IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 14 October 2023, accepted 6 November 2023, date of publication 13 November 2023, date of current version 21 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3332294

RESEARCH ARTICLE

Supervised Triple Macrosynchronized Lockstep (STMLS) Architecture for Multicore Processors

PABLO M. AVILES, (Graduate Student Member, IEEE), JOSE A. BELLOCH, (Member, IEEE), LUIS ENTRENA, (Member, IEEE), AND ALMUDENA LINDOSO, (Senior Member, IEEE)
Electronic Technology Department, University Carlos III of Madrid, 28911 Madrid, Spain

Corresponding author: Almudena Lindoso (alindoso@ing.uc3m.es)

This work was supported in part by the Spanish State Investigation Agency under Project PID2022-138696OB-C21, and in part by the Madrid Government (Comunidad de Madrid-Spain) under the Multiannual Agreement with University Carlos III of Madrid (UC3M) in the Line of Excellence of University Professors under Grant EPUC3M26 and in the Context of the V Regional Programme of Research and Technological Innovation (PRICIT).

ABSTRACT In various fields, such as those with high-reliability requirements, there is a growing demand for high-performance microprocessors. Whereas commercial microprocessors offer a good trade-off between cost, size, and performance, they often need to be adapted to meet the reliability demands of safety-critical applications. To address this challenge, a Supervised Triple Macrosynchronized Lockstep architecture for multicore processors is presented in this work. Multiple recovery mechanisms, including rollback and roll-forward, have been implemented to harden the system. By integrating these mechanisms, the microprocessor becomes more robust and capable of mitigating potential errors or failures that may occur during operation. A quad-core ARM Cortex-A53 processor has been used as a case study, and an extensive fault injection campaign in the register file has been conducted to evaluate the effectiveness of our proposed approach. The results show that the hardened system exhibits high reliability, with 100% error coverage and error correction capabilities of up to 86.40%.

INDEX TERMS ARM, fault tolerance, lockstep, microprocessor, radiation hardening.

I. INTRODUCTION

Nowadays, electronic systems are becoming more and more prevalent in our lives, and the demand for these systems to exhibit high reliability is also on the rise. Reliable high-end microprocessors are required in safety-critical applications and harsh radiation environments, including aerospace applications where the need for computational capacity is increasing. Current available radiation-hardened (rad-hard) microprocessors cannot provide the required performance, size, weight, and power consumption [1], [2], [3]. As a result, Space agencies, private actors, and satellite manufacturers have shown interest in utilizing Commercial Off-The-Shelf (COTS) due to their benefits related to low power consumption and cost, as well as high performance [4]. However, the reliability of COTS microprocessors cannot be taken for granted and must be thoroughly studied [5], [6], [7].

One approach to address this challenge is to explore high-end COTS microprocessor architectures to propose reliable solutions that meet the new requirements for high computational capability. Several hardening architectures based on temporal or spatial redundancy can be found in the literature [8]. Another approach is to use lockstep microprocessors that contain many microsynchronized cores, and any discrepancy between the cores indicates an error. This approach requires specific hardware support, which may not be present in all architectures. ARM Cortex-R microprocessors, such as the Cortex-R5 [9], are examples of microprocessors that can support this lockstep behavior.

In order to explore solutions that are not limited to microprocessors with lockstep architectural support, we proposed in [10] a multiprocessor hardened system with self-recovery capabilities named Macrosynchronized Lockstep (MSLS).

The associate editor coordinating the review of this manuscript and approving it for publication was Harikrishnan Ramiah.

The approach effectively combines several techniques that can detect errors and implement various levels of recovery actions for optimizing availability without external intervention. The proposed approach is flexible, providing multiple levels of rollback capabilities.

This paper presents a Supervised Triple Macrosynchronized Lockstep (STMLS) architecture, which focuses on enhancing system reliability for multicore processors. The proposed approach utilizes four cores, with one core supervising the execution of the three redundant cores working in macrosynchronized lockstep mode. Multiple recovery and error correction mechanisms are implemented to ensure robustness. In addition to a 100% error coverage, the proposed hardened system achieves an improvement in the error correction capabilities, thanks in part to the utilization of an additional macrosynchronized core in lockstep mode and the roll-forward process, reaching a 86.40% compared to the 51.4% and 65.87% obtained in [10] and [11], respectively.

This paper is organized as follows. Section II summarizes related work in the field. Section III details the proposed hardened system and a case study using a quad-core ARM Cortex-A53 processor. In Section IV, the methodology to validate the proposed approach is detailed. Next, in Section V, the experimental results for the fault injection campaign are presented and discussed. Finally, Section VI summarizes the conclusions of this work.

II. RELATED WORK

There are several approaches to enhance microprocessor reliability [12]. Typically, these approaches face challenges such as a lack of information about circuit implementation and limited observation points for microprocessor behavior.

Redundancy is a common approach for enhancing the reliability of microprocessor systems. Temporal redundancy involves executing the algorithm multiple times on the same microprocessor to compare the results. In contrast, spatial redundancy entails using multiple microprocessors or cores to execute the same algorithm or some parts of it and compare the outputs. System reliability can also be improved by employing different microprocessor architectures or algorithm implementations. For instance, Dual Multiplexed in Time (DMT) and Dual duplex Tolerant to transients (DT2) are examples of successful redundancy techniques implemented and tested in space missions by the French Space Agency (Centre National d'Études Spatiales, CNES) [13]. These techniques utilize temporal and spatial redundancy, respectively, to harden microprocessors against failures.

An effective solution based on spatial redundancy is using redundant Central Processing Units (CPU). There are options available where these redundant CPUs are integrated on a single chip, and the outputs of all CPUs are compared every clock cycle to detect any discrepancies. ARM Cortex-R [9] microprocessors feature built-in hardware support for microsynchronization, also known as lockstep. The ARM Cortex-R5 can operate in a Dual-Core Lockstep (DCLS) configuration where both cores share inputs and caches, or in split mode where each core runs a different software application. The ARM Cortex-R5 is equipped with hardware mechanisms such as parity checking and Error-Correcting Code (ECC) to handle soft and hard errors, making it suitable for safety-critical applications. However, since there are only two CPUs, majority voting cannot be performed, allowing for error detection but not correction. Another example of a lockstep built-in feature is the Texas Hercules microprocessor [14], designed to enhance microprocessor reliability.

In the literature, there are alternative approaches for implementing DCLS in processors that lack this built-in feature. For instance, in [15], DCLS is successfully implemented using an ARM Cortex-A9 microprocessor and tested with heavy ions, resulting in a significant reduction in cross-section by one order of magnitude. Another hybrid technique that combines a dual-core microprocessor with thread replication and a trace Intellectual Property (IP) observer is proposed and tested with protons in [16]. Both approaches rely on microsynchronization to achieve system reliability. Additionally, in [10], we proposed a macrosynchronization approach to design a hardened microprocessor system with a dual-core Cortex-A9, resulting in error coverage of up to 99.3% in irradiation experiments and a reduction in cross-section by up to two orders of magnitude.

A third redundant CPU could be added to the DCLS system to enhance reliability and allow for error correction capability. In [17], Iturbe et al. introduced a Triple Core Lockstep (TCLS) architecture using the ARM Cortex-R5 microprocessor, discussed in more detail in [18]. TCLS provides the capability of rapid, automatic, and transparent error recovery within microseconds, a feature that is not present in DCLS. According to the authors, in a representative telecom satellite operating in a Low Earth Orbit (LEO) for a 10-year mission, the TCLS processor offers a significant advantage over COTS Cortex-R5 DCLS processors. It can detect errors in the CPUs and swiftly recover from them, typically within microseconds. This remarkable capability results in a dramatic reduction in system downtime, up to 1000 times less compared to the COTS Cortex-R5 DCLS processors. Although this proposal constitutes an excellent solution, to date and to the best of our knowledge, it is not offered commercially, and only a proof-of-concept TCLS chip using 32-nm Complementary Metal-Oxide Semiconductor (CMOS) Low-Power process technology has been implemented and evaluated [19].

In another study, a dual-core Cortex-A9 was combined with a MicroBlaze soft core to implement a TCLS with diverse microprocessor architectures, incorporating rollback and roll-forward capabilities [20]. The authors propose the TCLS approach to enhance dependability in a system consisting of two ARM cores in the Processing System (PS) and one MicroBlaze core implemented in the Programmable Logic (PL) of the Field-Programmable Gate Array (FPGA). To protect the MicroBlaze core against soft errors, a Triple Modular Redundancy (TMR) scheme is employed. The goal is to achieve replicated execution of the same application in a lockstep manner while ensuring consistency through a checker

module that monitors and verifies the outputs of the ARM cores. The novelty of this research lies in the introduction of a MicroBlaze core in a lockstep-based methodology, combined with hard-core ARM processors, to support both rollback and roll-forward recovery. However, it is important to note that this specific TCLS implementation may not be feasible for all systems due to potential constraints in size, weight, and cost, as it requires the use of a soft core.

In this paper, we present a Supervised Triple Macrosynchronized Lockstep (STMLS) architecture for multicore processors. The proposal implements several recovery and error correction mechanisms. In contrast to the proposed approach presented in [10] focused on dual-core processors, this approach utilizes four cores to harden the system and incorporates the roll-forward process. One core acts as the primary core, and the other three cores are considered secondary cores. The secondary cores run the same software application that needs to be hardened and are kept in macrosynchronization, controlled by the primary core. Using three cores for system hardening, based on Triple Modular Redundancy (TMR), allows for identifying the core where the failure occurred and correcting the error through the roll-forward mechanism. In the roll-forward, the error-free saved status of the non-erroneous secondary cores is utilized to overwrite the faulty state of the affected core. During this process, specific register values are adjusted to match the program memory region of the target core. In the event of an unsuccessful roll-forward, a rollback process can be performed. The proposed approach is validated through a fault injection campaign, demonstrating a 100% error coverage and system recovery with error correction up to 86.40%.

III. SUPERVISED TRIPLE MACROSYNCHRONIZED LOCKSTEP (STMLS)

STMLS utilizes triple spatial redundancy to harden a microprocessor system. The tripling of cores, as in a typical TMR system, enables the STMLS to perform majority voting. With this approach, the hardened system is able to detect errors in one or multiple cores and execute the appropriate implemented recovery mechanism to correct errors. The general architecture and the system design are detailed in the following subsections.

A. SYSTEM ARCHITECTURE

STMLS utilizes all four cores of a quad-core processor, with one core acting as the primary core and the other three considered as secondary cores. The architecture also comprises safe memories and bidirectional communication mechanisms (modules) between the primary core and the secondary cores. In Fig. 1, a simplified block diagram of the architecture of the proposed approach is presented.

Each core has its safe memory where the data to be used in the proposed recovery mechanisms are stored. Safe memories are memory regions embedded into the system or external to it, preferably independent from one another, and equipped with data protection mechanisms against errors. The safe memory

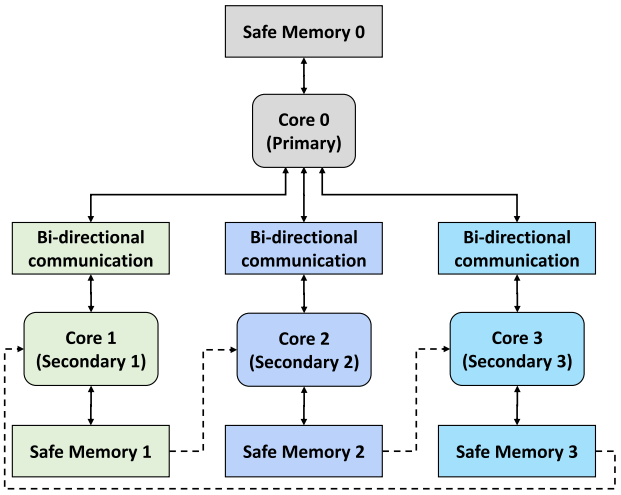


FIGURE 1. Simplified block diagram of the proposed approach.

is accessed by its respective core to store (context saving) or restore (rollback or roll-forward) the processor's status data when needed. Both read and write operations are performed in safe memories: the context-saving process requires writing to the safe memory, whereas the rollback and roll-forward process requires reading from the safe memory (although it also entails overwriting the processor's memory). In the roll-forward process, the safe memory is accessed by another specific core, as shown in Fig. 1 with dashed lines, for memory read operation. The context-saving, rollback, and roll-forward processes will be explained in subsection III-B.

