ×36	2 × 176	3000
COMP_MUX	29	12	205
CRB	8	50	853
Total	172	570	

Table 2 shows the partial reconfiguration hardware overhead due to PRR partitioning. The PRR size and the required resources for one Configuration Engine are 120 and 99 slices, respectively (including the overhead of $120 - 99 = 21$ slices for one Configuration Engine). The partial reconfiguration overheads for one softcore, the COMP_-MUX, and the CRB are, respectively, 36, 29, and 8 slices. Bitstream sizes of PRRs and related module-based reconfiguration times are also included, for completeness. The reconfiguration bandwidth of 60 MB/s was assumed, which corresponds to 8-bit ICAP interface operating at 60 MHz and 16-bit Flash "Golden Memory" accessed at 30 MHz.

The duration of task interrupt t_i (in case of a single error) will be evaluated depending on the source and type of error. A nonpersistent error necessitates frame-based reconfiguration (frame reload) to eliminate the error without interrupting the module operation. A persistent error requires module-based reconfiguration which resets the reconfigured module.

1. An error in the COMP_MUX
   a) A nonpersistent error: $t_{i_{1a}} = t_{sCM}$;
   b) a persistent error: $t_{i_{1b}} = t_{sCM} + t_{rCM}$,
   where t_{sCM}—the scan time of COMP_MUX and t_{rCM}—the reconfiguration time of COMP_MUX.
2. An error in a processor of the lockstepped pair
   a) A nonpersistent error: $t_{i_{2a}} = t_{sCM} + t_{sP} + t_{sw}$;
   b) a persistent error: $t_{i_{2b}} = t_{sCM} + t_{sP} + t_{sw} + t_{rec}$,
   where t_{sP}—the time to scan one processor, t_{sw}—the time to switch the COMP_MUX, and t_{rec}—the time to recover the processor context in case of persistent error.

The FT Configuration Engine requires 41 clock cycles to complete a frame scan. The number of configuration frames occupied by COMP_MUX and one processor are 72 (80 slices) and 604 (560 slices), respectively. The contents of BRAM varies during normal operation; hence, it is really difficult to check the correctness of its contents using configuration scan. Nevertheless, because the BRAM contents is protected by ECC, the BRAM frames of both the processor and the CRB are excluded from the scan. The maximum time to localize a fault is that required for scanning the COMP_MUX and one processor, i.e., $t_{sCM} = 72 \times 41 \times 16.7$ ns $= 49.2$ μs and $t_{sP} = 604 \times 41 \times 16.7$ ns $= 413.6$ μs. The minimum time is that required to scan one frame—690 ns. If the FT Configuration Engine detects an error in the COMP_MUX, the reconfiguration of COMP_MUX takes $t_{rCM} = 205$ μs. The time for COMP_MUX to switch the output of the lockstepped pair module—only one clock cycle equal to 8 ns—is negligible.

Fig. 16 shows the C code of the processor context saving and restoring processes. The registers R1-R31, the Machine Status Register, and the Program Counter register are saved from the starting address 0x83C18000 of the BRAM inside the CRB. The context saving and restoring processes need, respectively, 272 cycles (272×8 ns $= 2.2$ μs) and 325 cycles (2.6 μs). The time of the whole recovery process of Fig. 13, consisting of the durations of context saving, restoring, and control, is $t_{rec} = 5.2$ μs. Therefore, when a persistent error occurs in the last frame to be scanned, the maximum system time overhead resulting from the duration of task interrupt can be determined as follows:

1. If a nonpersistent or persistent error occurs in the COMP_MUX, it equals to $t_{i_{1a}} = 49.2$ μs or $t_{i_{1b}} = 49.2 + 205 = 254.2$ μs, respectively.
2. If a nonpersistent or persistent error occurs in the lockstepped pair module, it equals to $t_{i_{2a}} = 49.2 + 413.6 = 462.8$ μs or $t_{i_{2b}} = 49.2 + 413.6 + 5.2 = 468$ μs, respectively.

Besides the maximum durations of tasks interrupts due to persistent errors, Table 3 shows minimum durations when a persistent error occurs in the first frame to be scanned.

