S-SETA: Selective Software-Only Error-Detection Technique Using Assertions

Eduardo Chielle, Gennaro S. Rodrigues, Fernanda L. Kastensmidt, Sergio Cuenca-Asensi, Lucas A. Tambara, Paolo Rech, and Heather Quinn

***Abstract*—Software-based techniques offer several advantages to increase the reliability of processor-based systems at very low cost, but they cause performance degradation and an increase of the code size. To meet constraints in performance and memory, we propose SETA, a new control-flow software-only technique that uses assertions to detect errors affecting the program flow. SETA is an independent technique, but it was conceived to work together with previously proposed data-flow techniques that aim at reducing performance and memory overheads. Thus, SETA is combined with such data-flow techniques and submitted to a fault injection campaign. Simulation and neutron induced SEE tests show high fault coverage at performance and memory overheads inferior to the state-of-the-art.**

***Index Terms*—Aerospace applications, control-flow, energy constraints, error detection, fault coverage, fault tolerance, memory overhead, performance degradation, processors, reliability, soft errors, software-based techniques.**

I. Introduction

THE advances in the semiconductor industry have allowed the fabrication of high density integrated circuits (ICs). Nowadays, we are reaching the physical limits of a couple atoms to form the transistor's gate [1][2]. However, the higher quantity of transistors per die combined with reduced voltage threshold and increased operating frequencies have made ICs more sensitive to faults caused by radiation [3]. Such faults can be caused by energized particles present in space or secondary particles such as alpha particles, generated by the interaction of neutron and materials at ground level [4].

Transient ionization may occur when a single radiation ionizing particle strikes the silicon, creating a transient voltage pulse, or a Single Event Effect (SEE). This effect affects processors by modifying values stored in the sequential logic, known as Single Event Upset (SEU), or by changing the function of a circuit in the combinational, known as Single Event Transient (SET). Such faults may lead the system to incorrectly execute an application. In critical processor-based systems, an error is unacceptable. Thus, to ensure reliability for the microprocessor against SEEs, the use of fault tolerance techniques is mandatory.

There are two main types of fault tolerance techniques that aim to harden processors under neutron induced soft errors, which are hardware-based and software-based techniques. The first one relies on replicating or adding hardware modules. And the second one relies on the replication of information and instructions in the program code [5]. Hardware-based techniques usually change the original processor architecture by adding logic redundancy, error correcting codes and majority voters. They can also be based on hardware monitoring devices, called watchdog processors [6], to monitor memory accesses. However, hardware-based techniques present significant overheads, like increase in area and power consumption, and high design and manufacturing costs [7]. Software-based techniques are a well-known approach to protect systems against SEEs by modifying the program code, without having to change the underlying hardware. They rely on adding instruction redundancy and comparison to detect or correct errors. These techniques provide high flexibility and low development time and cost. In addition, they allow the use of commercial off-the-shelf (COTS) processors since no modification to the hardware is required. Although software redundancy brings reliability to the system, it requires extra processing time since more instructions are being executed, and more memory, since redundancy is inserted. As a consequence, the energy consumption is increased [8][9]. In a previous paper, we reduced those overheads for data-flow techniques [10].

In this paper, we propose a new control-flow technique called SETA (Software-only Error-detection Technique using Assertions) and a new method for selective hardening, called tunnel effect. SETA is a technique to detect control-flow errors using assertions with lower overheads designed to be used together with data-flow techniques. Then, the selective hardening method is implemented with SETA, creating S-SETA (Selective SETA). S-SETA increases the flexibility of the original technique and allows optimizing the trade-off between overheads and reliability. The techniques are evaluated in terms of execution time, code size and fault coverage, and compared to the literature. Experiments were performed with processors miniMIPS and ARM Cortex-A9. The fault coverage was obtained by simulation and also neutron induced SEE tests. Results from fault injection show around 98% fault coverage at overheads in performance and memory inferior to

Manuscript received July 09, 2015; revised September 08, 2015; accepted September 28, 2015. This work was supported in part by CNPq and CAPES, Brazilian agencies.

E. Chielle, G. S. Rodrigues, F. L. Kastensmidt, L. A. Tambara and P. Rech are with the Instituto de Informática, PGMICRO,UFRGS, Porto Alegre, Brazil (e-mail: echielle@inf.ufrgs.br; gsrodrigues@inf.ufrgs.br; fglima@inf.ufrgs.br; latambara@inf.ufrgs.br; prech@inf.ufrgs.br).

S. Cuenca-Asensi is with the Computer Technology Department, University of Alicante, Alicante, Spain (e-mail: sergio@dtic.ua.es).

H. Quinn is with the Space Data System Group, Los Alamos National Laboratory, Los Alamos, NM, USA (e-mail: hquinn@lanl.gov).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNS.2015.2484842

state-of-the-art techniques. Results from the neutron radiation test confirm the high fault coverage obtained by simulation.

II. Fault Tolerance in Software

Software-based fault tolerance techniques, also referred in the literature as Software-Implemented Hardware Fault Tolerance (SIHFT) [11], are techniques implemented in software to protect processor against soft errors that may affect the program flow or the data stored in registers or memory. The techniques that are aim to protect the data are called data-flow techniques, and the ones to protect the control-flow are the control-flow techniques. There are also techniques that combine features of both data and control-flow techniques. They consist of transformation rules of the code and can be understood as a data-flow and control-flow technique applied together. Table I summarizes both types of software-based techniques.