Different approaches can be used for safe memory implementation depending on the availability of the selected architecture. Special attention must be paid to the speed and size of the memories, the availability of single or separate interfaces for read/write operations, and Error Detection and Correction (EDAC) capabilities. Data size can be the most problematic issue and can limit the memory selection options, especially for software applications that occupy a large amount of memory. The amount of data to be stored for system recovery determines the level of reliability, but there is always a memory limit in a system architecture. In such a case, the usual choice is to limit the amount of data to be protected, i.e., the replication sphere [21]. The concept of the replication sphere, also used in [22], [23], [24], and [25], encompasses both the physical redundancy found in a lockstepped system and the logical redundancy inherent in a Simultaneous and Redundantly Threaded (SRT) processor. Components operating within the replication sphere benefit from fault tolerance through replicated execution, while those outside do not and thus necessitate alternative techniques. Accurately determining a system's replication sphere helps implement an appropriate but not excessive set of replication and comparison mechanisms. Adjusting the size of the replication sphere is a trade-off between reliability (protected data) and overheads in terms of performance and area (in this case, occupied memory). EDAC capabilities are

also fundamental factors as they ensure data integrity in the event of errors in this part of the architecture. Both internal and external memories can be used to implement safe memories. Opting for internal memory is often preferred to avoid adding external components and the performance degradation that can affect the system with external memory connections. However, one disadvantage is that internal memory cannot be directly protected against soft errors if the memory does not provide this kind of protection. An external memory with additional protection against soft errors can be used. In the context of System on Programmable Chip (SoPC), it is possible to implement custom-protected memory blocks within the Programmable Logic (PL), incorporating various techniques such as ECC and scrubbing to mitigate the impact of soft errors.

To facilitate data sharing between the primary core and the secondary cores, a mechanism must be established. The bidirectional communication mechanism is utilized to exchange information regarding synchronization, and additional information is used to compare the current status of the secondary cores. Different strategies can be used depending on the architectural choices available. An option to consider is a shared (internal or external) memory. In System-on-Chips (SoCs), On-Chip Memory (OCM) can be used, which offers the advantages of being integrated into the system and designed to provide fast and efficient access. A shared memory strategy can be combined with Software Generated Interrupts (SGI) to notify when the information is available. Customized alternatives can also be used, such as the implementation in PL of modules designed specifically for this purpose.

B. SOFTWARE DESIGN

Control tasks of the hardening technique are performed on the primary core. The control tasks include synchronization, majority voting, triggering the recovery mechanisms, or triggering the processor context-saving process. The processor context saving is the basis of two of the three available recovery mechanisms in the proposed hardening technique. It consists in storing in the safe memory all the relevant information that defines the state of the processor. This relevant information may include the content of cache memories (or part of it), register file, variables, etc. The recovery mechanisms utilize an error-free saved context to overwrite the current erroneous context and return to a previous execution point in the software application. Achieving a successful return of the processor to a previous execution point requires storing the processor context using as much information as possible. The processor context can include both local and global variables, and the more information is stored, the more comprehensive the processor status will be. This also results in a higher reliability, but it will increase performance and memory overheads. For this reason, if the amount of data to store is very large, it is necessary to decide which variables are critical for the execution of the software application and store only those variables to reduce overheads but at the expense of losing reliability. As a first approach, we have selected to protect all local variables and the register file.

During function calls, local variables are stored on the stack for context switching, along with the register file. Therefore, the stack can be utilized for the context-saving process. The stack consists of multiple frames, with each software application function having its unique frame to store its context. The Frame Pointer (FP) register points to the start of the frame, whereas the Stack Pointer (SP) register points to the last element of the frame. By using the FP and SP special registers as references, it is possible to copy the program stack partially or entirely. This enables the creation of copies for all local variables or a subset of variables belonging to specific software application functions. We have implemented the context-saving process through an SGI. When the Interrupt Service Routine (ISR) is called, the microprocessor context is automatically stored in the stack. Then, the stack (containing all local variables and register file), as well as special registers, are stored in the safe memory. When the context-saving process is complete, the application returns from the ISR to the main program function.

As we mentioned before, the primary core oversees the control tasks of the STMLS. A software application without reliability requirements can be executed on the primary core besides the STMLS control tasks since this core is not designed to offer reliability in the software application it runs (no redundancy has been implemented).

The secondary cores are macrosynchronized cores controlled by the primary core. The same software application runs on each of the secondary cores. The software application is divided into several blocks, and several verification points (VP) are defined along the entire application. The VP is used as a stopping point to keep the macrosynchronization between the secondary cores and to perform the majority voting process by the primary core. Comparing all relevant data included in the processor context between each core to determine if a mismatch has occurred is not recommended due to the significant performance overhead it implies. A better practice is to generate a signature that involves the parameters we want to compare. Fig. 2 shows the stages of execution in the system considering both primary and secondary cores and the usage of the signature.

The arrow in Fig. 2 indicates signaling sent using the bidirectional communication mechanism. When a VP is reached, a signature involving the most significant parameters of the processor state, such as the status register and application software variables, is computed on each secondary core. The signatures are sent to the primary core for majority voting and mismatch detection. Signature sending is carried out through the bidirectional communication block, and interrupts are used to notify the primary core of the availability of the signatures. If an error is detected (signatures are different) through the voting process, the appropriate recovery mechanism will be performed. Otherwise (signatures are identical), the context of each core is stored in its corresponding safe memory. Once the action triggered by

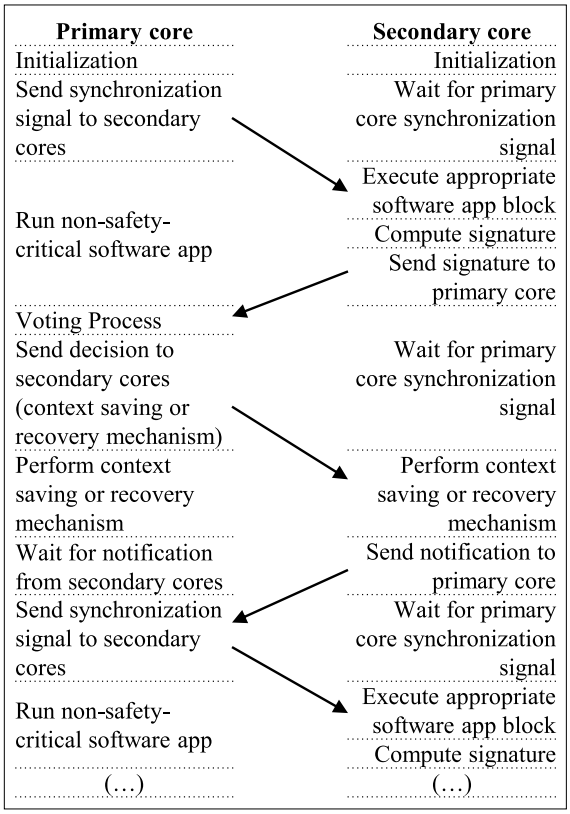


FIGURE 2. Stages of execution in the system considering the primary core and a secondary core.

the majority voting decision (processor context storage or recovery mechanism) has been completed in each secondary core, the primary core is notified. If the primary core has also finished storing the processor context or performing the recovery mechanism, it sends a notification to the secondary cores to continue with the execution.

In our proposed hardening technique, multiple strategies are implemented as recovery mechanisms when errors are detected: rollback, roll-forward, and restart. All these mechanisms achieve system recovery; however, it is essential to make the following clarification. Rollback and roll-forward mechanisms are error recovery and correction mechanisms, meaning they recover the system by correcting the produced error. On the other hand, the restart mechanism recovers the system, not by error correction, but by performing a reboot. Restart can be performed by software (software restart) or when a watchdog timer reaches zero (watchdog timer restart). The software restart is performed by writing the adequate system control register and has a similar effect to a Power-On reset. The watchdog timer is used to detect and recover from system malfunctions. It is commonly used to prevent system lockup, such as when the software becomes stuck in a deadlock. The watchdog timer is restarted by the primary core in the verification process, and the timeout period can be easily selected by the user regarding the specific needs of each application.

The rollback process restores an error-free processor context that was previously stored. It is performed simultaneously in all three secondary cores when their outputs (signatures) are all different from each other, and thus, there is no majority in the voting process. The maximum number of consecutive rollbacks is defined as rollback depth, and it can be set depending on the total available safe memory for context saving, as well as the specific characteristics of the software application. When a first attempt at rollback fails, a consecutive second attempt is made, and so on, until the rollback depth is reached. Once the rollback depth is reached without success, a software restart is performed.

The roll-forward process is performed when the signatures of two of the secondary cores match but differ from the signature of the remaining secondary core. To overwrite the erroneous state, the current error-free processor context of one of the other two secondary cores is used. The values of specific registers are modified in the process to match the program memory region of the target core. If the roll-forward is unsuccessful, consecutive rollbacks are performed as many times as needed until a successful rollback is achieved or the rollback depth is reached. Once the rollback depth is reached, a software restart is performed. The roll-forward and rollback processes are done by using SGI.

In scenarios requiring high reliability, an unresponsive state of the processor resulting from radiation-induced exceptions is undesirable [26]. To overcome this situation, exception recovery is also implemented through a rollback mechanism. When an exception occurs, some elements related to the processor state must be modified to effectively return from the exception through a recovery mechanism. Status-related register fields that are checked and modified if needed when an exception is triggered can encompass execution state controls, exception mask bits, access control bits, etc. After modifying status-related register elements, a rollback is performed, and the execution continues from a previous error-free execution point. If an exception occurs in the primary core, the rollback process is performed on this core but not the secondary cores; otherwise, the primary core performs a rollback in all four cores (secondary cores and primary core itself). When recovering from exceptions, the rollback depth can also be reached if the rollbacks were unsuccessful. In this case, a software restart is carried out.

All recovery mechanisms (rollback, roll-forward, and restart) have been implemented in all secondary cores. However, the proposed approach does not use redundancy in the primary core. That means that in the event of a data error in this core, the proposed approach is not able to detect it. Despite that, we have implemented some mechanisms to ensure robustness against failures that may cause a hang in the primary core. Consequently, the primary core has the following two peculiarities: roll-forward is not performed in the primary core, and rollbacks performed in this core are due to the recovery process of exceptions. An interesting feature to consider for future improvement is using temporal redundancy in the primary core to provide robustness against data errors.

C. CASE STUDY: QUAD-CORE ARM CORTEX-A53 IMPLEMENTATION

As a case study, we have selected the quad-core ARM Cortex-A53 [27]. In this subsection, the main characteristics of the ARM Cortex-A53 processor are presented, as well as the implementation of STMLS in this processor.

1) ARM CORTEX-A53 PROCESSOR

The Cortex-A53 processor is a mid-range processor known for its low power consumption [27]. It features up to four cores, each with its own separate L1 cache memory for instructions and data, with sizes of 8, 16, 32, or 64 kB, and a single shared L2 cache with sizes of 128, 256, 512, 1024, or 2048 kB. It implements the Armv8-A architecture [28], supporting both AArch32 and AArch64 execution states. It supports several exception levels in each execution state and for A32, T32, and A64 instruction sets. The Cortex-A53 processor is designed to support various advanced features, including Single Instruction Multiple Data (SIMD) and floating-point extension for efficient integer and floating-point vector operations. Additionally, it also includes the Armv8 Cryptography Extensions to enhance its cryptographic capabilities. Fig. 3 shows the main features of the ARM Cortex-A53 processor architecture.