6.3 Comparison with Other Designs

Table 4 summarizes various characteristics of different single-processor architectures, all using MicroBlaze softcore. To the best of our knowledge, only two FPGA architectures suitable to implement a fault-tolerant softcore processor were reported in the literature. One is a lockstep system using hardcore embedded processor PowerPC [3], and the other is the only fault-tolerant FPGA implementation using the MicroBlaze softcore processor using TMR [5]. To have some reference complexity figures, we have also implemented the basic lockstep scheme. In our Enhanced Lockstep scheme, the two softcores and the voter are bound

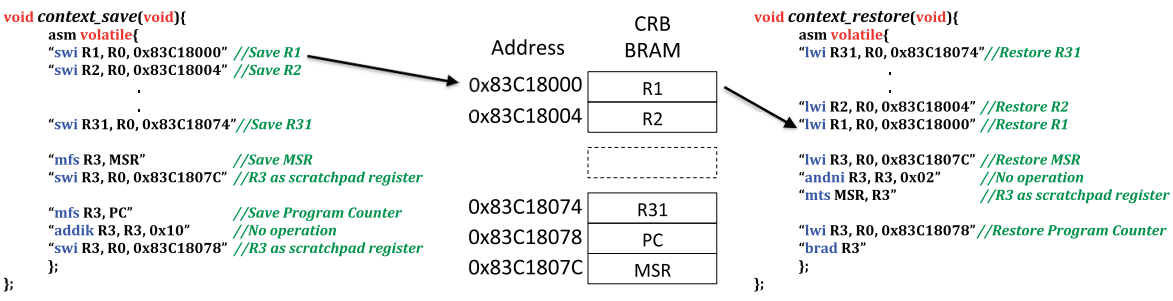

Fig. 16. Context recovery C code.

TABLE 3
Durations of Task Interrupts Depending on Error Types, Affected Modules, and Error Positions (in μs)

Interrupt duration	COMP_MUX		Processor	
	Non-persist.	Persistent	Non-persist.	Persistent
Min	0.69	205.7	49.9	55.1
Max	49.20	254.2	462.8	468.0
Average	24.95	230.0	256.8	261.6

to one PRR. We have found that the hardware overhead of the basic lockstep is the same as of the proposed Enhanced Lockstep solution: either system consumes twice of slices as the simple processor. Because there must be a fault-tolerant spare processor which controls the reconfiguration process of the dual-core module in case of mismatch, we have chosen the TMR implementation of the 8-bit PicoBlaze (like in the FT Configuration Engine). Obviously, it uses the same amount of slices as the FT Configuration Engine, the COMP_MUX, and the CRB ($384 + 80 + 80 = 544$ slices, i.e., 97 percent of those of the simple processor using 560 slices). As for BRAM, a simple processor requires 32 KB, whereas our Enhanced Lockstep scheme requires 112 KB consisting of two copies of BRAM of each processor (2×48 KB) and the BRAM of the CRB (16 KB). Because the TMR softcore system from [5] uses ECC for protecting BRAM data against errors, it uses a total of 48 KB BRAM.

Table 4 shows that our solution requires significantly lower time overhead than the basic lockstep system. First, because the scan process which pauses the system is significantly shorter than the reconfiguration process of the whole dual-core block to correct an error according to the basic lockstep scheme. (The maximum time overhead when an error occurs in the Enhanced Lockstep is the scan time and the proposed roll-forward recovery time.) Second, the time overhead of the Enhanced Lockstep scheme, dominated by t_{scan}, is about 95 times larger than $t_{rec} = 5.2\mu$s (the proposed roll-forward recovery duration). Furthermore, the proposed roll-forward recovery strategy used in our system is always faster than the checkpointing and rollback strategies used in the basic lockstep scheme. When an error occurs in the basic lockstep scheme, the time overhead is dominated by the correction process using dynamic reconfiguration (about 1,150 times longer than one roll-forward recovery). Moreover, the major drawback of the basic lockstep scheme is low continuity of services because, after error occurrence, its dual-core module needs to be reconfigured immediately to correct error.