TABLE I
Types of Software-Based Fault Tolerance Techniques

Data-flow techniques	Control-flow techniques
• Protect the data-flow ▪ data stored in registers ▪ data stored in memory	• Ensure the integrity of the execution flow
• Registers are replicated	• Divide the code into basic blocks
• Checkers are inserted	• A unique signature is assigned to each basic block
• Every operation performed on a register must be performed on its replica	• Checking instructions are inserted in the program code

A. Data-Flow Techniques

Data-flow techniques are designed to protect the data stored in registers or memory. These techniques replicate the registers, assigning copies to the original ones. When the aim is error detection, registers are duplicated, and when correction is included, registers are triplicated. Checkers (voters, if correction) are inserted in the code to compare the registers with its copies. An example is shown in Fig. 1. The points where the checkers are inserted depend on the technique. Since error detection presents lower overheads than correction due to duplication instead of triplication, this paper focuses on error detection. Some well-known data-flow techniques present in the literature are EDDI [12] and Variables [13].

B. Control-Flow Techniques

Control-flow techniques are designed to protect the program's flow, i.e., to protect against incorrect branches. Such techniques divide the code into basic blocks. A basic block (BB) is a branch-free sequence of instructions, i.e., a portion of code that is always executed in sequence. There only can be a branch instruction at the end of the basic block. Further, there are no branches to the basic block, except to the first instruction. For each basic block, a signature is assigned. The signature is attributed to a global register at the beginning of the basic block. Checkers are inserted in the code to verify if the signature register contains the expected value. If it does not, it means there was an incorrect branch and an error is reported. The main control-flow techniques present in the literature are CFCSS [14], YACCA [15] and CEDA [16]. Table II shows the execution time and fault coverage of these techniques. As one can see, CEDA is the one with the highest trade-off between fault coverage and performance, and that is why it is used as the baseline technique of this paper.

TABLE II
State-of-the-Art Control-Flow Techniques [16]

Benchmarks	CFCSS		YACCA		CEDA	
	UF	ET	UF	ET	UF	ET
parser	4.6%	1.14x	1.0%	1.34x	1.1%	1.14x
gzip	3.4%	1.58x	0.7%	1.84x	0.6%	1.58x
ammp	4.7%	1.04x	0.3%	1.79x	0.2%	1.03x
twolf	2.8%	1.08x	0.6%	1.40x	0.6%	1.10x
equake	2.8%	1.19x	0.5%	1.34x	0.5%	1.18x

UF: Undetected Faults
ET: Execution Time

III. Proposed Techniques

A. VAR3+ Data-Flow Technique

In [10], seventeen data-flow techniques that aim at reducing the overheads in performance, memory and energy consumption were presented and validated by fault injection. They consist of three types of different rules: global, duplicating and checking rules, as one can see in Table III. There is only a global rule, and it is applied to all techniques. It states that every register used by the program has a spare register assigned as its replica. The duplicating rules regard how the instructions will be duplicated. They are only applied when write operations in a register or memory are performed. Therefore, branch instructions are not considered in this case. There are two types of duplicating rules. Each technique can only have one duplicating rule. D1 duplicates all instructions, including stores, which allow the use of unprotected memories, since the original value and its replica can be stored in different positions in the memory. D2 duplicates all instructions, except stores. The last one is adequate when the memory is hardened because the data in memory do not need to be duplicated. Thus, the duplication overhead and the number of memory accesses are reduced. The checking rules indicate when a register and its replica must be compared aiming at verifying if an error has occurred (when they present different values). The techniques can have more than one checking rule. Theoretically, the more checkers are included in one technique, the more reliability is achieved.

Of the seventeen proposed data-flow flow techniques that were evaluated by fault injection, we selected the one with the lowest overheads of the ones with the highest error detection rate. This technique is named VAR3+ and the rules it uses are presented in Table IV. As one can notice, VAR3+ uses duplicating rule D2 and checking rules C3, C4, C5, and C6. Table V shows an example of a code hardened by VAR3+. The original code is presented as normal text, the code inserted by the duplicating rule is formatted as italic (lines 3, 6 and 8), and the checkers are bold (lines 1, 4, 9 and 10).

TABLE III
RULES FOR DATA-FLOW TECHNIQUES [10]

	Global Rules (valid for all techniques)
G1	each register used in the program has a spare register assigned as replica
	Duplication Rules (performing the same operation on the register's replica)
D1	all instructions
D2	all instructions, except stores
	Checking Rules (compare the value of a register with its replica)
C1	before each read on the register (except load/store and branch/jump instructions)
C2	after each write on the register
C3	the register that contains the address before loads
C4	the register that contains the datum before stores
C5	the register that contains the address before stores
C6	before branches or jumps

This paper proposes a new control-flow technique. However, we will use VAR3+ data-flow technique to complement the proposed control-flow technique once it has been compared to the state-of-the-art.

B. SETA Control-Flow Technique

To complement the data-flow techniques and, thus, promote protection of both data and control-flow, we propose a new technique based on HETA [17] and CEDA. All of them use runtime signatures to detect faults in the control-flow. However, HETA makes use of hardware to help in the detection, which requires extra power and also cannot be applied to most COTS devices. Furthermore, both CEDA and HETA are concerned about the fault coverage, but not about the overheads they cause. Aiming at providing similar error detection rate as CEDA at lower overheads, SETA is proposed. The technique makes use of signatures, calculated a priori and processed at runtime. The program code is divided into basic blocks, and the correlation among them based on the program flow is used to calculate the signatures in such a manner to warrant detection of control-flow errors.