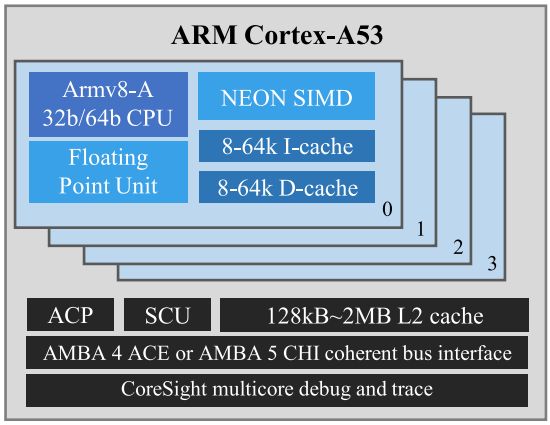


FIGURE 3. Simplified ARM Cortex-A53 processor architecture.

The ARMv8 exception model defines multiple exception levels, namely EL0-EL3, each with different levels of software execution privilege. EL0 is the lowest level, also known as the unprivileged execution level. As the exception level increases from 1 to 3, the privilege level of the software execution also increases. EL2 is designed for processor virtualization, whereas EL3 supports the secure state. The Cortex-A53 processor implements all exception levels, EL0-EL3, and supports both AArch64 and AArch32 execution states at each level.

The execution state of the processor encompasses various aspects, such as supported register widths, instruction sets, and key elements of the execution model, Virtual Memory System Architecture (VMSA), and the programmer's model. The two execution states are AArch64 and AArch32, which define the processor's execution environment and capabilities.

AArch64 is the 64-bit execution state, which encompasses several key features. It includes 31 64-bit general-purpose registers (X0-X30), along with 64-bit Program Counter (PC), Stack Pointer (SP), and Exception Link Registers (ELRs). It should be highlighted that the X30 register is the Link Register (LR) and is used to store the return address after a subroutine or function call. On the other hand, the ELR is specifically used to store the return address after an exception or interrupt occurs.

AArch64 provides a single instruction set known as A64 and defines the ARMv8 exception model mentioned before. An additional aspect is the introduction of the Process State (PSTATE) register. PSTATE is an abstraction for process state information, encompassing fields that contain relevant information exclusively in the AArch32 state, fields that contain relevant information exclusively in the AArch64 state, and significant fields in both execution states. The PSTATE register is similar to the Current Program Status Register (CPSR) in ARMv7 [29]. Both PSTATE in ARMv8 and CPSR in ARMv7 are status registers that store flags regarding the current state of the processor. Whereas there are specific differences in the flags and supported functionalities of these registers in their respective architectures, their fundamental purpose remains: to provide information about the processor's state and enable instructions and operations to be executed correctly based on that state. The A64 instruction set includes instructions to operate on these PSTATE elements.

2) ARM CORTEX-A53 BASED IMPLEMENTATION

We have selected the quad-core ARM Cortex-A53 processor as a case study. We have used the Zynq UltraScale+ Multiprocessor System-on-Chip (MPSoC) ZU3EG A484 device [30], which contains a Cortex-A53 processor, a dual-core ARM Cortex-R5F processor [9], an ARM Mali-400MP2 Graphics Processing Unit (GPU), and a 16 nm FinFET+ Programmable Logic (PL). The system is divided into two main parts: the PS (Processing System) and the PL. The PS contains, among others, the Application Processing Unit (APU) with the quad-core ARM Cortex-A53. The PL can accommodate various hardware designs based on the application requirements and available resources. The selected device contains 154K System Logic Cells, 141K Configurable Logic Block (CLB) Flip-Flop (FF), 71K CLB Lookup Tables (LUTs), 360 Digital Signal Processing (DSP) slices, and Block Random Access Memory (BRAM) that make a total of 7.6 MB. Only the ARM Cortex-A53 processor and the PL are used to implement the STMLS. A simplified block diagram of the Zynq UltraScale+ MPSoC ZU3EG A484 device is shown in Fig. 4.

The safe memories and the bidirectional communication blocks of the proposed STMLS are located in the PL. Safe memories are implemented in BRAM using Block Memory Generator IP [31] and Advanced eXtensible Interface (AXI)

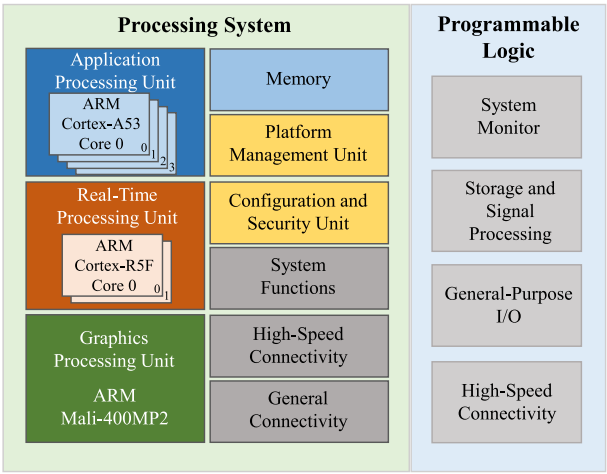


FIGURE 4. Simplified block diagram of the Zynq UltraScale+ MPSoC ZU3EG A484.

BRAM Controller IP [32] core from AMD. The Block Memory Generator core leverages embedded block memory primitives in AMD Field-Programmable Gate Array (FPGA) to enhance the functionality and capacity of a single primitive, allowing for memories of varying widths and depths. This core has two fully independent ports that enable access to a shared memory space. Ports A and B have separate interfaces for both write and read operations. In cases where not all four interfaces are required, it is possible to select a simplified memory configuration, such as a Single-Port Memory or Simple Dual-Port Memory, to reduce FPGA resource consumption. Key features of the Block Memory Generator core include optimized algorithms that minimize block RAM resource usage or optimize for low power consumption, as well as the ability to configure memory initialization according to specific requirements. In our implementation, the Block Memory Generator IP is utilized with the AXI BRAM Controller IP. The AXI BRAM Controller core offers flexible configuration options for utilizing the BRAM block. It can be set up to use a single or both ports to access the BRAM block in either an AXI4 or AXI4-Lite controller configuration. Additionally, the AXI BRAM Controller IP supports ECC functionality on the datapath. ECC allows an AXI master to detect and correct single-bit errors and detect double-bit errors in the BRAM block. The ECC functionality can be enabled regardless of whether the BRAM access is single port or dual port. If ECC will be used, it must be enabled during the design stage. ECC feature is only available when the BRAM block is configured with a data width of 32, 64, or 128 bits, offering both Hamming code and HSIAO [33] algorithms for 32 and 64 bits, whereas only the HSIAO algorithm is available for 128-bit configurations. To implement the STMLS safe memories, each Block Memory Generator has been configured in BRAM controller mode, with a size of 64 kB, True Dual Port RAM mode, and ECC enabled using HSIAO.

For bidirectional communication between the primary core and the secondary cores, AMD Mailbox IP [34] core is used. The Mailbox core facilitates bidirectional communication between processors or cores in the same processor and serves as a connection point between separate systems. Apart from facilitating data transmission, the Mailbox core also supports the generation of interrupts between the systems. The Mailbox core features two bus interfaces that provide access to internal resources. Each interface can be individually configured to utilize either an AXI4-Lite or AXI4-Stream interface. Three mailboxes have been implemented with the same configuration, allowing for the use of one mailbox for communication between each secondary core and the primary core. For both port 0 and port 1, the AXI4-Lite interface is utilized. To implement the First In First Out (FIFO) memory of the mailbox, the distributed Random Access Memory (RAM) type has been chosen. The depth of the FIFO in the Mailbox, which can range from 16 to 8192, is set to 16. Increasing the depth of the FIFO in an FPGA results in higher resource consumption, primarily in terms of memory block and logic utilization. Due to the relatively small amount of information to be exchanged between secondary cores and the primary core, a size of 16 has been selected. This size not only meets the requirements of the hardening technique but also avoids excessive use of memory and programmable logic, as well as an increase in latency.

There are multiple types of connections between the PL and the device. Each connection has significant characteristics that can impact the design. The connection types include AXI interfaces, interrupts, clocks, dedicated streams, etc. The primary mechanism for communication between components in the Zynq UltraScale+ MPSoC device is the ARM AXI interconnect. Numerous AXI links between the PS and the PL are available. Selecting the appropriate AXI link for communication between the PS and PL is one of the crucial decisions when developing applications for the Zynq UltraScale+ MPSoC device. The selected interfaces for connecting the PS with the modules implemented in the PL are M_AXI_HPM0_FPD and M_AXI_HPM1_FPD. Since more than two modules need to be connected, the AXI SmartConnect IP [35] core from AMD is used. AXI SmartConnect is a versatile solution commonly used in systems that employ AXI memory-mapped transfers. Each instance of AXI SmartConnect supports up to sixteen Slave Interfaces (SI) and up to sixteen Master Interfaces (MI). It is fully compliant with the AXI protocol, allowing each SI and MI to be connected to a master or slave IP interface of type AXI3, AXI4, or AXI4-Lite. Many interface data widths are supported, and transactions between interfaces with different data widths are automatically converted by SmartConnect. The address width can be up to 64 bits. In the proposed design, two slave interfaces (S00_AXI and S01_AXI, *S* from *Slave*) and fourteen master interfaces (M00_AXI to M13_AXI, *M* from *Master*) are used in the AXI SmartConnect IP. The slave interfaces of the AXI SmartConnect IP, S00_AXI and S01_AXI, are connected to the two high-performance AXI interfaces provided by AMD to push a large amount of data from the PS to the PL: Master AXI High-Performance

Full-Power Domain Interface 0 (M_AXI_HPM0_FPD) and Master AXI High-Performance Full-Power Domain Interface 1 (M_AXI_HPM1_FPD) [30]. Full-power domain (FPD) refers to one of the four power domains: Low-power domain (LPD), Full-power domain (FPD), PL power domain (PLPD), and Battery power domain (BPD), each of which can be isolated independently. The PS considers the Real-time Processing Unit (RPU) and APU MPCores as two separate power domains: the low-power domain (LPD) and the full-power domain (FPD), respectively. As our focus is on the APU with the ARM Cortex-A53, we utilize the master interfaces of the FPD.

The blocks implemented in the PL do not occupy all the available resources in the Zynq UltraScale+ device. When the PS attempts to access an unused area, it can result in a hang. To address this behavior, we have modified the attributes of the device's translation table, protecting the unused memory regions. This allows for exceptions to be generated when attempting to access one of these regions from the PS, enabling the exception recovery mechanism.

In Fig. 5, a simplified block design of the proposed STMLS in the Zynq Ultrascale+ MPSoC ZU3EG A484 is shown. The design includes the four Safe Memories (each Safe Memory consists of a BRAM Memory and an AXI BRAM Controller), the three Mailboxes used for bidirectional communication between the secondary cores and the primary core, and the AXI SmartConnect for connecting all the implemented blocks in the PL with the PS. This design represents a possible implementation and interconnection of the blocks required in the STMLS architecture. Other designs and interconnections can be implemented depending on the platform used.

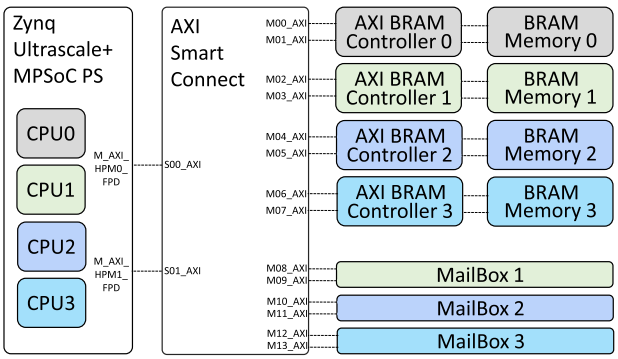


FIGURE 5. Simplified block design of the proposed STMLS in the Zynq Ultrascale+ MPSoC ZU3EG A484.

An overview of the PL resources utilized is presented in Table 1. The following resources are reported in the table: Lookup Tables (LUT), Lookup Table Random Access Memory (LUTRAM), Flip-Flops (FF), Block Random Access Memory (BRAM), and Buffer Gates (BUFG).

