# Two Soft-Error Mitigation Techniques for Functional units of DSP Processors

Alireza Rohani
*Testable Design and Testing of Integrated Systems Group*
*CTIT/University of Twente*
Enschede, The Netherlands
a.rohani@utwente.nl

Hans G. Kerkhoff
*Testable Design and Testing of Integrated Systems Group*
*CTIT/University of Twente*
Enschede, The Netherlands
h.g.kerkhoff@utwente.nl

***Abstract*—This paper presents two soft-error mitigation methods for DSP processors. Considering that a DSP processor is composed of several functional units and each functional unit constitutes of a control unit, some registers and combinational logic, a unique characteristic of DSP workloads has been deployed to develop a masking mechanism for the control-logic of each functional unit. Combinational logic has been elaborated with a fast recovery mechanism to isolate the fault-free functional units and re-execute the erroneous instruction. These techniques have been implemented on a DSP processor in order to assess the achieved fault-tolerance versus the imposed overheads.**

## I. INTRODUCTION [1]

Increasingly miniaturized CMOS technologies along with the reduction of operating voltage have made soft-errors a major source of threat for today's digital ICs. The impact of soft-errors on a digital IC can be classified into two categories: Single-Event-Upset (SEU) and Single-Event-Transient (SET). A SEU occurs when a high-energy particle hits a storage element (memory or flip-flop) and consequently changes its stored value. A SET occurs when combinational logic is being hit by a high-energy particle and a momentary pulse will be generated at the output of the strike gate [1] and [2]. Radiation-based experiments [3] show that the length of such a momentary pulse is between 50ps to 150ps, depending on the particle of strike. Historically, memory elements and flip-flops were the point of concern with regard to soft-errors; as a result mature and effective Error Detection And Correction (EDAC) codes were developed to deal with SEUs. In contrast, developing low-overhead mitigation methods for unstructured and irregular parts of a processor such as the control unit is still an open question [4]. The concern of SETs will escalate when the amount of chip area devoted to complex structures will grow with chip complexity. Moreover, increasing the system frequency will cause the system errors to be dominated by the SETs originating from combinational logic rather than SEUs from storage elements [5].

## II. RELATED WORKS

One of the most well-known approaches to eliminate the impact of soft-errors in modern processors is the Checkpoint and Recovery (CR) method in which the current status of the processor is stored in a memory device at various time instances of workload execution (referred to as check-points). Upon soft-error detection, the processor status will be re-loaded with the last saved check-point (referred to as roll-back). Generally, CR-based methods impose a heavy load on the system.

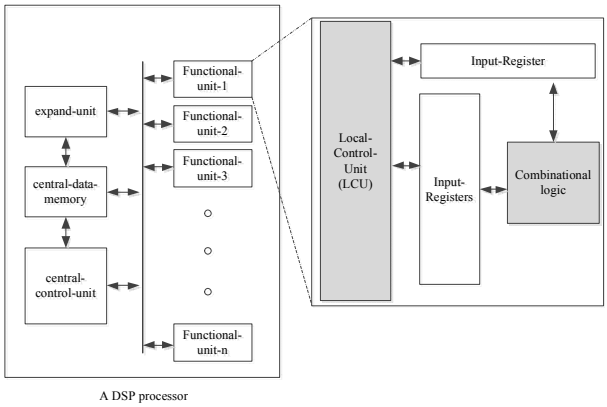


Figure 1. A typical architecture of a DSP processor. The gray parts will be elaborated in this paper

Paper [6] introduces the ReStore architecture, in which the activation of a roll-back is triggered by symptoms such as a high number of cache misses. Reference [4] presents another signature-based CR-based technique for the control logic of MIPS processors. Authors in [7] reconfigure the redundancy of functional units of a DSP processor into a m-way replication, however the execution-time of a program will be duplicated due to assigning some functional units to fault-tolerance. Recently, a hardware\software CR-based scheme, called Reli, has been proposed in [8] which is based on elaborating micro-instructions with additional micro-operations to facilitate check-pointing.