Two Basic Block Types (BBT) are defined: A and X. A basic block is of type A if it has multiple predecessors and at least one of its predecessors has multiple successors, and it is of type X if it is not of type A. After that, the basic blocks are grouped into networks. Basic blocks with common predecessors belong to the same network. Then, SETA calculates the signatures. Two different Basic Block Signatures (BBS) are used: a Node Ingress Signature (NIS) and a Node Exit Signature (NES). To verify those signatures during runtime, a Signature Register (S) is used. The signatures values are divided into two parts: an upper half and a lower half, as shown in Table VI. Each part is calculated differently, and their sizes may vary according to the program code requirements to avoid aliasing. The upper half is used to identify the network that the basic block belongs to (in the case of the NIS) and the network of the successor basic block (in the case of the NES). The lower half is used to differentiate the basic blocks inside of a network. An example can be seen in Fig. 2. At runtime, S is updated according to the following rules by using XOR or AND operation with an invariant. The

TABLE IV
RULES FOR VAR3+ DATA-FLOW TECHNIQUES

Technique	Duplicating Rule	Checking Rules
VAR3+	D2	C3, C4, C5, C6

TABLE V
EXAMPLE OF VAR3+ DATA-FLOW TECHNIQUE FOR MINIMIPS PROCESSOR

#	Unhardened code	Code hardened by VAR3+
1		**bne $2,$12,error**
2	lw $4,0($2)	lw $4,0($2)
3		*lw $14,0($12)*
4		**bne $2,$12,error**
5	lw $5,4($2)	lw $5,4($2)
6		*lw $15,4($12)*
7	add $3,$3,1	add $3,$3,1
8		*add $13,$13,1*
9		**bne $4,$14,error**
10		**bne $5,$15,error**
11	sw $4,0($5)	sw $4,0($5)
12	ble $3,$6,loop	ble $3,$6,loop

TABLE VI
SIGNATURE DIVISION

Upper half	Lower half
01000100	01001011110101001011101

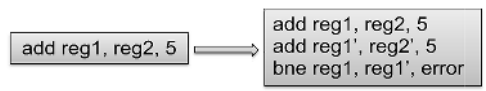

Fig. 1. Example of a data-flow technique.

invariant is a constant that updates the signature to its new value. Equations (1) and (2) are the two possible ways to update S. The updating of S follows the rules presented by Table VII. The operation will use and AND when BBT is A and S is updating to NIS. Otherwise, S will be updated by an XOR

$$S \leftarrow S \oplus invariant\ (BBS) \quad (1)$$

$$S \leftarrow S \bullet invariant\ (BBS). \quad (2)$$

XOR is a preferable operation to update S because it does not mask bits. Thus, an error affecting the signature will be propagate to the next basic block, and it can be detected later. However, when a basic block is of type A, i.e., when it has multiple predecessors and at least one of its predecessors has multiple successors, it is necessary to mask some bits to keep the signature consistent. An XOR cannot be used in this case because it does not mask bits. On the other hand, the AND can do that, masking only the bits that needed to be masked.

To detect incorrect branches, checkers can be inserted before exiting transitions. The more checkers, the lower is the latency to detect errors. On the other hand, higher is the overhead. The maximum number of checkers matches the number of basic blocks since only one checker is needed per basic block. Table VIII shows an example of SETA for miniMIPS processor.

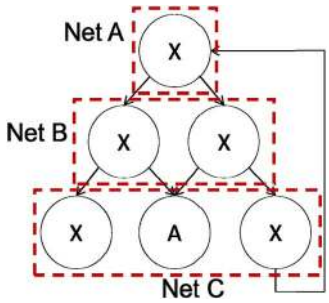

Fig. 2. Representation of a program flow. Basic blocks (circles) classified as type A or X, and grouped into networks. The arrows indicate the possible directions that a basic block may take.

TABLE VII
SIGNATURE UPDATE

BB Type	NIS	NES
A	AND	XOR
X	XOR	XOR

TABLE VIII
EXAMPLE OF SETA CONTROL-FLOW TECHNIQUE FOR MINIMIPS PROCESSOR

#	Unhardened code	Code hardened by SETA
1	jal dfs	jal dfs
2		*xori $7,$7,41407*
3	la $2,$result	la $2,$result
4	lw $4,0($6)	lw $4,0($6)
5	sw $6,4($2)	sw $6,4($2)
6	sw $4,0($2)	sw $4,0($2)
7		**li $8,41407**
8		**bne $7,$8,error**
9		*xori $7,$7,29184*
10	j loop	j loop

At the left, there is a portion of an unhardened code and, at the right, the same code hardened by SETA. The checkers inserted by SETA are in bold and the signature updates are formatted as italic. The first XOR (*xori*) is to update the signature to the basic block's NIS. The instructions *li* and *bne* are used to compare the signature register *$7* with the expected signature for that basic block. Finally, the last XOR is used to update the signature to the expected NES. Since new instructions are inserted, it is clear that the execution time and the code size will increase.

The main differences from CEDA to SETA are:

- Removed inverted branches check. CEDA inserts branches at both possible targets of each branch to check it was taken correctly. SETA does not implement it because the fault coverage it provides is negligible in relation to the overheads it causes. It only detects errors affecting the decision of a branch when the registers and the comparison are correct, but the branch take the wrong direction.
- Removed extra instructions used to avoid aliasing. SETA does not need to insert instructions to "clear" the signature, as it is done in CEDA, because the signature values are assigned in a different way. The upper half is deterministic and the lower half is randomly determined. Thus, the signature register can be always directly updated, which reduces the overheads. SETA avoid aliasing by varying the size of the "halves", trying to maximize the size of the lower half.