The recovery mechanisms of the proposed STMLS have been effectively implemented in the ARM Cortex-A53 using the AArch64 execution state (64-bit) and considering its specific features. The exception level EL3 is used to provide

TABLE 1. Summary of resource utilization in PL.

Resource	Available	Utilization	Utilization (%)
LUT	70 560	29 445	41.73
LUTRAM	28 800	6 288	21.83
FF	141 120	35 728	25.32
BRAM	216	80	37.04
BUFG	196	1	0.51

the highest privilege level in executing the STMLS control tasks. For the implementation of rollback and roll-forward, no specific considerations need to be taken into account, except for exception recovery. The PSTATE fields must be modified to return from exceptions. The PSTATE fields that are verified and, if necessary, modified when an exception is triggered include the execution state controls, exception mask bits, and access control bits. In the context-saving process, we have included the entire register file of the 64-bit architecture (X0-X30 general-purpose registers and special registers). The restart mechanism performs not only a PS reset but also a reset and reprogramming of the PL. The software restarts are performed by accessing and writing the reset control (RESET_CTRL) register of the Zynq. This is an internal reset with a system-level software restart effect. To implement watchdog timer restart functionality, one of the three system watchdog timer (SWDT) units in the PS can be utilized. These watchdog timers are all built upon the ARM system watchdog timer architecture and have the same programming model and similar control registers. We have used the Full-Power Domain (FPD) watchdog timer, FPD_SWDT, that protects the Application Processing Unit (APU) Multiprocessor core (MPCore) and its interconnect.

IV. METHODOLOGY FOR EVALUATING THE PROPOSED APPROACH

A. PLATFORM SELECTION FOR TESTING

In order to assess the effectiveness of the proposed hardening approach, we utilized an Ultra96-v2 development board [36] from Avnet. The Ultra96-v2 contains a Zynq UltraScale+ MPSoC ZU3EG A484 [30]. The board features 2 GB of DDR4 memory and a microSD card slot. The Ultra96-v2 includes various connectivity options, such as Gigabit Ethernet, Wi-Fi, and Bluetooth. It also provides USB 3.0 ports and Mini DisplayPort for connecting peripherals and displays. The board offers multiple expansion interfaces, including Pmod connectors, allowing users to connect a wide range of expansion boards and peripherals. The Ultra96-v2 supports the Vitis software development platform from AMD [37], enabling software developers to leverage the power of the FPGA fabric alongside the processing cores. It also supports various operating systems, such as Linux, providing a familiar development environment.

This board has been chosen to validate our proposed hardening technique because it constitutes a flexible heterogeneous

computing platform for prototyping and developing advanced applications easily. By utilizing a single microUSB connection, the Ultra96 USB-to-JTAG/UART Pod enables communication with both the UART and JTAG headers on the Ultra96 board. The Ultra96 USB-to-JTAG/UART Pod is a programming and debugging device that offers a cost-effective and convenient solution for incorporating USB-to-UART and AMD USB-to-JTAG functionality. This gives engineers access to a serial terminal and AMD JTAG tools, facilitating tasks such as communication, debugging, and simulation.

B. REGISTER FILE INJECTOR

To validate the proposed technique, we implemented a software-based fault injector considering the one used in [38], and we conducted an extensive fault injection campaign. The injector is implemented in core 0 (primary core) of the ARM Cortex-A53 processor.

The injector randomly produces a bit-flip in one of the selected registers of the register file, in one of the four cores of the quad-core ARM Cortex-A53 processor at a random time instant. The selected registers for injection are all the X0-X30 general-purpose registers, the ELR, the Control Processor Trace Register (CPTR), and the PSTATE register. During the experiments, we collect information about fault injection and errors for later analysis, as well as application execution and processor state-related information. To this purpose, we connected the Ultra96 board to a computer through a serial connection. This information allows us to detect and classify events using specific identification codes.

C. BENCHMARKS

In order to assess the reliability of microprocessors in radiation environments, a variety of benchmarks are commonly employed [39]. For the injection experiments, we have used three different benchmarks: Matrix Multiplication (Mmult) algorithm, Advanced Encryption Standard (AES) algorithm [40], and NIR HAWAII-2RG application [41].

The Mmult benchmark consists of several matrix multiplications, with 20×20 matrices of 16-bit integers, whereas the AES algorithm encrypts a data value of 16 bytes with a key length of 128 bits. In both benchmarks, the software application is partitioned into 12 blocks of code, with a verification point (VP) established after each block. An additional VP is inserted at the beginning of the software application, resulting in a total of 13 VPs. For both algorithms, 10 000 iterations are performed in each block of code to increase the processing time of each block and ensure that a high percentage of total faults are injected into the piece of code related to the software application.

The NIR HAWAII-2RG application [41], provided by the European Space Agency (ESA), serves as a benchmark for processing images captured by the Teledyne Near InfraRed HAWAII detector [42]. It offers synthetic image generation capabilities, eliminating the need for a physical sensor during testing. This benchmark is designed to generate a high computational load by utilizing a large amount of data. It has been widely used to evaluate the performance impact of different processors when considering various pre-processing steps. The Near InfraRed HAWAII detector requires multiple readouts to obtain a final pre-processed image for compression. Each readout or frame from the detector consists of a 2048×2048 pixel size. Multiple frames are combined to form a group, and an exposure can comprise several groups. The number of frames and groups can be configured. The algorithm encompasses several blocks or steps, as shown in Fig. 6. Firstly, the Saturation detection block identifies pixels that have reached saturation, halting the flux estimation in subsequent steps. Next, the Co-adding step reduces readout noise by summing the frames within each group (typically 1-8 frames). The Super-bias subtraction block removes pixel-to-pixel offset variations by subtracting a bias frame from the frame obtained by the detector. Following that, the Non-linearity correction step corrects for detector non-linearity, whereas the Reference pixel subtraction step eliminates common noise. In the final two steps of the algorithm, Cosmic ray detection and Linear least square fit, the flux is estimated, and disturbances caused by cosmic rays are detected.

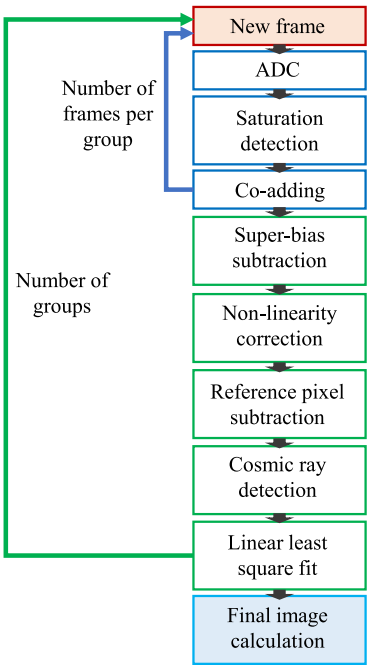


FIGURE 6. Steps of NIR HAWAII-2RG algorithm.

Each step of the algorithm is processed as an individual block of code, and a VP has been established between each algorithm step, resulting in 8 VPs. In addition to these 8 VPs, we have introduced 5 additional VPs, resulting in a total of 13 VPs. The first extra VP is positioned at the beginning, preceding the initialization of data. Another VP is placed before the generation of random input data. The third extra VP

is established before the first step of the algorithm. The final two additional VPs are positioned before the copying of data arrays for a new iteration and at the end of the infinite loop. Since the application uses a large amount of data, we have followed the same strategy presented in [11]: not all data are stored in the stack and saved in safe memories. Contrary to the original NIR HAWAII-2RG algorithm, frames and data arrays are not overwritten when processed. In a block of code, a new data array is used to avoid overwriting the array utilized in the previous block. While this allows the data from previous blocks of code to be available for rollbacks and enhances system reliability, it also entails an increase in both time and memory. Therefore, for such data-intensive applications, a trade-off must be established between reliability and the memory and time overhead the system permits.

All benchmarks are compiled without optimization (no optimization flag, -O0), using the GNU C++ ARM cross-compiler for AMD MPSoC devices, targeting the aarch64 architecture without operating system (aarch64-none-elf-gcc). The gcc 10.2.0, among other GNU toolchain components, is installed with the Vitis software platform version 2021.2. For each benchmark, all secondary cores run the same C language software application in an infinite loop. Although the software application is identical, each core has an independent software application located in a distinct memory location, resulting in each core accessing different memory addresses.

V. RESULTS AND DISCUSSION

A. MEMORY AND PERFORMANCE OVERHEAD

Table 2 provides an overview of the memory overhead of the proposed system when considering both unhardened (Unhard.) and hardened (Hard.) versions of the benchmarks. The unhardened version has been implemented on a single core, where the software application is executed. The software application in the unhardened version is the same as that which runs on secondary cores in the hardened version, except for the code related to the STMLS hardening technique. The results show that the proposed technique does not impose significant overhead in terms of program and data size, considering Mmult and AES benchmarks. However, for the NIR Hawaii algorithm, the overhead reaches a value of 2.33, mainly due to the increase in data memory size. As previously stated, frames and data arrays are not overwritten in the hardened version to ensure that data from previous blocks of code remain available in case a rollback is required. Most of the variables used by the algorithm are 2048×2048 matrices of 32 bits integers, and each of these variables occupies 16 MB. Therefore, replicating these variables to ensure high reliability leads to an increase in data memory size. It is also important to note that, in all benchmarks, additional memories (safe memories 0, 1, 2, and 3, each with a size of 64 kB) are added to the hard-core microprocessor, and the execution is carried out simultaneously in three secondary cores.

Table 3 summarizes the performance overhead required for the STMLS. The microprocessor operates at a frequency of 1.2 GHz. The first row of the table shows the additional execution time incurred due to the verification and context-saving processes. This time is added to each block of code in the software application; therefore, the total overhead depends on the number of blocks of code defined by the user. Additionally, the time for the context-saving process depends on the amount of data being protected in the software application.

TABLE 2. Memory overhead.

Benchmark	Core	Unhard.	Hard.	Overhead
Mmult	Primary	63 kB	76 kB	1.21
	Secondary	64 kB	69 kB	1.08
AES	Primary	60 kB	72 kB	1.20
	Secondary	61 kB	66 kB	1.08
NIR Hawaii	Primary	60 kB	71 kB	1.18
	Secondary	76 MB	177 MB	2.33

TABLE 3. Performance overhead.

Category	Mmult	AES	NIR Hawaii
Verification & context saving (µs)	208	24	-
Verification & rollback (µs)	246	28	-
Verification & roll-forward (µs)	1360	49	-
Execution time - Unhard. (ms)	5920	2331	3975
Execution time - Hard. (ms)	5923	2332	14 946
Software restart (ms)	489	502	497

The second and third rows of Table 3 present the execution time for the verification process plus rollback and roll-forward, respectively. It is observed that the *Verification* & *roll-forward* category requires a time that is approximately 5.53 and 1.75 times the one required by the *Verification* & *rollback* category, considering Mmult and AES, respectively. This is because, in addition to increasing the number of instructions needed for performing roll-forward, it includes the context-saving processes performed by the non-faulty cores, as well as the context-saving process of the faulty core once its context has been overwritten. Considering that in the NIR Hawaii application, the blocks of code are all different, the time required to perform context saving, rollback, and roll-forward processes varies depending on the block of code. Additionally, since the algorithm has been modified to avoid overwriting frames, it would not be convenient to compute average times for context saving, rollback, and roll-forward processes, as they do not include the total time added to the original algorithm. Hence, the fourth and fifth rows in Table 3 show the total execution time of the software application for both unhardened (Unhard.) and hardened (Hard.) versions. It can be observed that whereas the total time overhead in Mmult and AES is practically negligible, in NIR Hawaii, it reaches a value of 3.76, which corresponds to the large amount of data handled in the algorithm. Table 3 also shows the time

required for booting after a software restart. It is important to highlight that the system boot time has not been considered when measuring the execution time of the software application.