Another category of recovery methods are redundancy-based techniques. Authors in [9] and [10] explored a signature-based caching scheme in which all control signals are integrated into a signature and then they are verified before the commitment stage. The disadvantage of this method lies in the data dependency which can stall the pipeline stages for a long time. VOLTaiRE, is a low-cost fault *detection* solution, proposed in [11] which detects *permanent faults* in the data-path of DSP processors. Two soft-error mitigation schemes, namely Soft-Error Mitigation (SEM) and Soft and Timing Error Mitigation (STEM), using the approach of multiple clocking of data for protecting combinational logic, have been proposed in [12]. While both of these methods can achieve nearly 100% fault coverage, they impose 100% degradation on the performance.

In our paper, we will show that employing the unique characteristics of DSP workloads along with DSP architectures

[1] This research has been conducted as part of the "ELESIS" project (co)financed by the Netherlands Enterprise Agency (RVO).

978-1-4799-3415-7/14/$31.00 ©2014 IEEE

can result in a method that benefits from the advantages of redundancy-based methods (short detection latency) and CR-based methods (low performance degradation).

## III. OUR CONTRIBUTIONS

This paper proposes a new architecture for DSP processors by developing two architectural mechanisms to mitigate SETs in functional units of a DSP processor. These mitigation mechanisms have been developed based on employing the unique characteristics of DSP workloads as well as DSP architectures. As depicted in Figure 1, a DSP processor constitutes of several functional units that execute a Very-Long-Instruction-Word (VLIW) instruction in parallel. Each functional unit is composed of several input-registers, a Local-Control unit (LCU) and combinational logic. Considering the fact that input-registers can be protected by readily-available EDAC codes, a SET masking method has been developed for the LCUs. A SET recovery mechanism has been designed for the combinational logic. In order to protect the LCUs, the control signals have been classified into either opcode-dependent or instruction-dependent, based on their changeability over time during execution of an instruction. The opcode-dependent control signals have been replaced by a ROM memory, acting as a look-up table. To protect instruction-dependent control signals, an inherent characteristic of DSP workloads, the locality of references, has been employed. Combinational logic inside of each functional unit has been enriched by shadow registers which make it feasible to re-execute a very fine-grained part of an instruction while the rest of the processor is waiting, the so called freezing. Experimental results show that the masking method in LCUs imposes 4% increase in silicon-area and 10% degradation in performance, while the percentage of induced failures drops from 40% to 5.4%. The enriched combinational logic imposes 10% overhead in area and causes no degradation on the performance.

## IV. SET MASKING IN LCUS

The mechanism of masking SETs in the LCUs is based on classifying the control signals of each functional unit to either *opcode-dependent* or *instruction-dependent* signals. As can be seen in Figure 2, the value of *opcode-dependent control signals* depends *only* on the opcode part of an instruction. In contrast, the value of *instruction-dependent control signals* depends on the whole instruction and not only on the opcode part. The following subsections present two different masking mechanisms for each category.

### *A. ROM-based Masking Approach*

Since the value of opcode-dependent control signals depends *only* on the opcode part of an instruction, and the number of possible opcodes are limited per functional unit, a *distributed* ROM memory has been deployed to store the value of the opcode-dependent control signals for each opcode. The term *distributed* implies that each pipeline stage can access this ROM unit. A limited number of different opcodes per functional unit (32 different opcodes per functional unit in our case study) along with a limited number of opcode-dependent control signals make it feasible to store the value of these control signals for each opcode in a ROM memory during the design phase and then retrieve them during run time. The

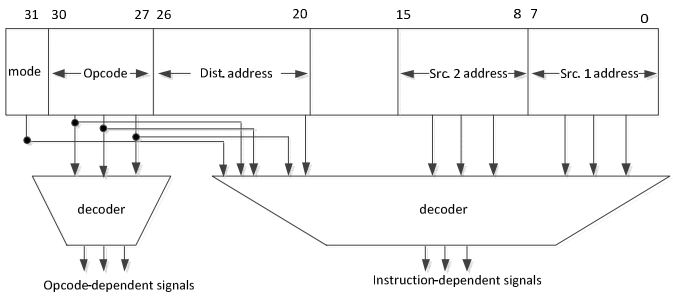


Figure 2. opcode- and instruction-dependent control signals

organization of this ROM memory consists of several entries (equal to the number of different opcodes/per functional unit) and the expected value of their control signals as the contents. In order to retrieve the value of a particular control signal during run-time, the opcode of a fetched instruction is converted into an input-address for the ROM memory in which the expected value of a particular control signal has already been stored.

