A Novel Phase-based Low Overhead Fault Tolerance Approach for VLIW Processors

Anderson L. Sartor[1], Arthur F. Lorenzon[1], Luigi Carro[1], Fernanda Kastensmidt[1], Stephan Wong[2] and Antonio C. S. Beck[1]

[1] Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS), Brazil
{alsartor, aflorenzon, carro, fglima, caco}@inf.ufrgs.br
[2] Computer Engineering Laboratory, Faculty of EEMCS, Delft University of Technology, The Netherlands
j.s.s.m.wong@tudelft.nl

***Abstract*— Because of technology scaling, the soft error rate has been increasing in digital circuits, which in turn affects system reliability. Therefore, modern processors, including VLIW architectures, must have means to mitigate such effects to guarantee reliable computation. In this scenario, our work proposes two new low overhead fault tolerance approaches, with zero latency detection, that correct soft errors in the pipelines of a configurable VLIW processor. Each approach has a distinct way to detect errors, but they both utilize the same rollback mechanism. The first utilizes redundant hardware by having specialized duplicated pipelines. The second uses idle issue slots to execute duplicated instructions and does this by first identifying phases within an application. Our implementation does not require changes to the binary code and has negligible performance losses. It has 50% of area overhead with 35% power dissipation for the full pipeline duplication, and only 7% of extra area when using idle resources. We compared our approach to related work and demonstrate that we are more efficient when one considers the area, performance, power dissipation and error coverage altogether.**

Keywords—fault tolerance; VLIW; soft errors; configurable processor

I. Introduction

Technology scaling has been allowing high logic integration and performance improvements of integrated circuits, as higher frequencies can be achieved. However, their reliability is compromised as they get more susceptible to soft errors [1]. Soft errors affect processors by modifying values stored in memory elements (such as pipeline registers, register files, and control registers) and are caused by numerous energetic particles such as protons and heavy ions from space or neutron and alpha particles at ground-level. In order to harden the processor against soft errors, fault-tolerant techniques implemented in hardware and software are mandatory to detect those errors and correct their effects before a failure in the system is observed [2].

Very Long Instruction Word (VLIW) processors are representative examples of current architectures that may suffer from the aforementioned issues. VLIW processors exploit Instruction-Level Parallelism (ILP) by means of a compiler, executing several operations per cycle depending on the processor's issue width and the intrinsic ILP available in the application. They occupy less area and dissipate less power when compared to superscalar processors, because their hardware is much simpler, e.g., the instruction queue, reorder buffer, and dependency-checking hardware are not needed.

In this paper, we propose two approaches for detecting and correcting soft errors in VLIW pipelines. The first approach is the full duplication based on a 4-issue configuration, with four main pipelines and four duplicated ones. The other is a phase-based configurable mechanism implemented in hardware that uses idle issue slots to execute duplicated instructions based on the application profiling. It is implemented on an 8-issue processor that can have a variable number of duplicated instructions depending on the application phase: between zero (no fault tolerance) and four (full duplication). These techniques are based on a modified dual modular redundancy (DMR) with instruction rollback mechanism, called DMRr.

In order to validate our proposed techniques, a fault injection campaign was performed in several benchmarks on four configurations of the VLIW processor: unprotected 4-issue, 4-issue with DMRr, unprotected 8-issue, and configurable DMRr 8-issue. We evaluated the results considering several axes, such as error coverage, area, power dissipation, code size, and performance. We also compared our approach to others, showing that we are more efficient when one considers these axes altogether.

The remainder of this paper is organized as follows. Section II presents the fault tolerance techniques for VLIW processors proposed in this work. Section III evaluates the implementation and shows the results. Section IV discusses related works, while Section V concludes this work.

II. Proposed Pipeline DMR with Instruction Rollback

The focus of this work is to protect the pipelines of a VLIW processor against soft errors. The pipelines occupy about 45% of the total area and the register file the other 55% (on the tested 4- and 8-issue configurations); however, the register file can be protected with parity or ECC. The pipelines are duplicated in hardware, and a checker compares the results (i.e., all output signals) of the main and the duplicated pipeline. Therefore, the checker will detect if there is any error in the execution (e.g., error in an arithmetic operation, jump address of a branch, or in the values of a memory operation). The destination register, the register file's and memory's write enable signal are also compared by a checker.