B. POWER CONSUMPTION CONSIDERATIONS

Determining power specifications for FPGA and MPSoC is a critical task that needs to be addressed early in the product design cycle, often before the Register Transfer Level (RTL) description is fully developed. To facilitate this process, AMD Vivado Integrated Design Environment (IDE) [43], which is used to design, simulate, and program AMD FPGA devices, provides an estimation of power consumption. The Environment tab for generating the power report offers various user-editable options. These settings can notably impact the overall estimated power. Among the user-editable selections is the process option, which can be chosen as either ''typical'' or ''maximum.'' The default ''typical'' setting provides a closer representation of statistically measured values, but switching to ''maximum'' will adjust the power specification to worst-case values.

The power estimation provided by Vivado constitutes a powerful tool. However, it should be highlighted that this analysis assumes that both the RPU and GPU are powered on. In the design we have implemented for STMLS evaluation, only the APU containing the quad-core ARM Cortex-A53 processor is utilized. Therefore, a more accurate estimation can be obtained if both these elements, RPU and GPU, are considered powered down. Furthermore, it is also assumed that all cores of each processor are active. This results in a less precise estimation for the scenario without STMLS, in which only one core of the ARM Cortex-A53 processor should be active.

In addition to the power report offered by Vivado, AMD provides the Xilinx Power Estimator (XPE) tool [44] that allows fine-tuning the parameters and obtaining a more accurate estimation. It allows the importation of parameters from the Vivado power report but also enables additional adjustments. For instance, it can be chosen whether other PS elements are powered on or down, and even the number of cores utilized in the processors.

To obtain a more accurate estimation of power consumption, the XPE tool has been utilized. First, a power consumption analysis was conducted using Vivado, and then the values provided by the power consumption report were imported into XPE. In XPE, both the RPU and the GPU were powered down, and two scenarios were considered. In the first scenario, the STMLS technique is not used; therefore, only the PS of the device is used, and no blocks are implemented in the PL. Additionally, the first scenario uses a single-core configuration. In the second scenario, a four-core configuration is utilized. In addition to using the PS, the necessary blocks for the STMLS technique are implemented in the PL. Concerning the user-editable options, we chose the ''maximum'' process setting for a worst-case analysis. The power estimation results using XPE are shown in Fig. 7.

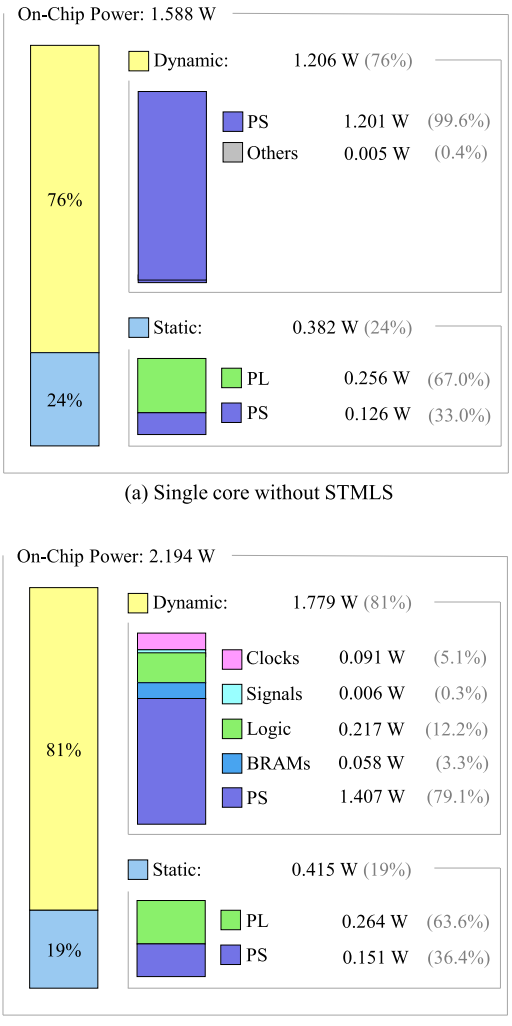


FIGURE 7. Power consumption estimation from Xilinx Power Estimator (XPE).

In Fig. 7(a), the single-core case without STMLS is shown. The overall power consumption is 1.588 W, with 76% attributed to dynamic power and 24% to static power. The PS consumes 99.6% of the dynamic power since no blocks are implemented in the PL. On the other hand, Fig. 7(b) presents data for the scenario employing four cores with STMLS. In this case, the total consumption is 2.194 W, marking an increase of 0.606 W (38.2%) in comparison to the design without STMLS. Both dynamic and static power consumption of the PS increases due to the usage of additional cores in the ARM Cortex-A53 processor (four-core configuration instead of single-core configuration): in the dynamic power of the PS, there is an increase of 0.206 W (17.2%, from 1.201 W to 1.407 W), while in the static power, there is an increase of 0.025 W (19.8%, from 0.126 W to 0.151 W). Furthermore, using BRAM and other PL blocks in the STMLS technique leads to an escalation in total dynamic power from 1.206 W to 1.779 W (47.5%). Regarding the static power, there is an

increase of 0.033 W (8.6%). In conclusion, it can be confirmed that XPE reports more accurate power consumption values, and there is an approximate increase of 38.2% in the total consumption when implementing the STMLS technique.

Today's safety-critical applications, besides reliability and high performance, also require efficient power consumption. This attribute is impacted by factors such as software structure, execution characteristics, and how the code leverages the hardware intricacies of the execution platform. The arrangement of the processor's internal architecture can significantly affect power consumption, depending on how effectively the operations specified in the software application can be matched with the functional components of the processor. Typically, the power dissipation in such systems is viewed as a hardware concern. Nevertheless, it can be efficiently addressed by combining hardware and software approaches to implement power-saving techniques. As recovery mechanisms devised for STMLS entail additional power consumption in contrast to the unhardened single-core version (due to the use of additional software code, hardware implemented in PL, and extra cores), power-saving techniques can be employed to enhance power efficiency during error correction procedures in the proposed hardening strategy. Several studies can be found in the literature in which power-saving techniques are applied on hard or soft core microprocessor-based systems to accomplish an efficient power consumption [45], [46], [47], [48], [49], [50], [51], [52]. In [53], the authors integrate Dynamic Voltage and Frequency Scaling (DVFS) and thread parallelism while also considering various novel metrics to assess the performance and energy consumption of an application. DVFS is an energy-saving method that doesn't necessitate modifying an HW/SW design afterward. This technique modifies the power usage and performance of an embedded device in real time, depending on its workload and operational conditions. The work presented in [54] examines the power monitoring and scaling functionalities of AMD Zynq-7000 SoCs and UltraScale+ MPSoCs, and a real-time operating system (RTOS) is used to administer application resources, voltage/frequency scaling, and power monitoring through its preemptive scheduling policies.

The present work focuses on analyzing the effectiveness of the proposed technique. Although it may impact the power consumption of the entire heterogeneous system, this analysis requires extensive study and consideration of the specific characteristics of the system in which it will be implemented. Therefore, the analysis of the energy efficiency of the system is deferred for future research.

C. FAULTS INJECTION OVERVIEW

The summary of the injection campaign is presented in Table 4. Over 90 000 faults were injected for each benchmark, with the number of faults injected in each of the four cores ranging from 24.79% to 25.30% of the total. An 8.82%, 4.59%, and 5.65% of the total injected faults resulted in errors for Mmult, AES, and NIR Hawaii, respectively, as shown in the last column.

TABLE 4. Summary of injection campaign.

Benchmark	Type of event	Quantity	%
Mmult	Injected faults	96 723	100%
	Observed errors	8010	8.82%
AES	Injected faults	97 566	100%
	Observed errors	4476	4.59%
NIR Hawaii	Injected faults	92 547	100%
	Observed errors	5226	5.65%

TABLE 5. Injections and errors per register for the primary core.

Benchmark	Event	ELR	PSTATE	Other
Mmult	Injected faults	681	648	22 950
	Errors	582 (85.46%)	150 (23.15%)	30 (0.13%)
AES	Injected faults	696	786	22 845
	Errors	567 (81.47%)	156 (19.85%)	51 (0.22%)
NIR Hawaii	Injected faults	624	735	21 699
	Errors	483 (77.40%)	132 (17.96%)	57 (0.26%)

A more detailed analysis is presented considering the injected registers. Since the primary core is unable to detect data errors, the occurrence of errors differs when compared to secondary cores. In this sense, we should analyze separately the data related to the primary core, and the data associated with the rest of the cores (secondaries). Table 5 reports the total injected faults for the most significant registers, as well as the percentage of these injections that caused errors for the primary core, whereas Table 6 shows the registers with the most significant amount of errors for secondary cores.

Both Table 5 and Table 6 display the percentage of errors for each presented register relative to the total faults injected in that register. In both cases, over 75% of the faults injected in the ELR register resulted in an error, as well as approximately 16.67% to 23.15% of faults injected in the PSTATE register. In the case of the X0-X4 registers, errors were only detected in the secondary cores with a significant percentage, unlike the primary core, where data are not protected. X0-X4 registers are the most frequently used general-purpose registers when no optimization is applied (-O0). For this reason, faults injected into these registers are more likely to result in an error. However, differences can be observed in the error rates produced for the same register when comparing the three benchmarks. In Mmult, 8.78% of the faults injected into X4 led to an error, whereas in AES and NIR Hawaii, it was only 0.15% in both cases. Similar behavior can be observed for X2 and X3 registers: in Mmult, the error percentage is higher compared to the values obtained in AES and NIR Hawaii. This is because the utilization of these three registers (X2, X3, X4)

TABLE 6. Injections and errors per register for secondary cores.

Benchmark	Event	X0	X1	X2	X3	X4	X30	ELR	PSTATE	Other
Mmult	Injected faults	2190	2007	2073	2031	2118	2106	2241	2004	55 674
	Errors	1422 (64.93%)	786 (39.16%)	1437 (69.32%)	792 (39.00%)	186 (8.78%)	0 (0%)	2121 (94.65%)	426 (21.26%)	78 (0.14%)
AES	Injected faults	2199	2028	2295	2214	2001	2262	2013	2292	55 935
	Errors	801 (36.43%)	288 (14.20%)	78 (3.40%)	39 (1.76%)	3 (0.15%)	222 (9.81)%	1764 (87.63%)	432 (18.85%)	75 (0.13%)
NIR Hawaii	Injected faults	2022	1965	2088	1950	2031	2013	2079	1998	53 343
	Errors	777 (38.43%)	546 (27.79%)	225 (10.78%)	3 (0.15%)	3 (0.15%)	834 (41.43%)	1788 (86.00%)	333 (16.67%)	45 (0.08%)

is higher in the Mmult benchmark, where they are used as indices in the implemented matrix multiplication loops.

The X30 register is a particular case. This register in ARMv8 is the Link Register (LR), which stores the return address after a subroutine or function call. In the primary core, no specific software application is running. Only tasks related to the STMLS technique are executed. Therefore, there are no subroutine and function calls, and faults injected into the X30 register do not cause errors (the X30 register is not included in Table 5). However, as shown in Table 6, the behavior of X30 in the secondary cores differs for each benchmark. In Mmult, the loops for multiplications are implemented in the main function without using subroutines or functions, resulting in 0% errors in X30. In AES, there is moderate usage of subroutines and functions (9.81% errors in X30). In contrast, in NIR Hawaii, due to the complexity of the application, there is a much higher usage, leading to a higher occurrence of errors in X30 (41.43%).

For the rest of the registers, the percentage of error occurrence is very low (0.13% to 0.26% for the primary core and 0.08% to 0.14% for the secondary cores), since these registers are hardly used without optimization.