An example of this organization is shown in Figure 3. Suppose that a typical functional unit has four different opcodes, *add=00*, *shift=01*, *multiply=10*, *load=11* and two control signals of which their value depends on the opcodes, named $ALU_{sel}$ and *shift*. The designer can store the value of these control signals per opcode during the design phase. For example, $ALU_{sel}$ is *1* for the *add* and *multiply* opcodes and *0* for the other two opcodes. The *shift* signal is only *1* during the execution of the *shift* operation. Consequently, the ROM structure has four entries (associated with four opcodes) while each entry has a two-bit width representing the contents. The content of this ROM memory is constant independent of the executed workload.

The probability that a SEU/SET can alter the contents of a ROM memory is very low (near 0%) as compared to the traditional structure of control units [13]. Moreover, EDAC codes can be readily used to protect this ROM memory [14]. However, this structure is still vulnerable to SETs (in the input/output lines or in the opcode-to-address convertor). In the experimental results, the efficiency of this method will be assessed. It is important to mention that the value of each opcode-dependent control signal will be generated by this ROM-unit, like a look-up table rather than the LCU of the associated functional unit.

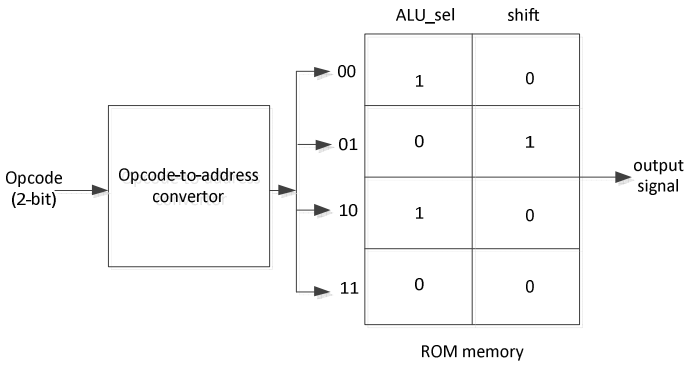


Figure 3. ROM structure to mask soft-errors in the opcode-dependent control signals

### A. Cache-based Masking Method

Another category of control signals is instruction dependent control signals. Since the number of different instructions per functional unit is infinite, the previously introduced look-up table is not feasible in this case. In order to propose a novel masking mitigation method, a common principle in computer architectures, the so-called locality of reference has been employed. This concept implies that most of the program execution time is spent on a small piece of code. Especially for DSP workloads, about 90% of the computational time is spent in a very small kernel [15]. As a result, the *variety* of instructions per workload is limited, however, the exact instructions are not known to the designer before run-time. The idea of our mechanism is to store a history of each instruction-dependent control signal during the first and second executions and subsequently compare the succeeding generated run-time value with the stored history to detect any momentary change. Considering that the values of an instruction-dependent control signal are identical for all the executions of the same instruction, unless an error occurred, this mechanism can detect any singular errors in these signals.

A cache structure has been deployed to implement this mechanism. This cache architecture has several entries which are associated with the number of different instructions within the kernel of the DSP program. The more number of entries in the cache, the more number of different instructions can be tracked. To track each instruction, the unique Program-Counter (PC) can be used. The structure of this cache has been depicted in Figure 4. Suppose that *N* different signals have been classified as *instruction-dependent* control signals. The *PC-to-cache-address-decoder* assigns a unique address in the cache entries to each instruction. The value of the control signals, which has been produced by the conventional LCUs during run-time, are saved in the cache memory during the first and second executions of the kernel of the DSP program. From the third execution afterwards, the run-time value of a signal (which has been generated by the conventional LCUs) will be compared with two previously stored instances. The final output is the majority voter of these three values at any instant of time. Considering that the likelihood of perturbation of two instances of one signal is very small, this scheme can mask the effects of SETs for the associated signals.