Comparing single fault-tolerant softcore processors, one using the TMR approach from [5] and our using Enhanced Lockstep scheme, it is seen that the latter enjoys an advantage of smaller hardware overhead. This advantage grows with the number of processors—due to the presence of only one FT Configuration Engine per FPGA device. An FT-MPSoC system applying our approach incurs a constant hardware overhead of the FT Configuration Engine (384 slices) and 720 slices (114 percent of a processor) for each added lockstepped pair (560 slices for one processor, 80 slices for one COMP_MUX, and 80 slices for one CRB). In an FT-MPSoC system applying TMR, 1,120 slices (2×560) overhead is required for each added processor. Table 5 compares hardware overhead of the two approaches depending on the number of processors. The more processors the FT-MPSoC contains, the relatively lower hardware overhead our system requires—thanks to the presence of only one FT Configuration Engine in the FPGA. In particular, all slices available in the XC5VSX50T FPGA suffice to construct an FT-MPSoC system containing six processors using Enhanced Lockstep methodology and only four processors using TMR (shaded cells in Table 5). Unfortunately, the tradeoff is significantly longer error recovery time overhead of our design.

TABLE 4
Comparison of Single-Processor Architectures

	Simple	Basic Lockstep	TMR [5]	Enhanced Lockstep
Slices [%]	100	297	384	297
BRAM [KB]	32	64	48	112
Time Overhead	–	t_{cor}=1150t_{rec}	t_{rec}	t_{scan}=95t_{rec}
Services Contin.	No	Low	High	High

TABLE 5
Usage of Slices by FT-MPSoC Systems (560 Slices = 100 Percent; Xilinx FPGA XC5VSX50T = 1,457 Percent)

No. of processors	1	2	3	4	5	6	7
Ours [%]	297	525	753	981	1209	1437	1665
TMR from [5] [%]	384	699	1014	1329	1644	1959	2274

6.4 Handling Logic Errors

Many fault-tolerance techniques in reconfigurable architectures are intended to deal with SEUs in configuration memory as the major source of concern [2], [8], [9], [10], [21]. Compared to state-of-the-art solutions dealing with configuration memory upsets in softcore systems, our methodology offers better performance. As for logic errors (due to upsets in registers, flip-flops, etc.), TMR has been a commonly accepted strategy [38]. In our system, these types of errors are handled using the following strategy involving low hardware and timing costs. Periodically, the two cores perform checkpointing to allow for recovery from upsets affecting logic, if needed. During checkpointing, the cores' contexts are copied into the first half of the CRB memory. While copying the context, the data present in the Recovery Buses are compared to detect any mismatch. If the two contexts agree, the context of μP1 is saved as correct into the second half of the CRB memory and, should a mismatch be detected during next checkpointing, it is used for rollback,

To deal with logic errors, the Configuration Engine must scan the COMP_MUX and each core of the lockstepped pair. If the scan process cannot identify the faulty module while the comparator still continues to signal a mismatch, it is assumed that an SEU caused a logic error. As a consequence, the scan process could last longer, because more modules need to be scanned and within these modules all the frames must be scanned (max scan time), which would involve more timing overhead.

As already mentioned in Section 2.3.2, the probability that an SEU affects logic could be three orders of magnitude lower than for configuration bits. That is why our system

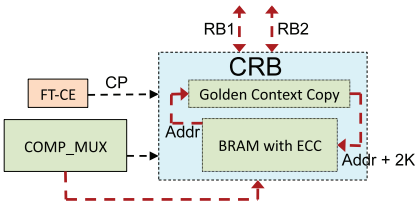

Fig. 17. New CRB to deal with upsets affecting logic.

can perform checkpointing and rollback significantly less frequently than the classical lockstep scheme (T_{CP} in Fig. 5 could be significantly longer), because the latter treats upsets affecting both configuration and logic uniformly. Consequently, the resulting timing overhead (related to checkpointing and rollback) of the Enhanced Lockstep scheme is significantly lower.