C. Evaluation Methodology

The following parameters were used to evaluate the quality of the proposal and to compare with the state-of-the-art:

- **Execution time**: it expresses the amount of time an application takes to execute. The execution time of a hardened application is presented normalized, i.e., it is divided by the execution time of the unhardened application;
- **Code size**: it refers to the amount of memory a program occupies in disk. The code size of a hardened application is also normalized by the unhardened application;
- **Fault coverage**: the fault coverage is the amount of detected and masked faults. It is expressed in percentage, and it is given by the equation (3). The fault coverage is the sum of detected faults with masked faults, divided by the total of faults. It can also be expressed as one minus the total of faults that caused undetected errors, divided by the total of faults

$$F_{coverage} = \frac{F_{detected} + F_{masked}}{F_{total}} = 1 - \frac{F_{undetected}}{F_{total}}. \quad (3)$$

- **Mean Work To Failure (MWTF)**[18]: the MWTF, given by equation (4), was used as an overall quality metric. It captures the tradeoff between reliability and performance, since the more time an application needs to run, the higher the probability of being hit by a particle and, consequently, affected by a fault. AVF (Average Vulnerability Factor) is used to measure microarchitectural structure's susceptibility to transient faults [19]. The *raw error rate* is the percentage of errors not detected. In this paper, MWTF of a hardened application is normalized by MWTF of the unhardened application

$$MWTF = \frac{amount\ of\ work\ completed}{number\ of\ error\ sencountered} = (raw\ error\ rate \times AVF \times execution\ time)^{-1}. \quad (4)$$

Large fault injection campaigns were performed over a set of benchmarks. Most of them were done by means of simulation tools because the amount of information necessary for all the tests is infeasible to be obtained by radiation test only. Anyhow, radiation tests were performed to check if their results match with ones obtained by simulation, and thus, validate the fault injections by simulation.

Faults were injected by forcing a bit-flip at RTL level in the miniMIPS [20] processor's internal signals using ModelSim [21], a simulation tool. Every signal is considered. A total of 10,000 faults is injected per application. Only one fault is injected per execution. The fault duration is set to one clock cycle in order to force their effect to hit the clock barrier of the flip-flops and, therefore, increase the probability of an error. A golden execution (with no injected faults) is executed. Then, the program is submitted to faults, and the memory results of the program under test are compared to the golden results. The

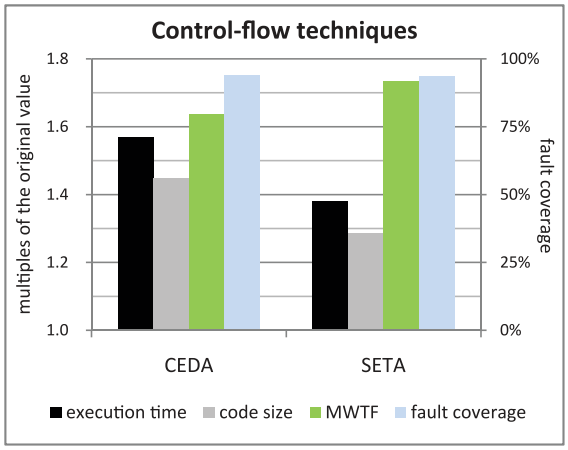

Fig. 3. Comparison between CEDA and SETA for miniMIPS processor. Both techniques present similar fault coverage. However, SETA presents significant lower overheads, which explains its higher MWTF.

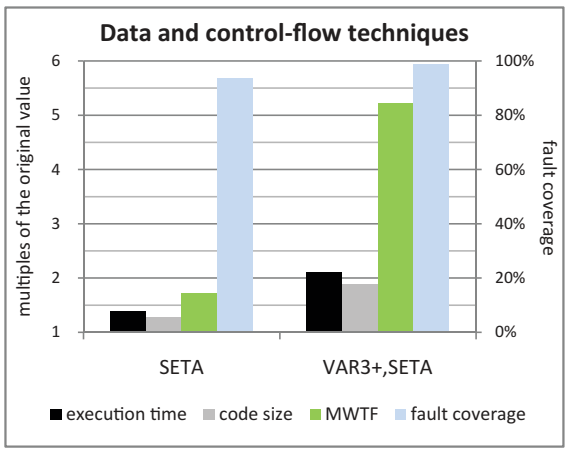

Fig. 4. Comparison between SETA and VAR3+,SETA for miniMIPS processor. Although the increase of the overheads of both techniques together (VAR3+,SETA), there is a significant increase of the MWTF due to the higher fault coverage.

error is signaled when the result stored in the memory differs from the expected one.

D. Simulation Results on MiniMIPS Processor

Nine case-study applications are hardened with SETA and compared to CEDA. They are Bubble Sort, sequential Depth-First Search, recursive Depth-First Search, Dijkstra's Algorithm, Matrix Multiplication, Run Length Encoding, Summation, TETRA Encryption Algorithm (TEA2) and Tower of Hanoi.

Fig. 3 shows the average results for each technique. When CEDA and SETA have one checker per basic block, the fault coverage of both techniques are similar, around 94%. The advantages of SETA are due to its reduced overheads. While CEDA presents execution time of 1.54x and code size of 1.45x, while SETA achieves an execution time of 1.38x and code size of 1.29x. That is the reason why SETA's MWTF is 1.73x while CEDA's is 1.64x.