The replacement mechanism of this cache structure plays an important role in the efficiency of our mechanism. The random policy replacement was used here to simplify the implementations. Moreover, the number of entries of each cache memory has been bounded to 16, i.e. 16 different instructions can be tracked at any given point of time. A larger cache can protect more signals, however as a trade-off, the complexity of the *PC-to-cache-address-decoder* and the area overhead of the cache structure need to be considered also. Another issue that needs to be addressed here is the controller (or FSM) which is responsible for determining the status of an instruction-dependent control signal with regard to its history. Figure 5 illustrates this FSM that will accompany the scheme depicted in Figure 4.

The complete scheme is depicted in Figure 6. The following paragraph explains this architecture using the tracing of the following simple pseudo-code. Suppose that a control signal *nt*, whose value depends on the whole instruction, should be stored and retrieved via the cache structure. The *nt* signal is

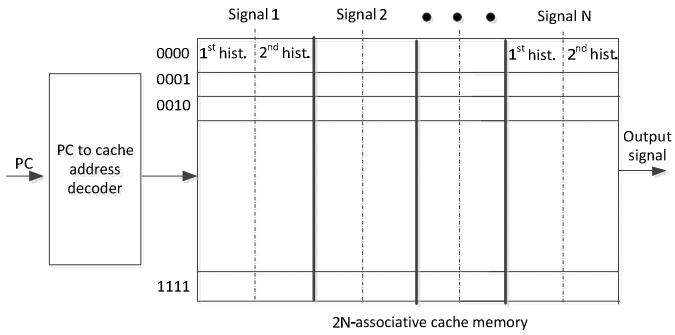


Figure 4. Cache structure to store a history of control signals

| PC | Instruction |
|---|---|
| 0XXXX | beginning of the program |
| 0X0001 | $instr_i$ |
| 0X0010 | $instr_j$ |
| 0X0011 | loop |
| 0X0100 | $instr_1$(the $active_1$ signal is 1) |
| 0X0101 | rest of the loop |
| 0X0110 | end loop |
| 0XXXX | rest of the program |

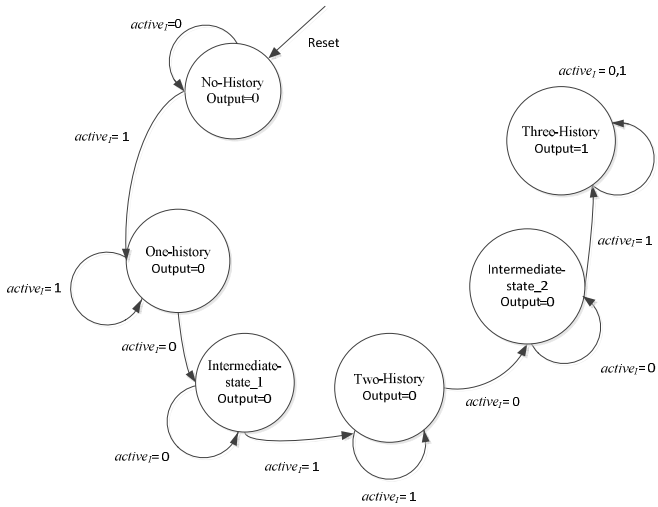


Figure 5. The FSM controller of the cache structure

elaborated during execution of a particular instruction, named $instr_1$ in the above pseudo-code. At the first iteration of the loop, there is no history of *nt*. So the FSM of Figure 5 is in the status labelled *no-history* in which the *output* signal is zero; this subsequently indicates that the value of *nt* is written in the cache memory and is also passed through the rest of the pipeline stages. At this point, an activation signal named $active_1$ is raised which indicates that $instr_1$ has been reached in the program flow. The FSM will move on to the status labelled *one-history* which means that one history of the instruction-dependent control signals associated with the $instr_1$ resides in the cache.