Fig. 17 shows the new CRB modified to deal with upsets affecting logic. Before each checkpointing driven by the FT Configuration Engine (FT-CE) via the CP signal, the CRB is refreshed through partial reconfiguration, followed by internal reset to eliminate any undetected persistent errors, except for BRAM whose contents is protected by ECC. During checkpointing, the contexts of two cores are compared and the context of μP1 is copied by default into the first half (Addr) of the CRB memory (BRAM with ECC). Because the Recovery Buses RB1 and RB2 connect also to the PLB bus and the COMP_MUX compares the PLB signals, there is no need to compare RB1 and RB2, as this comparison is already performed by the COMP_MUX. If no error is signaled by the COMP_MUX during checkpointing, this context is assumed correct and copied into the second half $(\mathrm{Addr} + 2\mathrm{K})$ which serves as a golden copy for rollback (Golden Context Copy in Fig. 17).

As for the additional hardware overhead needed to deal with upsets affecting logic, the CRB can also be used but at the cost of extra 84 slices, which include four slices of the Golden Context Copy block and 80 slices of the EDAC circuitry of the (32,7) Hamming code (see [36]).

7 Validation Using Fault Injection

To validate the efficiency of the fault-tolerant approach proposed, we have provided the FT Configuration Engine with the possibility to carry out automatic fault injection campaigns for the configuration memory of μP1 and the COMP_MUX. The goal is to obtain some statistical data on the design robustness, measured by the percentages of sensitive bits and persistent errors. However, notice that the system malfunctions occurring due to the faults in the BRAM of the CRB are not easy to observe, because the contents of BRAMs change during operation, so the sensitivities and error persistences of the BRAM bits vary as well. As a consequence, it is nearly impossible to obtain the statistically meaningful fault injection results for the BRAM. Moreover, because here BRAM blocks are protected against errors by ECC, faults are not injected in the BRAM of μP1 and the CRB. Therefore, we have considered only single configuration bit flips injected using the frame-based reconfiguration through ICAP according to the fault injection procedure shown in Fig. 18.

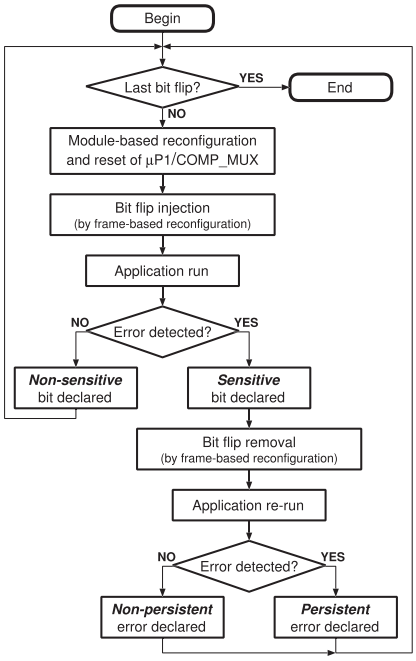

Fig. 18. Flow diagram of the fault injection procedure.

Recall that μP1 and the COMP_MUX occupy 604 and 72 configuration frames (which are separate for these two blocks), respectively. Because each configuration frame contains 41 32-bit words, i.e., 1,312 bits, the total of 792,448 bits (604 frames $\times$1,312 bits) and 94,464 bits (72 frames $\times$1,312 bits) are related to μP1 and the COMP_MUX, respectively. To evaluate the sensitivity of configuration bits and persistence of errors of the Enhanced Lockstep scheme, single bits are flipped in randomly chosen locations in about 5 percent of the total bits (exactly 40,000 and 4,000 configuration bits of μP1 and the COMP_MUX, respectively).

Once a fault is injected, the lockstepped pair of processors μP1 and μP2 executes the application program that checks whether a peripheral of a softcore operates correctly. This application was chosen, because it involves a relatively large number of transactions passing through PLB bus and thus offers the advantage that COMP_MUX can verify the consistency of outputs of μP1 and μP2 for frequently varying data. Although this application program does not cover the complete set of instructions, it is nevertheless a valuable proof of concept for the proposed system. For each fault, the application starts from the same initial state and runs during 120×10^6 cycles = 0.96 s. The procedure flow differs depending on whether COMP_MUX detects any mismatch during this period:

1. If there is no mismatch, the bit is declared nonsensitive and the injection fault experiment terminates for this bit. Then, the tested block (μP1 or COMP_MUX) is reconfigured with a correct bitstream using module-based reconfiguration (to eliminate any undetected persistent errors that could have resulted from the last bit flip, and thus make the FPGA ready for the next bit flip).
2. The first mismatch detected during this period indicates that the bit is sensitive and hence the first