SETA was combined with VAR3+ in order to protect both data and control-flow of the application. Fig. 4 shows that and compares with SETA only. Although the increase of the execution time from 1.38x to 2.11x, and of the code size from 1.29x to 1.89x, there was an significant increase of the MWTF, from 1.73x to 5.23x, due to the higher fault coverage of almost 99%.

E. Neutron Experiment on ARM Cortex-A9 Processor

Experiments were performed at Los Alamos National Laboratory's (LANL) Los Alamos Neutron Science Center (LANSCE) Irradiation of Chips and Electronics House II, Los Alamos, US, in December 2014 in order to validate the fault injection campaign by simulation. As mentioned in [22], LANSCE provides a white neutron source that emulates the energy spectrum of the atmospheric neutron flux. The relationship between neutron energy and modern devices cross section is still an open question. Nevertheless, LANSCE beam has been empirically demonstrated to be suitable to mimic terrestrial radiation environment [22].

The setup, shown in Fig. 5, consists of a board, computer, USB net switch, cables for communication, and cables for power supply. The computer is connected to the board by two USB cables. One is used to program the board, and the other is used to receive the output from the board. The power supply of the board is connected to the USB net switch, which is connected by USB to the computer. It is used to control when the power supply is available to the board. The board utilized in the tests is the ZedBoar. It is a low-cost development board for the Xilinx Zynq-7000 All Programmable SoC, XC7Z020-CLG484 part, that has embedded a Dual-core ARM Cortex-A9 processor [23]. Only one core was utilized during the test and both caches were enabled. It executed a target application that sends the output by UART to the computer and, then, restarts its execution. The computer was running a monitoring application that listens to the COM port related connected to the board's UART and classifies the output. In case of error in the ARM processor, the processor is reset.

The neutron flux was approximately 1.5×10^6 n/(cm^2 $\cdot$ s) for energies above 10 MeV. The beam was focused on a spot with a diameter of 2 inches plus 1 inch of penumbra, which provided uniform irradiation of the device without directly affecting nearby board power control circuitry. Irradiation was performed at room temperature with normal incidence and nominal voltages.

Two versions of case-study Tower of Hanoi have been tested, one unhardened and the other hardened by VAR3+ and SETA techniques. Table IX summarizes the data from the neutron experiment. The unhardened version was executed for 100 minutes under the beam, receiving a total fluence of 9.0×10^9 n/cm^2 in average. The hardened version was executed for 730 minutes under the beam, receiving a total fluence of 6.57×10^{10} n/cm^2 in average. We observed 6 incorrect executions out of 1557, which results in a SER of 3.854×10^{-3} and a cross-section of 6.667×10^{-10} cm^2 for the unhardened application. In the hardened version, we observed 5 undetected errors that lead to incorrect output on a total of 4872 executions, which results in a SER of 1.026×10^{-3} and a cross-section of 7.61×10^{-11} cm^2. The detection techniques were capable of detecting 90.9% of the errors affecting the processor. That is the reason why we can see a reduction of the SER by 3.76 and of the cross-section by one

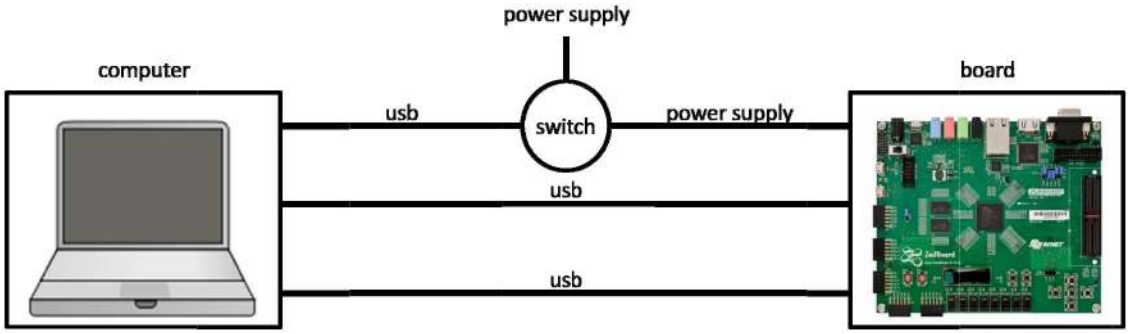

Fig. 5. Setup for radiation test. The computer is connected to the board by two USB cables. One is used to program the board and the other is used to get the output from the board by UART. The computer is also connected by USB to a switch that controls the power supply of the board.

TABLE IX
SUMMARY OF NEUTRON EXPERIMENT ON ARM CORTEX-A9 PROCESSOR

BB Type	Unhardened	Hardened by VAR3+ and SETA
Flux	$1.5x10^{6}$ n/(cm².s)	$1.5x10^{6}$ n/(cm².s)
Time of exposure	100 min	720 min
Fluence	$9.0x10^{9}$ n/cm²	$6.57x10^{10}$ n/cm²
SER	$3.854x10^{-3}$	$1.026x10^{-3}$
Cross-section	$6.67x10^{-10}$ cm²	$7.61x10^{-11}$ cm²
Executions	1557	4872
Execution time	3.85 s	9.00 s
Code size	472 B	1004 B
MWTF	1.00x	1.61x

order of magnitude when hardening using VAR3+ and SETA. However, the execution time of the hardened case-study application used in ARM is 2.33x, and the code size is 2.13x compared to the unhardened application. That results in a MWTF of 1.61x for the hardened application.