If the program hits the $instr_1$ for the second time, the value of *nt* will be stored as the second history of this signal. Similar to the previous situation, the run-time value of *nt* will be passed to the rest of the pipeline stages (as *output* is still *0*). Activation of the $active_1$ signal for the third time raises the *output* signal and consequently the run-time value of this control signal goes into

the majority voter along with two previous stored values, as shown in Figure 6.

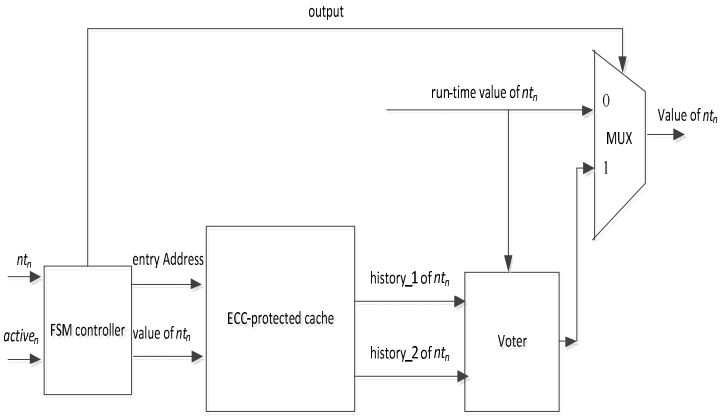


Figure 6. The complete scheme composed of a cache, majority voter and the FSM controller

Even if one instance is corrupted, either one of the stored values or the run-time value, the faulty value will be masked by the majority voter and the fault-free signal will traverse through the rest of the pipeline. The *n* parameter in $active_n$ in Figure 6 depends on how many different instructions exist in a functional unit. This cache structure is a redundant module along with the conventional LCUs of functional units. Furthermore, many readily available mechanisms to harden the cache memory with regard to soft-errors can be used to even make this cache structure more resilient [16].

## V. RECOVERY MECHANISM IN COMBINATIONAL LOGICS

Each functional unit receives its associated opcode from the expand unit and the required data from the data-memory, as already shown in Figure 1. The opcode of the received instruction is decoded by the LCU (or equivalent logic, such as a look-up table) and subsequently the decoded signals will be stored in the associated input-registers. Similarly, the required operands from the memory or register-file will be fetched and stored in their associated input-registers. As long as the data presented in the input-registers are identical at $T_1$ and $T_2$, the output-signal of the combinational logic at $T_1$ and $T_2$ will be identical. The idea of our recovery method is based on accompanying every input-register with one shadow-register in order to hold a copy of the associated data during one consecutive clock-cycle. Since every instruction in a VLIW architecture is distributed over different functional units, it is feasible to halt the *fault-free* functional units and re-execute the faulty instruction in the associated functional unit. To achieve this goal, both the decoded signal received from LCU and the data received from the data memory need to be available for one extra clock-cycle. Upon an error detection, the normal flow of the processor will be halted, and the stored data will be sent to the combinational logic one more time, resulting in one-clock latency on the overall execution time. The limitation of this method is that if two SETs occur at two consecutive clock-cycles, the proposed mechanism will fail to recover the processor. Even though the probability of such an occurrence is very rare, accompanying more shadow-registers per input-register can solve this problem.