D. STMLS EFFECTIVENESS ANALYSIS

In Fig. 8, the results of the effectiveness of the STMLS technique are presented, considering the corrected errors, software restart (SW restart), or watchdog timer (WDT). We can distinguish between two main classifications: recovered errors, when the system is able to recover from the occurrence of an error through any of the recovery mechanisms; and corrected errors, when the proposed technique corrects the error without triggering a restart. It should be highlighted that all observed errors were detected by the proposed technique, achieving a 100% error coverage. Considering the total number of detected errors (8010 in Mmult, 4476 in AES, and 5226 in NIR Hawaii), a 86.40%, 79.96% and 84.50% was corrected in Mmult, AES and NIR Hawaii, respectively. In [10], we obtained a 51.4% error correction rate in the fault injection campaign, whereas in an enhanced MSLS in [11], we obtained 65.87%; therefore, the architecture proposed in this work achieves an improvement of up to 35.0% in error correction. The errors processed by rollback or roll-forward that are not corrected are recovered through a restart.

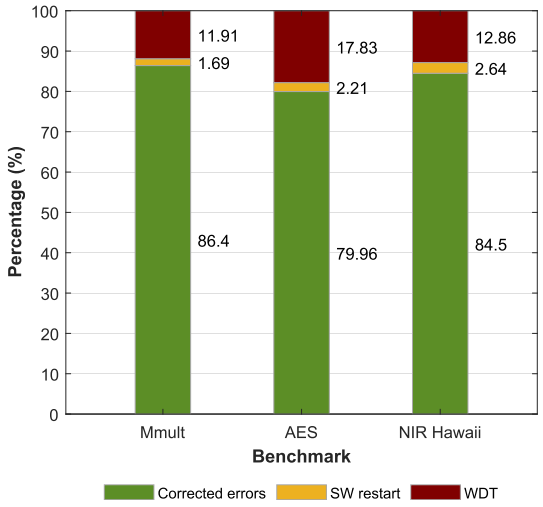


FIGURE 8. Distribution of recovered and corrected errors.

The error correction values achieved by the STMLS technique not only exceed the MSLS results proposed in [10] and [11] but also other approaches available in the literature. In [15], Oliveira et al. propose and evaluate a DLCS technique on the dual-core hard-core ARM Cortex-A9 processor, achieving 33% and 45% masked and mitigated errors, respectively, in the best case of the injection campaign. This results in a recovery percentage of up to 78%. On the other hand, Kasap et al. present a TCLS technique for commercial processors in [20], which uses a dual-core hard-core ARM Cortex-A9 processor and a Microblaze soft-core processor implemented in PL. In addition to evaluating the effectiveness of the proposed TCLS technique, the authors present experimental results corresponding to a version of DCLS. According to the data provided by the authors, in DCLS, 52% of corrected errors and a 21% system restart rate are achieved, resulting in a total recovery percentage of 73%. Regarding the TCLS technique, the authors report a 51.1% corrected error rate and a 21.3% system restart rate for a total recovery rate of 72.4%. In Table 7, a comparison of error correction, system restarts, and total recoveries is presented

for the mentioned techniques, as well as the proposed STMLS. It is worth noting that in the case of the DCLS technique proposed by Oliveira et al. [15], the masked errors reported by the authors do not encompass all the corrected errors. Some corrected errors are classified as mitigated errors; for instance, those corrected soft errors triggered exceptions. However, system recoveries reported by the authors, encompassing both masked and mitigated errors, provide a more accurate comparison value with the rest of the techniques.

TABLE 7. Comparison between different DCLS and TCLS approaches.

Approach	Corrected errors (%)	System restarts (%)	Recoveries (%)
DCLS - Oliveira *et al.* [15]	33.00	45.00	78.00
DCLS - Kasap *et al.* [20]	52.00	21.00	73.00
TCLS - Kasap *et al.* [20]	51.10	21.30	72.40
MSLS - Aviles *et al.* [11]	65.87	34.13	100
STMLS	86.40	13.60	100

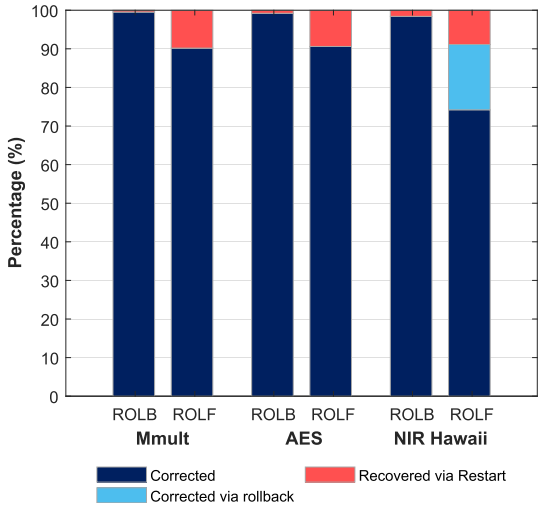


FIGURE 9. Effectiveness of rollback and roll-forward mechanisms for error correction.

Considering the results presented in Fig. 8, corrected errors can be divided into corrected errors through rollback and corrected errors through roll-forward. Out of the total detected errors for each benchmark, 32.32%, 50.20% and 59.07% were corrected by rollback, whereas 54.08%, 29.76% and 25.43% were corrected by roll-forward, for Mmult, AES and NIR Hawaii, respectively, resulting in the percentages of the corrected errors presented in Fig. 8. However, it should be noted that in the case of NIR Hawaii, a peculiarity occurs. Unlike Mmult and AES, in which, when the roll-forward process is performed, either the error is corrected, or a restart is done (SW restart or WDT), in NIR Hawaii there are cases in which a roll-forward process failed, and a subsequent rollback process was triggered ending in a successful error correction. Therefore, out of the 59.07% of errors corrected through rollback in NIR Hawaii, 5.80% corresponds to roll-forward processes that ended with a successful rollback process.

In Fig. 9, the rollbacks (ROLB) and roll-forwards (ROLF) performed in each benchmark are shown to analyze the effectiveness of these mechanisms. Most rollbacks and roll-forwards are successful (errors are corrected), and their high error correction capability is confirmed. Taking into account the rollbacks, they were successful in over 98.00%: 99.54%, 99.21%, and 98.41% in Mmult, AES, and NIR Hawaii, respectively. In the case of roll-forwards, the success rate is lower, but it remains high: 90.19%, 90.61%, and 74.20% for Mmult, AES, and NIR Hawaii, respectively. It should be clarified that in addition to the 74.20% of successful roll-forwards in NIR Hawaii, 16.92% of the roll-forwards resulted in error correction but through the execution of rollbacks, as mentioned before.

We have also analyzed the occurrence of rollbacks and roll-forwards and their effectiveness considering the core where the event occurred. Table 8 shows the total number of rollbacks and roll-forwards per core, as well as the amount of those events that were corrected. Also, the percentages of errors corrected through rollback and roll-forward, relative to the total number of rollbacks and roll-forwards, respectively, are shown. The percentage of error correction through rollback is higher than 96.86% in all four cores and benchmarks. In the case of errors corrected through roll-forward, they are higher for the three secondary cores in Mmult and AES and lower for NIR Hawaii. It should be noted that the roll-forwards that appear in the column of the primary core (6 roll-forwards in Mmult and 6 roll-forwards in NIR Hawaii) have been triggered by an error produced in this core, but the roll-forwards have not been performed on this core. These errors occurred during the signature verification process of the three secondary cores, which is carried out on the primary core. As a result, one of the signatures resulted in being different from the others, and therefore, the roll-forward process was triggered in the corresponding secondary core. It is worth noting that the primary core lacks the ability to perform roll-forward within the primary core itself.

In Fig. 10 and Fig. 11, the number of corrected rollbacks, corrected roll-forwards, SW restarts, and WDT are presented considering the affected register. The results related to primary core are presented in Fig. 10: Fig. 10(a) Mmult, Fig. 10(b) AES and Fig. 10(c) NIR Hawaii; whereas the results related to secondary cores are shown in Fig. 11: Fig. 11(a) Mmult, Fig. 11(b) AES and Fig. 11(c) NIR Hawaii.

In Fig. 10, registers with the highest number of errors are shown, considering the primary core. Most errors occur in ELR (76.38% in Mmult, 73.26% in AES and 71.88% in NIR Hawaii) and PSTATE (19.96% in Mmult, 20.16% in AES and 19.64% in NIR Hawaii). Since the primary core is not capable of detecting data errors through comparison, most injected faults in general-purpose registers do not result in observable errors. If we analyze the errors specifically produced in ELR, it is observed that 76.29%, 69.31% and

TABLE 8. Distribution of rollbacks and roll-forwards per core.

Benchmark	Event	Primary core	Secondary cores Core1	Core2	Core3
Mmult	Total rollbacks	585	666	669	681
	Corrected rollbacks	573 (97.95%)	666 (100%)	669 (100%)	681 (100%)
	Total roll-forwards	6	1602	1653	1542
	Corrected roll-forwards	0 (0%)	1440 (89.89%)	1470 (88.93%)	1422 (92.22%)
AES	Total rollbacks	564	570	528	603
	Corrected rollbacks	549 (97.34%)	570 (100%)	525 (99.43%)	603 (100%)
	Total roll-forwards	0	513	519	438
	Corrected roll-forwards	-	435 (84.80%)	483 (93.06%)	414 (94.52%)
NIR Hawaii	Total rollbacks	477	762	747	843
	Corrected rollbacks	462 (96.86%)	759 (99.61%)	732 (97.99%)	831 (98.58%)
	Total roll-forwards	6	630	576	579
	Corrected roll-forwards	6 (100%)	477 (75.71%)	435 (75.52%)	411 (70.98%)
	Corrected roll-forwards via rollback	0 (0%)	120 (19.05%)	81 (14.06%)	102 (17.62%)

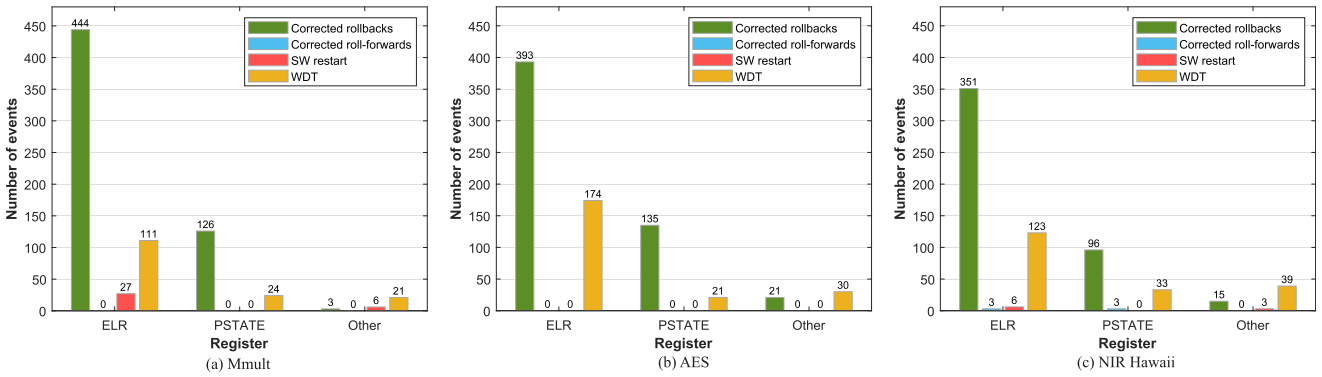


FIGURE 10. Distribution of errors considering the affected register on the primary core.

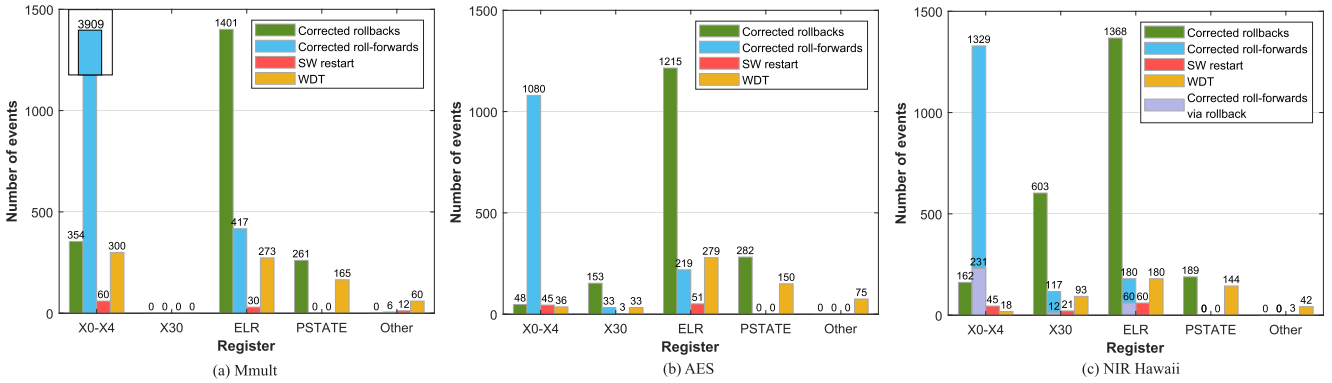


FIGURE 11. Distribution of errors considering the affected register on secondary cores.