TABLE 6
Statistical Results of Fault Injection

Module	Slices	Utilization	Injected bit flips	Sensitive bits	Sensitivity	Persistence
μP1	560	93%	39360	3165	8.6%	2.3%
COMP_MUX	80	83%	3936	83	2.1%	0.4%

application run is stopped. To determine whether this sensitive bit causes a persistent error or not, the bit flip is first corrected by replacing the faulty frame with the correct one by frame-based reconfiguration. Then, the same application is rerun from the very beginning for another 0.96 seconds. Any mismatch signal triggered during the second application run is attributed to the presence of a persistent error. As in the previous case, the module-based reconfiguration is launched to correct this error and make the FPGA ready for another bit flip.

Finally, notice that any bit flip injected in COMP_MUX may result in only two outcomes: either it has no impact on COMP_MUX (no mismatch signal triggered), or COMP_-MUX triggers erroneously a mismatch signal, which allows us to qualify the bit as nonsensitive or sensitive, respectively.

Table 6 shows the results of fault injection campaigns carried out for the single Enhanced Lockstep scheme implemented on Virtex-5 FPGA, involving the slice configuration frames of μP1 and the COMP_MUX (but excluding the BRAM configuration frame of the CRB). Because usually the PRRs are larger than the required resources of the reconfigurable module, the Utilization column in Table 6 represents the occupation ratios of the related blocks. Fault injection experiments reveal that only 8.6 and 2.1 percent configuration bits of μP1 and COMP_MUX, respectively, are sensitive. More important, only a negligible percentage of these few sensitive configuration bits actually cause persistent errors—2.3 and 0.4 percent for μP1 and COMP_MUX, respectively, i.e., on average, only one out of 43 sensitive configuration bit flips in μP1 and as few as one out of 250 sensitive configuration bit flips in COMP_MUX causes a persistent error (which confirms some observations made earlier by other researchers). The considerably larger sensitivity of μP1 than of the COMP_MUX explains why one of the two processors (μP1 or μP2) is scanned before the COMP_MUX. In particular, the fault injection engine can also be used to find out sensitive bits of all configuration frames—to optimize the scanning order: in case of error, the configuration frames can be scanned in order of their decreasing sensitivities, beginning with the configuration frame containing the most sensitive bits.

Based on the fault injection results given in Table 6, the average interrupted duration caused by a configuration error in μP1 is: $261.6\ \mu s \times 2.3\% + 256.8\ \mu s \times (8.6 - 2.3)\% = 23\ \mu s$. Now, consider the basic lockstep system without the identification capability of the faulty core. The two processor cores must be reconfigured at the same time to eliminate the sensitive error, which requires 6 ms (2×3 ms) to complete. Therefore, the average duration of interrupt caused by a configuration error in the basic lockstep is: $6\text{ ms} \times 8.6\% = 516\ \mu s$, which is 22.4 times longer than in our Enhanced Lockstep. Notice also that this $516\ \mu s$ has not yet taken into account the duration of task interrupts due to checkpointing and rollback, should they be integrated in the system.

8 Conclusion and Perspectives

In this paper, we have proposed a new architecture of a fault-tolerant reconfigurable system which can be implemented on any SRAM-based FPGA with integrated softcore processors. An Enhanced Lockstep scheme built using a pair of MicroBlaze cores was proposed and implemented on Xilinx Virtex-5 FPGA. Unlike the basic lockstep scheme, ours allows to identify the faulty core using a Configuration Engine which allows to recover from single-event upsets through partial reconfiguration combined with roll-forward recovery technique. As a result, the problem of fault latency is alleviated, because faults are detected immediately, once they cause an error. To handle permanent faults, the tiling technique was also suggested. The Configuration Engine was built using PicoBlaze cores and, to avoid a single point of failure, was implemented as fault-tolerant using triple modular redundancy (TMR). The scheme proposed is a valuable alternative to other fault-tolerant softcore schemes: it requires significantly shorter error recovery time compared to conventional lockstep scheme, and uses significantly smaller number of slices compared to known TMR-based design although at significantly longer error recovery time. The advantage of lower hardware overhead of the new scheme compared to TMR grows steadily with the number of processors, when applied to build a fault-tolerant Multiprocessor System-on-Chip. The efficiency of the proposed approach was validated through fault injection experiments. Finally, to deal with SEUs affecting logic which are significantly less likely than those affecting configuration, low hardware and timing overhead technique of checkpointing and rollback was suggested.