In order to not only detect an error, but also correct it, a rollback mechanism is used. When a mismatch is found in any of the compared signals, the rollback executes the last instruction again, and it flushes the pipeline in order to prevent data corruption. As the checker has zero latency error detection, the memory, and the register file will not be corrupted in case of an error, because the writing to these components will be disabled. The rollback overhead has a fixed cost of 5 cycles (because of the pipeline length of five stages), which is negligible when compared to the total number of cycles of an application. In addition, this cost is only paid in case of an error, having no overhead otherwise. Both the checkers and the rollback mechanism do not affect the critical path of the processor, as they operate in parallel to the pipelines, and the area overhead for each checker is of about 1%.

The VLIW processor used in this work is the ρ-VEX softcore VLIW processor [3], implemented in VHDL. The ρ-VEX core has a five-stage pipeline, and it can be configured to have different number of issue slots (e.g., 2, 4, or 8). Each issue may contain different functional units from the following set: ALU (always present), multiplier, memory, and branch units. In this work, the 4-issue version has 4 ALUs, 2 multipliers, 1 memory unit and 1 branch unit; while the 8-issue has 8 ALUs, 4 multipliers, 1 memory unit, and 1 branch unit, which are similar to other VLIW processors (e.g., Intel Itanium [4]).

As already mentioned, two configurations using the DMR with instruction rollback (DMRr) are evaluated:

A. 4+4-issue

In this configuration, the 4-issue VLIW processor has its pipelines duplicated, and checkers compare the results from the main pipelines to the ones from the duplicated pipelines. Therefore, there is full duplication with dedicated pipelines to execute replicated instructions, as depicted in Fig. 1. This process is totally done in hardware, and the compiler is not aware of it.

B. Phase-configurable

Based on the 8-issue VLIW, idle issue slots during a whole given program phase (i.e., a sequence of instructions words that always have NOPs in specific issue slots) are used to execute duplicated instructions from other pipelines, and their results are checked. Consequently, it is a configurable fault tolerance mechanism, since it may have none (eight issue slots without duplication) to full duplication (four main issue slots and four duplicated ones), depending on the available resources.

The 8-issue configuration for this approach is depicted in Fig. 2. In order to keep the low overhead (area and delay), the duplication pairs are statically placed, i.e., pipeline 0 with pipeline 4, pipeline 1 with pipeline 5, and so on. Therefore, the issue slots are combined in a way that the first four pipelines are compared with the four last ones. For example, if the pipeline 6 is going to execute NOPs in a given phase, then it will execute the duplicated instructions from the pipeline 2 instead. Every functional unit is capable of executing both main program instructions and, in the case of pipelines P4, P5, P6, and P7, duplicated instructions from another pipeline. The exceptions are the memory unit in pipeline 4 and the branch unit in the pipeline 7: these units execute only duplicated instructions, since the ρ-VEX does not support more than one memory or branch operations per cycle. The compiler used in this work (HP VEX compiler) schedules the instructions starting from the lower issue slots (from 0 to 7), considering the availability of the functional units. Therefore, our approach in combining the pairs of the duplicated pipelines will efficiently exploit this scheduling mechanism.

For this approach to work, first, the application must be profiled in order to detect the phases. After that, a table indexed by the program counter and containing the configuration of each application's phase is statically created (a mechanism to dynamically detect phases will be developed in the future). The phase configuration represents the function of each pipeline in a given phase, informing whether each issue slot will execute regular instructions of the application or execute duplicated instructions of another pipeline. Based on this table, the processor will dynamically change the function of the pipelines and will enable or disable the checkers in each phase.