72.67% of them (444, 393 and 351) were corrected through a rollback process, followed by 19.07%, 30.69% and 25.47% (111, 174 and 123), that were detected through the WDT, considering Mmult, AES and NIR Hawaii, respectively. When analyzing errors in PSTATE, the behavior is similar to ELR: the highest percentage also corresponds to errors corrected through rollbacks, followed by errors detected through the WDT.

In Fig. 11, it can be observed that most errors produced in the secondary cores are related to PSTATE and ELR registers, as in the primary core case, but in addition, the X0-X4 registers and X30 are involved. The STMLS technique ensures the detection and correction of most data errors (X0-X4 errors) through the roll-forward mechanism. In X0-X4 registers, 92.21%, 93.30% and 95.95% of errors produced are corrected (4263, 1128 and 1491): 84.56%, 89.33% and

TABLE 9. Distribution of exceptions per core.

Benchmark	Event	Primary core	Secondary cores		
			Core1	Core2	Core3
Mmult	Total exceptions	726	948	912	930
	Corrected exceptions	570 (78.51%)	660 (69.62%)	651 (71.38%)	681 (73.23%)
AES	Total exceptions	774	789	687	780
	Corrected exceptions	549 (70.93%)	561 (71.10%)	522 (75.98%)	600 (76.92%)
NIR Hawaii	Total exceptions	654	918	894	987
	Corrected exceptions	459 (70.18%)	756 (82.35%)	741 (82.89%)	828 (83.89%)

85.52% are corrected through roll-forward (3909, 1080 and 1329), and 7.65%, 3.97% and 10.43% through rollback (354, 48 and 162). Only 7.79%, 6.70% and 4.05% needed to be recovered using restarts: 1.30%, 3.72% and 2.90% SW restarts and 6.49%, 2.98% and 1.15% WDT. The X0-X4 registers are general-purpose registers used as data registers, and consequently, errors in these registers are corrected in most cases. It should be noted that in Fig. 11(c), cases in which a roll-forward process is started, then the rollback process is triggered, and finally the error is corrected, are shown as *Corrected roll-forwards via rollback*. This situation only occurs in the NIR Hawaii benchmark.

In Fig. 11, it can also be observed the high correction capacity in the rest of the registers. Regarding register X30, it can be observed that in Mmult, there are no errors corrected or recovered through any mechanism because no errors occurred in that register. However, in both AES and NIR Hawaii, the majority of errors produced are corrected through rollback: 153 in AES and 603 in NIR Hawaii, representing 68.88% and 76.51% respectively, of the total errors produced in X30. If we analyze the behavior in the PSTATE register, it can be observed that there are no errors corrected through roll-forward nor errors recovered through SW restart. All errors were corrected through rollback or resulted in a restart through WDT. This results from all errors in PSTATE being associated with exceptions since it contains important information regarding the processor's state. Due to the significance of exceptions and their impact on the proper functioning of the processor, it is important to analyze the level of recovery and correction provided by STMLS against this type of failure.

When a failure occurs, it can result in an exception, incapacitating the system to continue executing the software application. Addressing this issue is essential to obtain a high error correction rate with the hardening technique. Table 9 shows the total and corrected exceptions per core. The percentages of exceptions corrected for each core, considering the total exceptions generated in that core, are also reported. The percentages of exception correction range from 69.62% to 83.89%. As a result, the system was able to correct through rollback, 72.87%, 73.66% and 80.56% of the total exceptions in Mmult, AES, and NIR Hawaii, respectively. This high percentage of exception correction has contributed to the high error correction capacity demonstrated by the STMLS technique.

VI. CONCLUSION

This work presents a novel Supervised Triple Macrosynchronized Lockstep (STMLS) architecture designed to enhance the resilience of high-end multicore COTS processors. This technique is specifically tailored for multicore processors with more than three cores, addressing the unique challenges and benefits of a higher core count. The STMLS hardening technique not only implements the rollback but also the roll-forward mechanism to enhance the error correction capabilities. The roll-forward process involves overwriting the erroneous state of the faulty core using the current error-free context from one of the other two secondary cores. Additional recovery mechanisms, such as Watchdog timer, software restart, and program memory delimitation, are also employed to enhance system reliability. In the proposed hardened architecture, lockstep is carried out at the software level without specific hardware support.

For the implementation and evaluation of the proposed approach, the quad-core ARM Cortex-A53 processor embedded into the AMD Zynq UltraScale+ MPSoC ZU3EG A484 device was used, and an extensive fault injection campaign was conducted. The experimental findings demonstrated that, apart from achieving full error coverage, the proposed STMLS achieved enhanced error correction capabilities, yielding 86.40%. This represents a notable improvement compared to the 51.4% and subsequent 65.87% we previously obtained in related work. Remarkably, these results substantiate the high efficacy of the STMLS in enhancing the reliability of high-end multicore COTS processors.

REFERENCES

[1] M. M. Ghahroodi, E. Ozer, and D. Bull, "SEU and SET-tolerant ARM Cortex-R4 CPU for space and avionics applications," in *Proc. Workshop Manufacturable Dependable Multi-Core Architectures Nanosc.*, 2013, pp. 1–4. [Online]. Available: https://www.median-project.eu/wp-content/uploads/median2013_submission_5.pdf

[2] *UT700 32-bit Fault-Tolerant SPARC V8/LEON 3FT Processor*, Frontgrade Gaisler, Gothenburg, Sweden, 2015.

[3] R. Berger, D. Artz, and P. Kapcio, 'RAD750 radiation hardened PowerPC microprocessor," Lockheed Martin Space Electron. Commun., BAE Syst., Manassas, VA, USA, 2000. [Online]. Available: http://powerpc.i-logout.cz/docs/rad750.pdf and https://www.baesystems.com/en-media/uploadFile/20210404045936/1434555668211.pdf