We believe that the proposed system can be applied to the new Virtex-6 devices as well (which also contain the FRAME_ECC and ICAP primitives), whereas the design methodology proposed can be adapted to some other softcore processors like Nios, LEON, etc. The same mechanism of context recovery could be performed, because the contexts of these softcore processors are usually contained in 32 registers as well. Should another processor with different number of registers be targeted, one would have only to adjust the memory size of the Context Recovery Block. If the number of registers increases, it could involve extended duration of the context recovery processes, which could impact the standby period of the system when SEUs occur.

Future work would include developing the methodology which would facilitate the creation of tiling configurations using Xilinx Design Language (XDL) representation of the circuit accompanied by RapidSmith taking into account the sensitivity and persistence of errors caused by faults of the configuration bits inside the permanently defected zone. We would like also to consider fault injection in user's logic and BRAM to validate the significant advantages of our system in dealing with SEUs directly affecting logic.

ACKNOWLEDGMENTS

The work of M.-P. Pham was supported by Conseil General des Cotes d'Armor—CG22 and the CIFAER project (No. ANR-07-ARFU-002-02). Stanisław J. Piestrak research was done when he was with IRISA/INRIA, Research Team CAIRN, 22305 Lannion, France, on leave from University of Metz, 57070 Metz, France. Sébastien Pillement research was done when he was with University of Rennes 1 and IRISA/INRIA, Res. Team CAIRN, 22305 Lannion, France. The authors would like to thank the reviewers for their insightful comments which significantly contributed to improving this paper.