The simulation was conducted in the Mentor Graphic's ModelSim. In this way, it was possible to save which instruction was executed in each cycle to generate the application profile. The benchmark set chosen is composed of the following 10 applications, which comprises a subset of the WCET benchmark suite [5]: ADPCM, CJPEG, CRC, DFT, Expint, FIR, Matrix Multiplication, NDES, Sums (recursively executes multiple additions on an array) and x264. The profiling was performed for all applications from our benchmark set. The results for five benchmarks are depicted in Fig. 3. The dots demonstrate when a given pipeline, identified by its ID (Y-axis), is being used in a

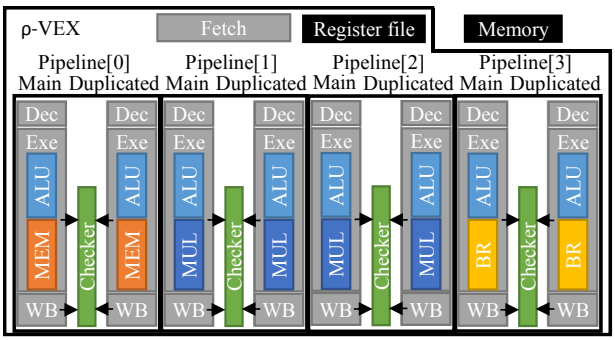

Fig. 1. DMRr 4+4-issue configuration

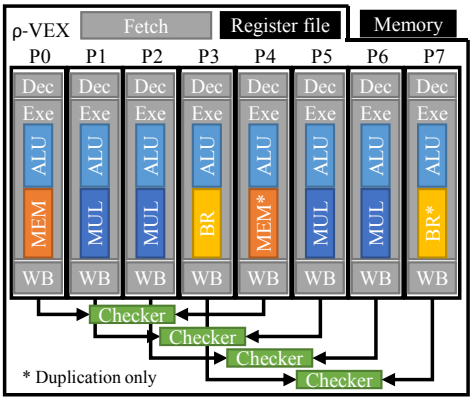

Fig. 2. DMRr phase-configurable duplication

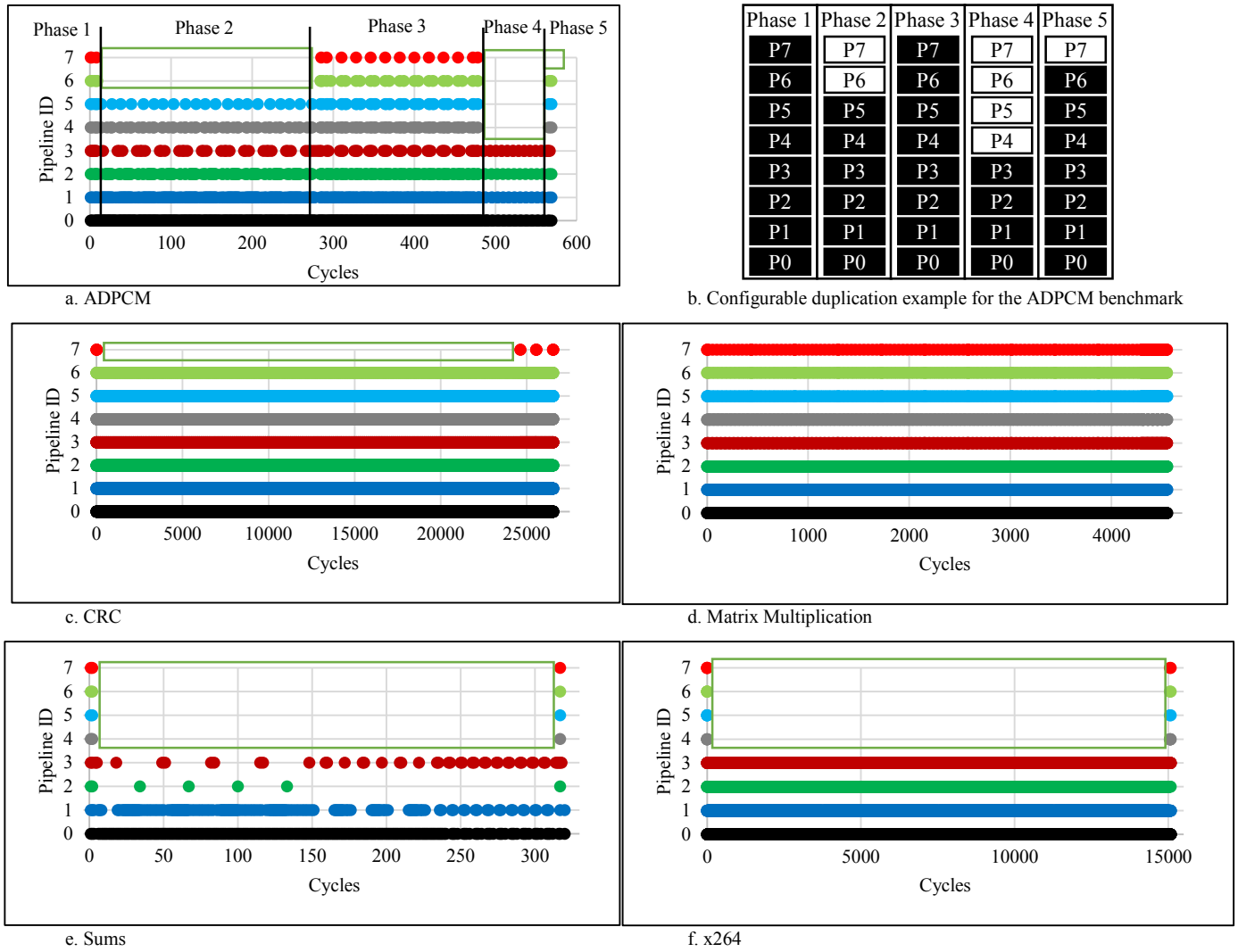

Fig. 3. Issue utilization and configurable duplication

given moment of the application's execution (X-axis). The profiling for the other five benchmarks has a similar behavior to the one from the Matrix Multiplication benchmark.

The idle phases that were used to execute duplicated instructions are highlighted in Fig. 3. An example of each phase for the ADPCM benchmark is depicted in Fig. 3b: the P0-P7 represent the pipelines 0 to 7. The white background on the pipelines indicate that they are executing duplicated instructions from the other pipeline in their pairs (as discussed in Fig. 2), and the black background represents that the pipeline is executing main program instructions (no duplication). On this example, there are phases with full duplication (phase 4), partial duplication (phases 2 and 5) and no duplication (phases 1 and 3). As it can be noticed, the ADPCM, CRC, Sums, and x264 benchmarks have phases when some issue slots are not utilized. On the other hand, as the Matrix Multiplication, CJPEG, DFT, Expint, FIR, and NDES benchmarks do not have such phases, they cannot take advantage of the phase-configurable approach, because it would have the same behavior as the unprotected 8-issue, once no issue slot would be duplicated and compared. Therefore, this second approach, even though it has no costs in terms of performance and negligible power overhead, can be only used when the application has phases with lower levels of ILP than the processor supports.