The MWTF obtained by simulation on the miniMIPS processor for the Tower of Hanoi hardened by VAR3+ and SETA was 2.68x. The same benchmark, but running on the ARM Cortex-A9 processor and tested under neutrons reached a MWTF of 1.61x. A factor that influenced in this difference is the different processor used in both tests. Thus, the final code and the processor architecture are not the same. Anyhow, it is noticeable an increase of the MWTF from the unhardened to the hardened version.

IV. SELECTIVE HARDENING

A recent approach to reduce overheads caused by software-based techniques consists of applying the techniques selectively. Only selected portions of the application will be protected, not the entire application. Few works based on selective hardening aim to guarantee application-level correctness in multimedia applications [24][25]. For multimedia applications, some errors can be tolerated since they will not be noticed by the user [26]. However, in critical systems, correctness is required. Recent works on this field have been proposed by [27]. In this paper, subsets of the registers used by the application were protected by data-flow techniques and evaluated.

With regards to control-flow techniques, the selective hardening can be applied to basic blocks. In this paper, we do it using the SETA technique. The selective hardening of SETA is implemented in two different ways, as summarized in Table X.

- **SETA-C (SETA minus Checkers)**: consists of removing checkers from the basic blocks, as stated in [16]. All the basic blocks are protected by SETA with signatures. However, not all of them receive a checker. Larger basic blocks have higher priority to receive a checker. If an error occurs in a basic block with no checker, it can be detected in a subsequent basic block since the error will propagate. It presents lower overheads than the standard SETA.
- **S-SETA (Selective SETA)**: is a new selective method. It consists of completely ignoring some basic blocks. The ignored basic blocks receive no signatures or checkers. Thus, it is possible to provide overheads even lower than only removing checkers. This selective hardening method of SETA is better explained below.

A. S-SETA

S-SETA ignores some basic blocks in order to reduce costs. This method was named as tunnel effect. It creates the effect of a tunnel between the predecessors and the successors of ignored basic blocks. Thus, S-SETA does not see ignored BBs and does not protect them. As in SETA-C, larger basic blocks have higher priority to be selected and, thus, protected. The size was selected as criterion because very small basic blocks are quickly executed and, therefore, are less vulnerable. If they are executed just a few times, they would not be very sensitive, so its protection is not very important. And if they are frequently executed, the insertion of protection in such small basic blocks would cause significant performance degradation. Fig. 6 shows how the tunnel effect is applied to a program. Fig. 6(a) presents the default program flow where all the basic blocks have been protected. If the protection is reduced to 70%, as shown in Fig. 6(c), basic blocks 1, 4, 8 and 9 are removed. The successors of BB 1 are attributed to its predecessor, BB 0. The successors of BB 2 now are BBs 3, 5 and 6, since BB 4 was removed. BBs 5 and 6 now point to BBs 2 and 7 instead of BB 1. Furthermore, BB 8 was removed. Therefore, BB 9 has no longer a successor. Following the same idea, Fig. 6(b), Fig. 6(d) and Fig. 6(e) show how S-SETA sees

TABLE X
APPROACHES FOR SELECTIVE HARDENING IN CONTROL-FLOW TECHNIQUES

S-SETA	SETA-C
• Protect only selected basic blocks with signatures and checkers • Other basic blocks are ignored	• Protect all basic blocks with signatures • Only insert checkers in selected basic blocks

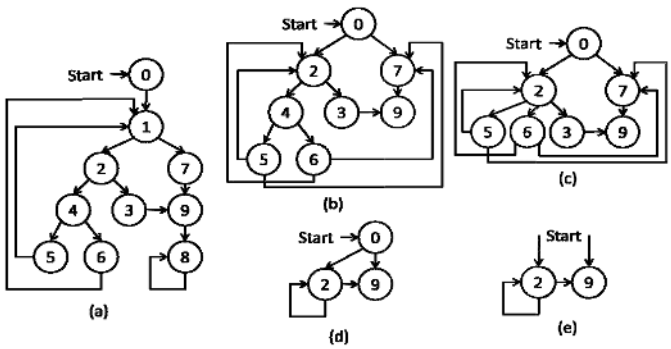

Fig. 6. Example of tunnel effect (S-SETA) (a) protecting 100% of BBs, equivalent to SETA, (b) protecting 80%, (c) protecting 70%, (d) protecting 30% and (e) protecting 20% of BBs. (a) protecting 100% (b) protecting 80% (c) protecting 70% (d) protecting 30% (e) protecting 20%.

the program flow for protecting of 80%, 30% and 20% of basic blocks, respectively.

B. Results

The same nine case-study applications used to compare SETA with CEDA were used to evaluate the two approaches for selective hardening. We have tested all the possible percentage of BBs protected. Fig. 7 shows the results for S-SETA and SETA-C in which the percentage of BBs protected provides the highest MWTF. That shows the maximum improvement we can get from selective hardening. CEDA and SETA with no selective hardening are also included for comparison. As one can notice, both selective approaches reduce the overheads while keeping the a similar fault coverage. While SETA with no selective hardening presents execution time of 1.38x and code size of 1.29x, SETA-C presents 1.28x and 1.21x of execution time and code size, respectively. S-SETA reduces even more the overheads; it presents both execution time and code size of 1.17x. Due to its lower overheads, S-SETA achieves the highest MWTF, 1.94x while SETA-C presents 1.87x, and SETA with no selective hardening reaches 1.73x. By comparing with CEDA the combination of the proposed control-flow technique with the proposed selective hardening method, S-SETA, we can see a reduction in the execution time from 1.54x to 1.17x, code size from 1.45x to 1.17x, and an increase of the MWTF from 1.64x to 1.94x.