REFERENCES

[1] Xilinx, Inc., "PowerPC 405 Processor Block Reference Guide (UG018)," www.xilinx.com/support/documentation/user_guides/ug018.pdf, 2012.
[2] Xilinx, Inc., "Single-Event Upset Mitigation Selection Guide,"-Appl. Note XAPP987 (v1.0), http://www.xilinx.com/support/documentation/application_notes/xapp987.pdf, Mar. 2008.
[3] Xilinx, Inc., "PPC405 Lockstep System on ML310,"Appl. Note XAPP564 v1.0.2, http://www.xilinx.com/support/documentation/application_notes/xapp564.pdf, Jan. 2007.
[4] F. Abate et al., "New Techniques for Improving the Performance of the Lockstep Architecture for SEEs Mitigation in FPGA Embedded Processors," *IEEE Trans. Nuclear Science,* vol. 56, no. 4, pp. 1992-2000, Aug. 2009.
[5] Y. Ichinomiya et al., "Improving the Robustness of a Softcore Processor against SEUs by Using TMR and Partial Reconfiguration," *Proc. IEEE Ann. Int'l Symp. Field-Programmable Custom Computing Machines,* pp. 47-54, May 2010.
[6] Aeroflex Gaisler AB, "LEON3 Product Sheet," www.aeroflex.com/gaisler, 2012.
[7] Aeroflex Gaisler AB, "LEON3-FT SPARC V8 Processor," www.aeroflex.com/gaisler, 2012.
[8] Xilinx, Inc., "SEU Strategies for Virtex-5 Devices,"XAPP864, http://www.xilinx.com/support/documentation/application_notes/xapp864.pdf, Mar. 2009.
[9] M. Lanuzza et al., "An Efficient and Low-Cost Design Methodology to Improve SRAM-Based FPGA Robustness in Space and Avionics Applications," *Proc. Int'l Workshop Reconfigurable Computing: Architectures, Tools and Applications,* vol. 5453, pp. 74-84, 2009.
[10] B. Dutton and C. Stroud, "Single Event Upset Detection and Correction in Virtex-4 and Virtex-5 FPGAs," *Proc. ISCA 24th Int. Conf. Computers and Their Applications (CATA 2009),* pp. 57-62, Apr. 2009.
[11] S.-F. Liu et al., "Increasing Reliability of FPGA-Based Adaptive Equalizers in the Presence of Single Event Upsets," *IEEE Trans. Nuclear Science,* vol. 58, no. 3, pp. 1072-1077, June 2011.
[12] M. Alderighi et al., "Evaluation of Single Event Upset Mitigation Schemes for SRAM Based FPGAs Using the FLIPPER Fault Injection Platform," *Proc. 22nd IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems,* pp. 105-113, Sept. 2007.
[13] B. Pratt et al., "Fine-Grain SEU Mitigation for FPGAs using Partial TMR," *IEEE Trans. Nuclear Science,* vol. 55, no. 4, pp. 2274-2280, Aug. 2008.
[14] H. Quinn, P. Graham, and B. Pratt, "An Automated Approach to Estimating Hardness Assurance Issues in Triple-Modular Redundancy Circuits in Xilinx FPGAs," *IEEE Trans. Nuclear Science,* vol. 55, no. 6, pp. 3070-3076, Dec. 2008.
[15] Xilinx, Inc., "Virtex-5 FPGA Configuration User Guide (UG191 v3.6)," www.xilinx.com/support/documentation/user_guides/ug191.pdf, 2009.
[16] Xilinx, Inc., "Two Flows for Partial Reconfiguration: Module Based or Small Bit Manipulations (XAPP290)," 2002.
[17] Xilinx, Inc., "Early Access Partial Reconfiguration (v1.2),"User Guide UG208, Sept. 2008.
[18] Aeroflex Gaisler AB, "LEON4 32-bit Processor Core," www.aeroflex.com/gaisler, 2010.
[19] Xilinx, Inc., "LogiCORE IP Processor Local Bus (PLB),"v4.6 (v1.05a), Data Sheet DS531, http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf, Sep. 2010.
[20] K. Morgan et al., "SEU-Induced Persistent Error Propagation in FPGAs," *IEEE Trans. Nuclear Science,* vol. 52, no. 6, pp. 2438-2445, Dec. 2005.
[21] Xilinx, Inc., "Correcting Single-Event Upsets through Virtex Partial Configuration,"Appl. Note XAPP216 v1.0, www.xilinx.com/support/documentation/application_notes/xapp216.pdf. June 2000.
[22] Xilinx, Inc., "Virtex-5 FPGA User-Guide,"UG190, v4.5, p. 383, Jan. 2009.
[23] F. Sturesson, S. Mattsson, C. Carmichael, and R. Harboe-Sorensen, "Heavy Ion Characterization of SEU Mitigation Methods for the Virtex FPGA," *Proc. European Conf. Radiation and Its Effects on Components and Systems (RADECS),* pp. 285-291, 2001.
[24] E. Fuller et al., "Radiation Testing Update, SEU Mitigation, and Availability Analysis of the Virtex FPGA for Space Re-Configurable Computing," *Proc. Int'l Conf. Military and Aerospace Programmable Logic Devices (MAPLD),* Sept. 2000.
[25] D. Pradhan, *Fault-Tolerant Computer System Design.* Prentice-Hall, 1996.
[26] D.K. Pradhan and N.H. Vaidya, "Roll-Forward and Rollback Recovery: Performance-Reliability Trade-Off," *IEEE Trans. Computers,* vol. 46, no. 3, pp. 372-378, Mar. 1997.
[27] D.K. Pradhan and N.H. Vaidya, "Roll-Forward Checkpointing Scheme: A Novel Fault-Tolerant Architecture," *IEEE Trans. Computers,* vol. 43, no. 10, pp. 1163-1174, Oct. 1994.
[28] H.-M. Pham, S. Pillement, and D. Demigny, "A Fault-Tolerant Layer for Dynamically Reconfigurable Multi-Processor System-on-Chip," *Proc. Int'l Conf. ReConFigurable Computing and FPGAs,* pp. 284-289, Dec. 2009.
[29] C. Lavin et al., "Rapid Prototyping Tools for FPGA Designs: RapidSmith," *Proc. Int'l Conf. Field-Programmable Technology (FPT),* pp. 353-356, Dec. 2010.
[30] A. Kanamaru et al., "Tile-Based Fault Tolerant Approach Using Partial Reconfiguration," *Proc. Int'l Workshop Reconfigurable Computing: Architectures, Tools and Applications,* pp. 293-299, 2009.
[31] W.-J. Huang and E.J. McCluskey, "Column-Based Precompiled Configuration Techniques for FPGA Fault Tolerance," *Proc. Ann. Int'l IEEE Symp. Field-Programmable Custom Computing Machines,* pp. 137-146, 2001.
[32] Xilinx, Inc., "Constraints Guide (UG625 v13.2)," http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_2/cgd.pdf, July 2011.
[33] Xilinx, Inc., "MicroBlaze Processor Reference Guide (UG081 v10.3),"http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf, 2009.
[34] A. Lesea, "Continuing Experiments of Atmospheric Neutron Effects on Deep Submicron Integrated Circuits (WP286)," www.xilinx.com/support/documentation/white_papers/wp286.pdf, 2009.
[35] Xilinx, Inc., "IP Processor LMB BRAM Interface Controller (v3.00b)," June 2011.
[36] Xilinx, Inc., "Single Error Correction and Double Error Detection (XAPP645 v2.2)," http://www.xilinx.com/support/documentation/application_notes/xapp645.p df, Aug. 2006.
[37] Xilinx, Inc., "Xilinx TMRTool," http://www.xilinx.com/publications/prod_mktg/XTMRTool_ssht.pdf, 2012.
[38] Xilinx, Inc., "Triple Module Redundancy Design Techniques for Virtex FPGAs,"Appl. Note XAPP197, v1.0.1, July 2006.