III. Results

In this section, the results will be discussed regarding the fault tolerance, area, performance, and power dissipation of the proposed mechanism and compared to those from the unprotected processor.

A. *Error Coverage Analysis*

An extended fault injection campaign was performed to inject soft errors in the pipeline, making it possible to evaluate the failure rate of the processor. Scripts in TCL (Tool Command Language) were created to inject transient faults in all pipelines and checkers of the processor. The script chooses a random bit from an arbitrary signal to be flipped at a random time during the execution of the application. The fault duration is one clock cycle, and one fault is injected per application's execution. The total number of injected faults was 2.882.376 (so there was the same number of application executions), varying from 2.750 to 322.495 injected faults per benchmark for each configuration.

The variation in the number of injected faults (executions) per benchmark is due to the difference between their simulation times. Also, the faults were injected randomly in both protected and unprotected versions because their hardware are different. Therefore, it would not be correct to randomly inject a fault in the unprotected version and inject the same fault in the protected version.

There are three possible reasons for failures in the system:

- ***Data failure:*** happens when there is a mismatch between the memory dump from the application and the golden memory dump. The dumps are compared once the application ends its execution.
- ***Data flow failure:*** happens when the application does not stop within the number of cycles that it should (the one from the application without any failures).
- ***Simulation failure:*** happens when some specific signals at specific times are flipped and crashes the simulation, (i.e.: ModelSim's simulation is aborted without finishing the execution of the application).