[4] A. Vernile, *The Rise of Private Actors in the Space Sector*. Cham, Switzerland: Springer, 2018.
[5] R. F. Hodson, Y. Chen, J. E. Pandolf, K. Ling, K. T. Boomer, C. M. Green, J. A. Leitner, P. Majewicz, S. H. Gore, C. S. Faller, E. C. Denson, R. E. Hodge, A. P. Thoren, and M. A. Defrancis, "Recommendations on use of commercial-off-the-shelf (COTS) electrical, electronic, and electromechanical (EEE) parts for NASA missions," NASA STI Program, NASA Langley Res. Center, Hampton, VA, USA, Tech. Rep. NESC-RP-19-01490 NASA/TM-20205011579, 2020. [Online]. Available: https://ntrs.nasa.gov/citations/20205011579
[6] M. Pignol, "COTS-based applications in space avionics," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2010, pp. 1213–1219.
[7] S. Esposito, C. Albanese, M. Alderighi, F. Casini, L. Giganti, M. L. Esposti, C. Monteleone, and M. Violante, "COTS-based high-performance computing for space applications," *IEEE Trans. Nucl. Sci.*, vol. 62, no. 6, pp. 2687–2694, Dec. 2015.
[8] L. Osinski, T. Langer, and J. Mottok, "A survey of fault tolerance approaches on different architecture levels," in *Proc. 30th Int. Conf. Archit. Comput. Syst. (ARCS)*, Apr. 2017, pp. 1–9.
[9] ARM. (2011). *Cortex-R5 and Cortex-R5F. Technical Reference Manual, Version R1P1 Issue C*. [Online]. Available: https://documentation-service.arm.com/static/5f042788cafe527e86f5cc83?token=
[10] P. M. Aviles, A. Lindoso, J. A. Belloch, M. Garcia-Valderas, Y. Morilla, and L. Entrena, "Radiation testing of a multiprocessor macrosynchronized lockstep architecture with FreeRTOS," *IEEE Trans. Nucl. Sci.*, vol. 69, no. 3, pp. 462–469, Mar. 2022.
[11] P. M. Aviles, L. Schäfer, A. Lindoso, J. A. Belloch, and L. Entrena, "High complexity reliable space applications in commercial microprocessors," *Microelectron. Rel.*, vol. 138, Nov. 2022, Art. no. 114679.
[12] M. Nicolaidis, *Soft Errors in Modern Electronic Systems*, vol. 41. New York, NY, USA: Springer, 2011.
[13] M. P. Pignol, "DMT and DT2: Two fault-tolerant architectures developed by CNES for COTs-based spacecraft supercomputers," in *Proc. 12th IEEE Int. On-Line Test. Symp. (IOLTS)*, Jul. 2006, p. 10.
[14] K. Greb and D. Pradhan, "Hercules^{TM} microcontrollers: Real-time MCUs for safety-critical products," Texas Instrum., Dallas, TX, USA, White Paper, 2011. [Online]. Available: https://www.electronicproducts.com/hercules-microcontrollers-real-time-mcus-for-safety-critical-products/ and https://www.ti.com/lit/fs/spry178/spry178.pdf?ts=1699989694500&ref_url=https%253A%252F%252Fwww.google.com%252F
[15] Á. B. de Oliveira, G. S. Rodrigues, F. L. Kastensmidt, N. Added, E. L. A. Macchione, V. A. P. Aguiar, N. H. Medina, and M. A. G. Silveira, "Lockstep dual-core ARM A9: Implementation and resilience analysis under heavy ion-induced soft errors," *IEEE Trans. Nucl. Sci.*, vol. 65, no. 8, pp. 1783–1790, Aug. 2018.
[16] M. Peña-Fernández, A. Serrano-Cases, A. Lindoso, S. Cuenca-Asensi, L. Entrena, Y. Morilla, P. Martín-Holgado, and A. Martínez-Álvarez, "Hybrid lockstep technique for soft error mitigation," *IEEE Trans. Nucl. Sci.*, vol. 69, no. 7, pp. 1574–1581, Jul. 2022.
[17] X. Iturbe, B. Venu, E. Ozer, and S. Das, "A triple core lock-step (TCLS) ARM Cortex-R5 processor for safety-critical and ultra-reliable applications," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshop (DSN-W)*, Jun. 2016, pp. 246–249.
[18] X. Iturbe, B. Venu, E. Ozer, J.-L. Poupat, G. Gimenez, and H.-U. Zurek, "The arm triple core lock-step (TCLS) processor," *ACM Trans. Comput. Syst.*, vol. 36, no. 3, pp. 1–30, Aug. 2018.
[19] X. Iturbe, B. Venu, J. Jagst, E. Ozer, P. Harrod, C. Turner, and J. Penton, "Addressing functional safety challenges in autonomous vehicles with the arm TCL s architecture," *IEEE Design Test.*, vol. 35, no. 3, pp. 7–14, Jun. 2018.
[20] S. Kasap, E. W. Wächter, X. Zhai, S. Ehsan, and K. D. McDonald-Maier, "Novel lockstep-based fault mitigation approach for SoCs with roll-back and roll-forward recovery," *Microelectron. Rel.*, vol. 124, Sep. 2021, Art. no. 114297.
[21] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," *ACM SIGARCH Comput. Archit. News*, vol. 28, no. 2, pp. 25–36, May 2000.
[22] Y. Hua, C. Gang, and Y. Xiao-Zong, "TRSTR: A fault-tolerant microprocessor architecture based on SMT," *Wuhan Univ. J. Natural Sci.*, vol. 10, no. 1, pp. 51–55, Jan. 2005.
[23] Y. Ma, H. Gao, M. Dimitrov, and H. Zhou, "Optimizing dual-core execution for power efficiency and transient-fault recovery," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 8, pp. 1080–1093, Aug. 2007.
[24] H. Zhou, "A case for fault tolerance and performance enhancement using chip multi-processors," *IEEE Comput. Archit. Lett.*, vol. 5, no. 1, pp. 22–25, Jan. 2006.
[25] Z. Wang and A. Chattopadhyay, *High-level Estimation and Exploration of Reliability for Multi-Processor System-on-Chip*. Singapore: Springer, 2017.
[26] P. M. Aviles, A. Lindoso, J. A. Belloch, and L. Entrena, "Evaluating reliability through soft error triggered exceptions at ARM Cortex-A9 microprocessor," *Microelectron. Rel.*, vol. 126, Nov. 2021, Art. no. 114323.
[27] ARM. (2018). *ARM Cortex-A53 MPCore Processor. Technical Reference Manual, Revision R0P4*. [Online]. Available: https://developer.arm.com/documentation/ddi0500/j
[28] ARM. (2020). *ARMv8-A Instruction Set Architecture, Issue 1.1*. [Online]. Available: https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Learn%20the%20Architecture/Armv8-A%20Instruction%20Set%20Architecture.pdf?revision=ebf53406-04fd-4c67-a485-1b329febfb3e
[29] ARM. (2018). *ARM Architecture Reference Manual. ARMv7-A and ARMv7-R Edition, Issue C.d*. [Online]. Available: https://developer.arm.com/documentation/ddi0406/c
[30] AMD. (2023). *Zynq UltraScale+ Device Technical Reference Manual, UG1085 (V2.3.1)*. [Online]. Available: https://docs.xilinx.com/r/en-US/ug1085-zynq-ultrascale-trm/Zynq-UltraScale-Device-Technical-Reference-Manual
[31] AMD. (2021). *Block Memory Generator V8.4 LogiCORE IP Product Guide, PG058*. [Online]. Available: https://docs.xilinx.com/v/u/en-US/pg058-blk-mem-gen
[32] AMD. (2019). *AXI Block RAM (BRAM) Controller V4.1 LogiCORE IP Product Guide, PG078*. [Online]. Available: https://docs.xilinx.com/v/u/en-US/pg078-axi-bram-ctrl
[33] M. Y. Hsiao, "A class of optimal minimum odd-weight-column SEC-DED codes," *IBM J. Res. Develop.*, vol. 14, no. 4, pp. 395–401, Jul. 1970.
[34] AMD. (2018). *Mailbox V2.1 LogiCORE IP Product Guide Vivado Design Suite, PG114*. [Online]. Available: https://docs.xilinx.com/v/u/en-US/pg114-mailbox
[35] AMD. (2022). *SmartConnect V1.0 LogiCORE IP Product Guide, PG247*. [Online]. Available: https://docs.xilinx.com/r/en-US/pg247-smartconnect/SmartConnect-v1.0-LogiCORE-IP-Product-Guide
[36] *Ultra96-V2 Single Board Computer Hardware User's Guide, Version 1.3*, Avnet, Phoenix, AZ, USA, 2021.
[37] AMD. (2021). *Vitis Unified Software Platform, V.2021.2*. [Online]. Available: https://www.xilinx.com/products/design-tools/vitis.html
[38] R. Velazco, S. Rezgui, and R. Ecoffet, "Predicting error rate for microprocessor-based digital architectures through C.E.U. (code emulating upsets) injection," *IEEE Trans. Nucl. Sci.*, vol. 47, no. 6, pp. 2405–2411, Dec. 2000.
[39] H. Quinn, W. H. Robinson, P. Rech, M. Aguirre, A. Barnard, M. Desogus, L. Entrena, M. Garcia-Valderas, S. M. Guertin, D. Kaeli, F. L. Kastensmidt, B. T. Kiddie, A. Sanchez-Clemente, M. S. Reorda, L. Sterpone, and M. Wirthlin, "Using benchmarks for radiation testing of microprocessors and FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 62, no. 6, pp. 2547–2554, Dec. 2015.
[40] W. E. Burr, "Selecting the advanced encryption standard," *IEEE Secur. Privacy*, vol. 1, no. 2, pp. 43–52, Mar. 2003.
[41] European Space Agency. (Sep. 2018). *NIR HAWAII-2RG Data Processing Algorithms—Benchmarking Software*. [Online]. Available: https://essr.esa.int/project/nir-hawaii-2rg-data-processing-algorithms-benchmarking-software
[42] J. W. Beletic, R. Blank, D. Gulbransen, D. Lee, M. Loose, E. C. Piquette, T. Sprafke, W. E. Tennant, M. Zandian, and J. Zino, "Teledyne imaging sensors: Infrared imaging technologies for astronomy and civil space," *Proc. SPIE*, vol. 7021, Jul. 2008, Art. no. 70210H.
[43] AMD. (2021). *Vivado ML Edition, V.2021.2*. [Online]. Available: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html
[44] AMD. (2023). *Xilinx Power Estimator, V.2023.1.2*. [Online]. Available: https://www.xilinx.com/products/technology/power/xpe.html
[45] D. V. C. D. Nascimento, K. Georgiou, K. I. Eder, and S. Xavier-de-Souza, "Evaluating the effects of reducing voltage margins for energy-efficient operation of MPSoCs," *IEEE Embedded Syst. Lett.*, early access, Jan. 30, 2023, doi: 10.1109/LES.2023.3240625.
[46] E. Taka, G. Lentaris, and D. Soudris, "Improving the performance of RISC-V softcores on FPGA by exploiting PVT variability and DVFS," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2022, pp. 1595–1599.

[47] H. Ali, U. U. Tariq, L. Liu, J. Panneerselvam, and X. Zhai, "Energy optimization of streaming applications in IoT on NoC based heterogeneous MPSoCs using re-timing and DVFS," in *Proc. IEEE SmartWorld, Ubiquitous Intell. Comput., Adv. Trusted Comput., Scalable Comput. Commun., Cloud Big Data Comput., Internet People Smart City Innov. (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, Aug. 2019, pp. 1297–1304.

[48] C. Gou, A. Benoit, M. Chen, L. Marchal, and T. Wei, "Reliability-aware energy optimization for throughput-constrained applications on MPSoC," in *Proc. IEEE 24th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2018, pp. 1–10.

[49] G. Onnebrink, F. Walbroel, J. Klimt, R. Leupers, G. Ascheid, L. G. Murillo, S. Schürmans, X. Chen, and Y. Harn, "DVFS-enabled power-performance trade-off in MPSoC SW application mapping," in *Proc. Int. Conf. Embedded Comput. Syst., Architectures, Modeling, Simulation (SAMOS)*, Jul. 2017, pp. 196–202.

[50] Y. Wang, J. Xu, Y. Xu, W. Liu, and H. Yang, "Power gating aware task scheduling in MPSoC," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 10, pp. 1801–1812, Oct. 2011.

[51] G. M. Banu, "MPSoC based dynamic power management in wireless sensor networks," in *Proc. Int. Conf. Inf. Commun. Embedded Syst. (ICICES)*, Feb. 2014, pp. 1–6.

[52] Y.-J. Lin, C.-L. Yang, J.-W. Huang, and N. Chang, "Memory access aware power gating for MPSoCs," in *Proc. 17th Asia South Pacific Design Autom. Conf.*, Jan. 2012, pp. 121–126.

[53] T. Rauber, G. Rünger, and M. Stachowski, "Performance and energy metrics for multi-threaded applications on DVFS processors," *Sustain. Comput., Informat. Syst.*, vol. 17, pp. 55–68, Mar. 2018.

[54] G. Akgün, L. Kalms, and D. Göhringer, "Resource efficient dynamic voltage and frequency scaling on Xilinx FPGAs," in *Proc. 16th Int. Symp. Appl. Reconfigurable Comput.*, Toledo, Spain, F. Rincón, J. Barba, H. K. H. So, P. Diniz, and J. Caba, Eds. Cham, Switzerland: Springer, Apr. 2020, pp. 178–192.

PABLO M. AVILES (Graduate Student Member, IEEE) received the B.S. degree in telecommunications and electronic engineering from Universidad Central "Marta Abreu" de Las Villas, Villa Clara, Cuba, in 2015, and the M.S. degree in electronic systems engineering from the University Carlos III of Madrid, Madrid, Spain, in 2019, where he is currently pursuing the Ph.D. degree in electrical engineering, electronics and automation with the Microelectronic Design and Applications (DMA) Research Group, Electronic Technology Department.

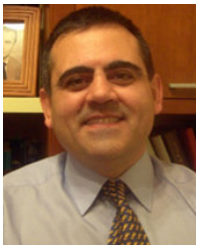
From 2015 to 2018, he was with Cuban Radiocommunication and Broadcasting Company (RADIOCUBA), Cuba. His research interests include microprocessor architectures, embedded systems, soft errors, radiation hardening, and fault tolerance.

Mr. Aviles was honored with a Predoctoral Fellowship from the Community of Madrid, co-financed by the European Social Fund through the Youth Employment Operational Program and the Youth Employment Initiative (YEI). He was awarded the Idea Puzzle Prize 2020 for the most coherent doctoral research design created with the Idea Puzzle software. He received Financial Aid from the University Carlos III of Madrid for a predoctoral research stay with Laboratoire d'Informatique, de Robotique et de Microelectronique de Montpellier (LIRMM), University of Montpellier, Montpellier, France, in 2022.

JOSE A. BELLOCH (Member, IEEE) received the degree in telecommunications engineering, the M.S. degree in parallel and distributed computing, and the Ph.D. degree in computer science from Universitat Politãcnica de Valãncia, Valencia, Spain, in 2007, 2010, and 2014, respectively.

Since 2017, he has been an Assistant Professor with the Electronic Technology Department, University Carlos III of Madrid, Madrid, Spain. He was a Visiting Researcher with the Department of Signal Processing and Acoustics, Aalto University School of Electrical Engineering, Espoo, Finland, as a Predoctoral Researcher, in 2013, and a Postdoctoral Researcher, in 2015. He carried out an internship with the Department of Measurement and Information Systems, Budapest University of Technology and Economics, Budapest, Hungary. His current research interest includes applying the new parallel architectures to signal processing algorithms.

Dr. Belloch was a recipient of the Extraordinary Ph.D. Thesis Award. In 2016, he was honored with a Postdoctoral Fellowship from the Regional Government of Valencia to conduct research with Universitat Jaume I de Castellãn in collaboration with Universidad Complutense de Madrid, Madrid.



LUIS ENTRENA (Member, IEEE) received the degree in industrial engineering from Universidad de Valladolid, Spain, in 1988, and the Ph.D. degree in electronic engineering from Universidad Politãcnica de Madrid, Madrid, Spain, in 1995.

From 1990 to 1993, he was with AT&T Microelectronics, Bell Laboratories, USA. From 1993 to 1996, he was the Technical Project Leader with TGI, Spain. In 1996, he joined the University Carlos III of Madrid, Spain, where he is currently a Full Professor. Previously, he was the Head of the Electronic Technology Department and the Director of the Postgraduate Program in Electrical Engineering. He has coauthored over 180 papers in journals and conferences and two patents. His current research interests include online testing, fault tolerance, soft error sensitivity evaluation and mitigation, hardware security, and hardware acceleration.

ALMUDENA LINDOSO (Senior Member, IEEE) received the M.S. degree in telecommunication engineering from Universidad Politãcnica de Madrid, in 2001, and the Ph.D. degree from the University Carlos III of Madrid, in 2009.

Since 2003, she has been a Professor and a Researcher with the Electronic Technology Department, University Carlos III of Madrid, where she is currently an Associate Professor. She has coauthored over 75 publications in journals and conferences. Her research interests include the fields of hardware acceleration, image processing, and fault tolerance. Her current research interest includes the adaptation in terms of reliability of high-performance commercial circuits for aerospace applications.