Hung-Manh Pham received the MSc degree in engineering science from the Pierre and Marie Curie University (UPMC) in Paris, France, in 2007, and the PhD degree in signal processing and telecommunication from the University of Rennes 1, France, in 2010. In 2011, he was a postdoctoral research assistant with CAIRN INRIA Res. Team, IRISA Lab. (Research Institute in Computer Science and Random Systems), in Lannion, France. Since 2012, he is the Director of the R&D Technology Center of the Vietnam Post and Telecommunication Industry JSC (VNPT Technology JSC). His research interests include design of embedded systems, dynamically reconfigurable systems and fault-tolerant computing. He is a student member of the IEEE.

Sébastien Pillement received the PhD degree and Habilitation degrees in computer science, respectively, from the University of Montpellier II and the University of Rennes 1. Since 2012, he is a full professor at Ecole Polytechnique de l' Université de Nantes, France. From 1999 to 2012 he was with IUT in Lannion, the subdivision of the University of Rennes 1, France, and was also a research member of the CAIRN INRIA Res. Team of the IRISA Lab. (Research Institute in Computer Science and Random Systems). His research interests include dynamically reconfigurable architectures, system on chips, design methodology and Network on Chip (NoC)-based circuits. He focuses his research on designing flexible and efficient architectures managed in real time. He is the author or coauthor of about 100 journal and conference papers. He is a member of the IEEE.

Stanisław J. Piestrak received the PhD and Habilitation degrees in computer science in 1982 and 1996, respectively, from the Wroclaw University of Technology and Gdansk University of Technology, both in Poland. He is a full professor at the University of Lorraine (created in 2012 from the fusion of four universities, including the University of Metz which he joined in 2004), France. During three academic years from 2008 to 2011, he was on leave with the CAIRN INRIA Research Team of the IRISA Laboratory in Lannion. Until 2004, he was with the Institute of Engineering Cybernetics at the Wroclaw University of Technology where he became a professor in 1999. While in Poland, he was invited by various universities abroad: as a visiting assistant professor at the University of Southwestern Louisiana in Lafayette, during the academic year 1984/5 and University of Georgia in Athens, during two academic years 1985-1987. He was also the JSPS visiting scientist at Tokyo Institute of Technology, Japan during the academic year 1993/1994. On numerous occasions, he also visited several French universities including TIMA/INPG in Grenoble, University of Rennes 1/ENSSAT in Lannion, and University of Metz. His research interests include design and analysis of VLSI digital circuits, fault-tolerant computing (in particular, self-checking circuits design, coding theory, and reconfigurable systems), and computer arithmetic (design of RNS-based hardware for high-speed digital signal processing). He is a member of the IEEE.