These failures are checked when the application finishes its execution. Not all injected errors cause failures: some of the errors are application masked, and some of the errors are detected by our proposed technique. Table I presents the failure rate of the chosen applications in all four VLIW configurations. The unprotected 8-issue has a lower failure rate than the unprotected 4-issue due to the elevated number of NOPs in the VLIW instruction; therefore, the probability of a flipping bit affecting the result of an instruction is lower than on the 4-issue configuration. Also, the 4+4-issue decreases the failure rate by 121 times when compared to the 4-issue; and the phase-configurable by 8 times when compared to the 8-issue configuration.

On average, the 4+4-issue has a 99.95% error coverage, while the phase-configurable has 99.55%. The latter has a lower coverage because of the ADPCM and CRC benchmarks. In the former, there are three significant phases in which no duplication can be done because there are no idle issue slots (i.e.: they achieve the maximum ILP possible). In the latter, in most of the times only one issue is idle and available to execute duplicated instructions. For the benchmarks that present a significant phase with four idle issue slots, as the example of Sums and x264 benchmarks, the coverage is close to the one achieved by the full duplication: 99.89% and 99.93%, respectively. Hence, when the application has idle phases to execute duplicated instructions, it is not only possible to provide the same level of fault tolerance as in the case of dedicated duplication; but also, keep the 8-issue's performance, which is better than the one from 4-issue configuration as more functional units are available to exploit the application's ILP.

The distribution of failures for each application running on each configuration is depicted in Fig. 4. Both protected and unprotected versions have a similar probability of simulation failures, because the simulation failure interrupts the execution of the application. Therefore, as the protected version is able to correct most of the errors that may lead to a data or control flow failure, the proportion of simulation failures in the protected configurations is bigger than the one from the unprotected ones.

TABLE I. FAILURE RATE WITH DIFFERENT CORE CONFIGURATIONS

			# Injected faults	# Errors	Failure rate (%)
Unprotected	4-issue	ADPCM	287,398	19,911	6.93
		CJPEG	250,441	23,911	9.55
		CRC	79,346	4,128	5.20
		DFT	52,531	2,432	4.63
		Expint	69,221	2,911	4.21
		FIR	5,649	618	10.94
		Matrix	59,906	5,934	9.91
		NDES	35,651	1,424	3.99
		Sums	322,495	17,814	5.52
		x264	55,502	2,891	5.21
	8-issue	ADPCM	143,077	5,234	3.66
		CJPEG	154,789	9,399	6.07
		CRC	43,444	1,280	2.95
		DFT	32,383	869	2.68
		Expint	53,949	1,281	2.37
		FIR	2,797	166	5.93
		Matrix	60,287	3,423	5.68
		NDES	20,574	430	2.09
		Sums	200,363	5,937	2.96
		x264	28,807	847	2.94
Protected	4+4-issue	ADPCM	183,168	103	0.06
		CJPEG	160,527	28	0.02
		CRC	39,157	22	0.06
		DFT	21,054	15	0.07
		Expint	70,396	37	0.05
		FIR	2,750	1	0.04
		Matrix	33,760	28	0.08
		NDES	17,755	7	0.04
		Sums	239,239	94	0.04
		x264	11,961	11	0.09
	Configurable	ADPCM	36,958	366	0.99
		CRC	16,720	107	0.64
		Sums	65,221	75	0.11
		x264	14,448	10	0.07
Average	Unprotected 4-issue				6.61
	Unprotected 8-issue				3.73
	4+4-issue				0.05
	Configurable				0.45

B. *Area and power dissipation evaluation*

The synthesis tools used to evaluate area, performance and power were: the Xilinx ISE synthesis tool to obtain the FPGA area and frequency to the Virtex 6 - XC6VLX240T FPGA; and the Cadence Encounter RTL compiler to obtain power dissipation and ASIC (Application Specific Integrated Circuit) area, using a 65nm CMOS cell library from STMicroeletronics.