We can see the same behavior when VAR3+ is included (Fig. 8). Reduction of the overheads when compared to VAR3+,SETA (no selective hardening), similar fault coverage and, consequently, higher MWTF. VAR3+,SETA presents 2.11x of execution time, 1.89x of code size and 5.23x of MWTF. SETA-C reduces the execution time to 2.05x and the

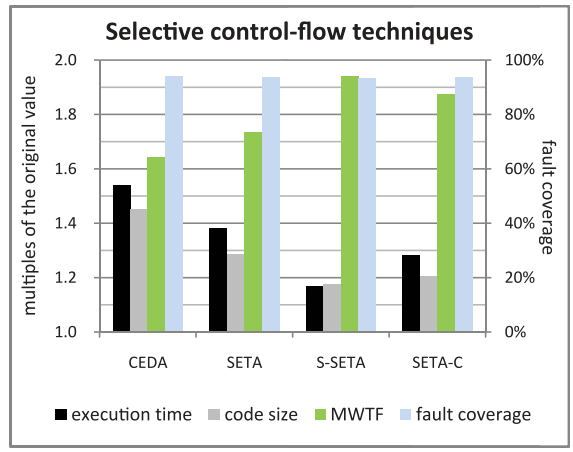

Fig. 7. Comparison between CEDA, SETA, S-SETA and SETA-C for miniMIPS processor. Both selective approaches present similar fault coverage to SETA with no selective hardening. And both reduce overheads. However, the overhead reduction is higher in S-SETA, which justify its higher MWTF.

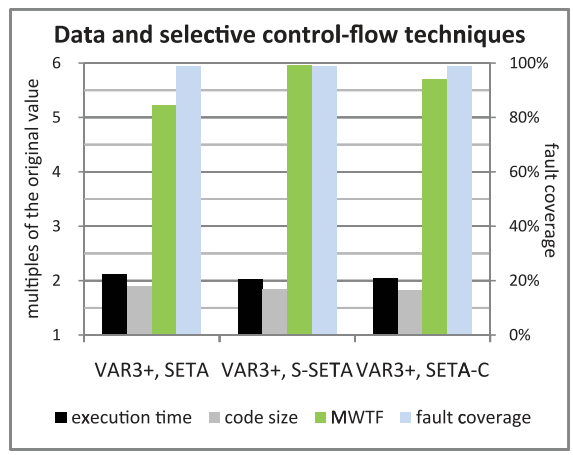

Fig. 8. Comparison between selective control-flow techniques applied together with VAR3+ data-flow technique for miniMIPS processor. The highest MWTF is presented by the proposed VAR3+,S-SETA due to its lower overheads.

code size to 1.82x and increases the MWTF to 5.69x. S-SETA goes even further, it reduces the execution time to 2.01x and code size to 1.84x and achieves a MWTF of 5.96x.

By only removing checkers from the basic blocks (SETA-C), the reduction in the overheads is not that expressive if compared to S-SETA, which clearly presents better gains in the MWTF. Thus, S-SETA is a better option, which is shown by its higher MWTF. An interesting approach to a future work would be applying both solutions together, using S-SETA, and then, inserting checkers only in some of the basic blocks protected by S-SETA.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced SETA, a new control-flow technique. The aims were to keep a similar fault coverage of state-of-the-art techniques and reduce the overheads. SETA does exactly that. It keeps the fault coverage and reduces the overheads, which impacts in an increase of the MWTF. Thus, it is possible to say that SETA is also more reliable since it provides the same fault coverage, and the application will be exposed for a shorter time. The execution time and code size were reduced from 1.54x

to 1.38x, and from 1.45x to 1.29x, respectively, when compare to the state-of-the-art. It increases the MWTF from 1.64x to 1.73x.

To go further with the reduction of the overheads, selective hardening was applied to SETA using an approach stated in the literature (SETA-C) and a proposed new approach (S-SETA). Both approaches for selective hardening reduce the overheads and keep similar fault coverage, which increase the MWTF. This fact is more noticeable in the proposed S-SETA. While SETA-C reduces the execution time from 1.38x to 1.28x and the code size from 1.29x to 1.21x, S-SETA reduces both the execution time and code size to 1.17x. That explains S-SETA's MWTF of 1.94x while SETA-C presents 1.87x.

As a future work, we intend to apply the selective hardening to VAR3+ data-flow technique, by selecting the registers that will be hardened. Thus, we can also reduce the overheads caused by the data-flow technique.

References

[1] N. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. Hu, M. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *IEEE Computer*, vol. 36, no. 12, pp. 68–75, Dec. 2003.

[2] S. Thompson *et al.*, "In search of forever: Continued transistor scaling one new material at a time," *IEEE Trans. Semicond. Manuf.*, vol. 18, no. 1, pp. 26–36, Mar. 2005.

[3] R. Baumann, "Soft errors in advanced semiconductor devices-part I: The three radiation sources," *IEEE Trans. Device Mater. Rel,*, vol. 1, no. 1, pp. 17–22, 2001, Los Alamitos, USA.

[4] "Design," in *International Technology Roadmap for Semiconductors*. Washington DC: Semiconductor Industry Assoc., 2005, pp. 6–7.