Table II presents the processor area for both FPGA and ASIC and power dissipation of the processor for all VLIW configurations. The operation frequency for these configurations was of 65MHz, and the speed up from the 8-issue configuration when compared to the 4-issue, for the aforementioned benchmark set, varies from 0.1% to 23.6%, with an average of 5.4%.

As it can be observed, the overhead of the full duplication (4+4-issue) is small in terms of area and power dissipation when compared to the 4-issue. The area overhead is of 30% for the FPGA (given in LUTs) and 50% for the ASIC cells, while the power dissipation overhead is of 35%. The overhead is almost

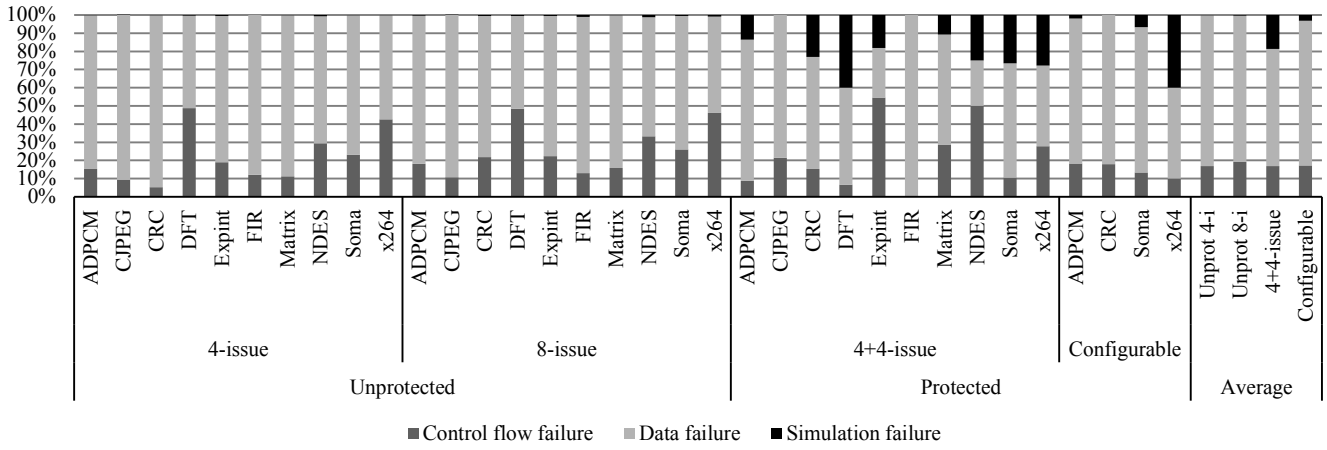

Fig. 4. Failure distribution

TABLE II. AREA AND POWER DISSIPATION COMPARISON

		FPGA		ASIC	
		Registers	**LUTs**	**Cells**	**Power dissip. (nW)**
Unprotected	**4-issue**	3,058	16,006	28,041	2,298,962.51
	8-issue	3,974	35,075	68,977	7,556,643.89
Protected	**4+4-issue**	4,102	20,819	42,121	3,109,613.33
	Configurable	4,133	35,973	73,988	7,484,929.91

negligible when one compares the phase-configurable approach with the base 8-issue configuration: 3% for the FPGA and 7% for the ASIC; with almost no overhead in power dissipation. This overhead comes from the checkers only, since no extra circuitry, such as duplicated functional units, is necessary for the latter approach.

IV. RELATED WORK

Several works have been proposed for the detection and correction of soft errors in VLIW and superscalar processors. These works aim to improve the fault tolerance of the target system, typically based on redundancy, which may be implemented either in software, in hardware, or both.

Dual modular redundancy (DMR) with rollback was used by [6] and [7] to detect and correct errors. These works used checkpoints in order to rollback, whenever an error is detected, to a state in which the execution was correct. Therefore, the latency to detect the error on these approaches will vary according to the periodicity of the checkpoints. On the other hand, the DMRr has zero latency detection as it compares the results at all times and executes again only the instruction that presented the error. Therefore, in addition to the zero latency detection, the control structure of the rollback is much simpler than the ones that use checkpoints.

Another common approach is to triplicate a processor and use a majority voter to vote the correct answer (triple modular redundancy - TMR), as implemented in [8] and [9]. In these cases, they only triplicate the functional units of a VLIW processor rather than the entire processor; therefore, reducing the area and power dissipation costs. In [8], both hardware and software needed to be changed, and if the two operations (main and duplicated) compute different results, the operation is executed a third time. This approach is called Reduced TMR. Both approaches only cover errors that happen in the computation of a given operation, therefore, not begin able to detect an error if the operands were wrong in a stage prior to the execution or after the operation was computed by the functional unit. DMRr has higher coverage than [9], occupying less area and dissipating less power than [8] and [9], and the proposed approach does not change the binary code of the application, as it is done in [8]. In [10], the authors propose a similar approach to [8], with the instruction replication done in software. In the same way, the replication is done partially to some instructions in order to avoid significant performance losses, but it also affects the capacity of providing fault tolerance. Therefore, the code is changed, performance is affected as well as the error coverage, even though there is no area or power overheads. In [11] the authors also propose a TMR approach on the synchronous flip-flops, with an area and power dissipation overhead higher than the one created by the DMRr, even when comparing to the full duplication configuration.

In [12] and [13], the authors propose a software-based redundancy based on duplication with comparison (DWC) for VLIW data paths aiming to reduce the performance overhead by using the idle functional units. However, these techniques still present performance degradation and increase code size, as they are in software. Authors in [14] propose an optimization to the DWC's generated code by reducing the impact of the basic block fragmentation caused by the check instructions, having lower, but still not negligible, performance degradation than the previous two techniques. Authors in [15] propose a compiler assisted multiple word retry scheme for VLIW architectures, and the authors evaluate the performance and code growth of having a rollback mechanism with different number of instructions.

The main limitations of software-based redundancy are the increase in the code size, energy consumption and performance overheads that come with it. On the other hand, hardware-based redundancy approaches increase area, power dissipation with little or no performance overhead. The approach proposed in this paper, even though implemented in hardware, has low overhead in area and power dissipation.

Table III presents the comparison, in several axes, between the results from the DMRr and the other works previously discussed in this section. These axes comprise error coverage, area, performance, power dissipation and code size increase. As it can be noticed, the DMRr has the lowest area and power dissipation overhead when compared to other hardware-based techniques (**in bold**). Software-based techniques (*in italic*) naturally do not affect the area nor the power dissipation, but they create a performance overhead and increase the code size, both affecting total energy consumption of the system, as the application will take longer to execute, and the memory will be more stressed when executing more instructions. In addition, several hardware-based techniques also have performance and/or code size overheads, besides the power and area overheads, impacting even more on the energy consumption of the system.

By using DMRr, on the other hand, the application's code is not changed, the performance overhead is negligible, and the power and area overheads are much lower than the other techniques. Therefore, being able to provide good coverage at a low cost, considering all axes: performance, power dissipation, energy consumption, and area. In addition, for benchmarks that have idle phases, the configurable duplication can be used in order to execute duplicated instructions in the idle pipelines, having only 7% of area overhead and negligible overhead on the other axes.

V. Conclusions and Future Work

The pipelines of a VLIW processor occupy a significant area of the processor core; therefore, it is important to protect them against soft errors with minimum cost while providing good coverage. In this work, a fault tolerance mechanism based on duplication and instruction rollback is proposed, which is able to not only detect a fault, as a conventional DMR approach, but also correct the error by executing the faulty instruction again via rollback. The performance overhead that a rollback causes is negligible (5 cycles) compared to the application's total number of cycles. In addition, this mechanism is able to provide fault tolerance with a minimum cost, by using idle resources of the VLIW processor or by duplicating the pipelines with low area and power overhead.

As future work, we will consider two more processor configurations: a phase-based approach with *dynamic* phase detection and an instruction-adaptive configuration that will duplicate instructions during runtime whenever a pipeline is executing a NOP instruction. In addition, a mechanism to detect permanent faults will be developed.

References

[1] P. Shivakumar, M. Kistler, S. Keckler, D. Burger and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," *International Conference on Dependable Systems and Networks,* pp. 389-398, 2002.