[5] O. S. Unsal, I. Koren, and C. M. Krishna, "Towards energy-aware software-based fault tolerance in real-time systems," in *Proc. Int. Symp Low Power Electronics and Design*, 2002, pp. 124–129.

[6] A. Mahmood and E. McCluskey, "Concurrent error detection using watchdog processors–a survey," *IEEE Trans. Computers*, vol. 37, no. 2, pp. 160–174, Feb. 1988.

[7] S. C. Asensi, A. M. Alvarez, F. R. Calle, F. R. Palomo, H. G. Miranda, and M. A. Aguirre, "A novel co-design approach for soft errors mitigation in embedded systems," *IEEE Trans. Nucl. Sci.*, vol. 58, no. 3, pp. 1059–1065, Jun. 2011.

[8] T. Yao, H. Zhou, M. Fang, and H. Hu, "Low power consumption scheduling based on software fault-tolerance," in *Proc. 9th Int. Conf. Natural Computation*, 2013, pp. 1788–1793.

[9] I. Assayad, A. Girault, and H. Kalla, "Tradeoff exploration between reliability, power consumption and execution time," presented at the 30th Int. Conf. Computer Safety, Reliability and Security, 2011.

[10] E. Chielle, F. L. Kastensmidt, and S. Cuenca-Asensi, "A set of rules for overhead reduction in data-flow software-based fault-tolerant techniques," in *FPGAs and Parallel Architectures for Aerospace Applications*, F. Kastensmidt and P. Rech, Eds. Berlin, Germany: Springer, 2015.

[11] O. Goloubeva, M. Rebaudengo, M. S. Reorda, and M. Violante, *Software-Implemented Hardware Fault Tolerance*. Berlin, Germany: Springer, 2006.

[12] N. Oh, P. P. Shirvani, and E. J. McCluskey, "Error detection by duplicated instructions in super-scalar processors," *IEEE Trans. Rel.*, vol. 51, no. 1, pp. 63–75, Mar. 2002.

[13] J. R. Azambuja, A. Lapolli, M. Altieri, and F. L. Kastensmidt, "Evaluating the efficiency of software-only techniques to detect SEU and SET in microprocessors," in *Proc. IEEE Latin Am Symp. Circuits and Systems*, 2011, doi: 10.1109/LATW.2011.5985914.

[14] N. Oh, E. Shirvani, and E. McCluskey, "Control-flow checking by software signatures," *IEEE Trans. Rel.*, vol. 51, no. 2, pp. 111–122, Mar. 2002.

[15] O. Goloubeva, M. Rebaudengo, M. S. Reorda, and M. Violante, "Soft-error detection using control flow assertions," in *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 2003, pp. 581–588.

[16] R. Vemu and J. A. Abraham, "CEDA: Control-flow error detection using assertions," *IEEE Trans. Computers*, vol. 60, no. 9, pp. 1233–1245, Sep. 2011.

[17] J. R. Azambuja, M. Altieri, J. Becker, and F. L. Kastensmidt, "HETA: Hybrid error-detection technique using assertions," *IEEE Nuclear and Plasma Sciences Society*, vol. 60, no. 4, pp. 2805–2812, Aug. 2013.

[18] G. A. Reis, J. Chang, N. Vachharajani, S. S. Mukherjee, R. Rangan, and D. I. August, "Design and evaluation of hybrid fault-detection systems," in *Proc. Int. Symp. Computer Architecture*, Jun. 2005, pp. 148–159.

[19] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proc. Annu . IEEE/ACM Int. Symp. Microarchitecture*, 2003, pp. 29–40.

[20] L. M. O. S. S. Hangout and S. Jan, in *The minimips project 2009*, Oct. 2010 [Online]. Available: http://www.opencores.org/projects.cgi/web/minimips/overview

[21] Mentor Graphics. ModelSim, 2010 [Online]. Available: http://www.model.com/content/modelsim-support, 2010

[22] M. Violante, L. Sterpone, A. Manuzzato, S. Gerardin, P. Rech, M. Bagatin, A. Paccagnella, C. Andreani, G. Gorini, A. Pietropaolo, G. Cardarilli, S. Pontarelli, and C. Frost, "A new hardware/software platform and a new 1/E neutron source for soft error studies: Testing FPGAs at the ISIS facility," *IEEE Trans. Nucl. Sci.*, vol. 54, no. 4, pp. 1184–1189, Aug. 2007.

[23] A. ZedBoard, in *featuring the Zynq-7000 All Programmable SoC 2015*, May 2015 [Online]. Available: http://www.em.avnet.com/en-us/design/drc/Pages/Zedboard.aspx

[24] J. Cong and K. Gururaj, "Assuring application-level correctness against soft errors," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, 2011, pp. 150–157.

[25] A. Sudaram, A. Akael, D. Lockhart, D. Thaker, and D. Franklin, "Efficient fault tolerance in multi-media applications through selective instruction replication," in *Proc. Workshop Radiation Effects and Fault Tolerance in Nanometer Technologies*, 2008, pp. 339–346.

[26] T. Y. Yeh, G. Reinman, S. J. Patel, and P. Faloutsos, "Fool me twice: Exploring and exploiting error tolerance in physics-based animation," *ACM Trans. Graphics*, vol. 29, no. 5, pp. 1–11, 2009.

[27] F. Restrepo-Calle, A. Martinez-Alvarez, S. Cuenca-Asensi, and A. Jimeno-Morenilla, "Selective SWIFT-R," *J. Electron. Test*, vol. 29, pp. 825–838, 2013.