[2] A. C. S. Beck, C. A. L. Lisbôa and L. Carro, Adaptable Embedded Systems, Springer Publishing Company, 2012.

[3] S. Wong, T. v. As and G. Brown, "ρ-VEX: A reconfigurable and extensible softcore VLIW processor," *International Conference on ICECE Technology,* pp. 369-372, 2008.

[4] H. Sharangpani and K. Arora, "Itanium processor microarchitecture," *IEEE Micro,* vol. 20, no. 5, pp. 24-43, 2000.

[5] J. Gustafsson, A. Betts, A. Ermedahl and B. Lisper, "The Mälardalen WCET benchmarks – past, present and future," *International Workshop on Worst-Case Execution Time Analysis (WCET),* p. 137–147, 2010.

[6] R. Xiaoguang, X. Xinhai, W. Qian, C. Juan, W. Miao and Y. Xuejun, "GS-DMR: Low-overhead soft error detection scheme for stencil-based computation," *Parallel Computing,* vol. 41, pp. 50-65, 2015.

[7] J.-M. Yang and S. Kwak, "A checkpoint scheme with task duplication considering transient and permanent faults," *Industrial Engineering and Engineering Management (IEEM), IEEE International Conference on,* pp. 606-610, 2010.

[8] M. Scholzel, "Reduced Triple Modular Redundancy for built-in self-repair in VLIW-processors," *Signal Processing Algorithms, Architectures, Arrangements and Applications,* pp. 21-26, 2007.

[9] Y.-Y. Chen and K.-L. Leu, "Reliable data path design of VLIW processor cores with comprehensive error-coverage assessment," *Microprocessors and Microsystems,* vol. 34, no. 1, pp. 49-64, 2010.

[10] Y. Li, J. Lee, Y. Ko, K. Lee and Y. Paek, "Compiler-directed instruction duplication for soft error detection," *Workshop on Synthesis And System Integration of Mixed Information technologies,* pp. 54-59, 2012.

[11] F. Anjam and S. Wong, "Configurable fault-tolerance for a configurable VLIW processor," *Reconfigurable Computing: Architectures, Tools and Applications,* vol. 7806, pp. 167-178, 2013.

[12] C. Bolchini, "A software methodology for detecting hardware faults in VLIW data paths," *IEEE Transactions on Reliability,* vol. 52, no. 4, pp. 458-468, 2003.

[13] J. Hu, F. Li, V. Degalahal, M. Kandemir, N. Vijaykrishnan and M. J. Irwin, "Compiler-assisted soft error detection under performance and energy constraints in embedded systems," *ACM Transactions on Embedded Computing Systems (TECS),* vol. 8, no. 4, p. 27, 2009.

[14] K. Mitropoulou, V. Porpodas and M. Cintra, "DRIFT: Decoupled CompileR-Based Instruction-Level Fault-Tolerance," *Languages and Compilers for Parallel Computing,* pp. 217-233, 2014.

[15] S.-K. Chen and W. Fuchs, "Compiler-assisted multiple instruction word retry for VLIW architectures," *IEEE Transactions on Parallel and Distributed Systems,* vol. 12, no. 12, pp. 1293-1304, 2001.

TABLE III. VLIW FAULT TOLERANCE TECHNIQUES COMPARISON

Technique	Error Coverage	Area overhead	Performance overhead	Power dissipation overhead	Code size increase
DMRr - 4+4 configuration	~100%	50%	~0%	35%	0%
DMRr - Configurable	~100%	7%	~0%	~0%	0%
DMR with rollback [5] and [6]	~100%	0%	51%-100%	0%	100%
TMR	~100%	200%	~0%	~200%	0%
Partial TMR [8]	95%-99%	100%	0.6%-34.3%	~100%	0%
Reduced TMR [7]	~100%	100%	0%-100%	~100%	> 0%
Reduced TMR - SW [9]	~100%	0%	30%-60%	0%	100%
Flip-flops TMR [10]	~100%	200%	~0%	~200%	0%
DWC - SW [11] and [12]	~100%	0%	28%-106%	0%	109-217%
DWC opt. - SW [13]	~100%	0%	29%	0%	100-150%