There are two possibilities to implement this mechanism. First is to store the value of an input-register at the $i^{th}$ clock-pulse, denoted at $data_i$ in a shadow-register; then upon an error detection, pass the $data_i$ to the combinational logic at the $(i+1)^{th}$ clock-pulse and simultaneously store the new arrived value $data_{i+1}$ (which was supposed to be applied to the combinational logic) in the input-registers. A second implementation is to re fetch the $data_i$ at the $(i+1)^{th}$ clock-cycle and store the $data_{i+1}$ in the shadow-register. The second implementation has been selected in this work and is shown in Figure 7. Referring to this figure, a *detection-signal* will be generated by the combinational logic. This signal can be generated by any mechanism, providing that it detects an error in less than one clock-cycle (a so called zero-latency), such as Duplication With Comparison (DWC). This *detection-signal* will set a *wait-register* that will raise the *wait-signal* during the next consecutive clock-cycle to halt fault-free functional units. These two signals are consistent with the timing diagram which has been shown in Figure 8. As can be seen in Figure 8, upon an error detection in the $(i+3)^{th}$ clock-cycle *tr2*, the input-register will be loaded with the same previous data in the $(i+4)^{th}$ clock-cycle while the original data *input-4*, will be temporary saved in the shadow-register to be passed into the combinational logic during the $(i+5)^{th}$ clock-cycle. The multiplexers *1* and *1'* provide the possibility of loading either a normal value from the data-memory or the value of the last clock-cycle, depending on the value of *detection-signal*. The multiplexers *2* and *2'* provide the possibility of loading the output of an input-register or a shadow-register into the combinational logic, depending on the value of the *wait-signal*. Referring to Figure 8, during the $(i+3)^{th}$ clock-pulse, the *detection-signal* is high (*tr1*), while at the *beginning* of the $(i+4)^{th}$ clock-cycle, the *detection-signal* is high and each input-register will be loaded by its previous value which a faulty results was produced for. During the $(i+4)^{th}$ clock-pulse, *input-3* will be processed again in the combinational logic. At the beginning of $(i+5)^{th}$ clock-cycle, the *wait-signal* is high and the combinational logic will be loaded by the contents of the shadow-register. Considering the experiments carried out in iRoC-Technologies [17] and [3], the duration of SETs is considerably less than one clock-cycle. So, re-executing the faulty instruction after a clock cycle will stop the faulty results to propagate through the rest of the processor.

The main novel feature of the presented recovery method is isolation of the faulty functional unit from the fault-free ones for one clock-cycle, referred to as *freezing*, and re-executing the faulty part of the instruction. Another novel feature is the minimum amount of information needed to be stored in each functional unit; this decreases the recovery overhead to only one clock-cycle, while a typical recovery mechanism takes 16 clock-cycles for the CR-based mechanism [8]. Moreover, the speed of the enriched processor is identical to the performance of the original processor, as long as no SET is present in the system. Furthermore, several clock-cycles are required to store a check-point in the CR-based methods irrespective of the occurrence rate of SETs. Our presented method stores the value of every input-register simultaneously in a shadow-register and as long as no error has been detected by the detection mechanism, the total execution time of a workload is identical by the original or enriched processor.

## VI. EXPERIMENTAL RESULTS

In this section, our results are presented based on the implementation of the described methods in a DSP Xentium

processor, from Recore System [18]. The RTL code of the Xentium processor has been modified such that the LCUs of functional units are enhanced by the method presented in Section IV and the rest of the functional units have been modified based on V.

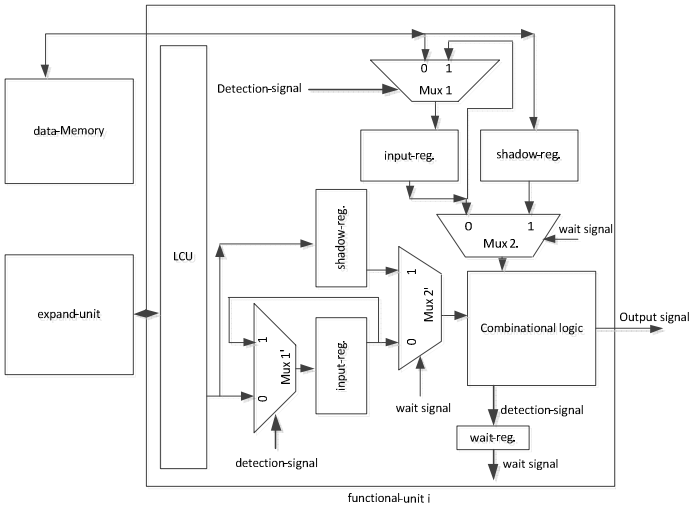


Figure 7. Recovery method in combinational logic part

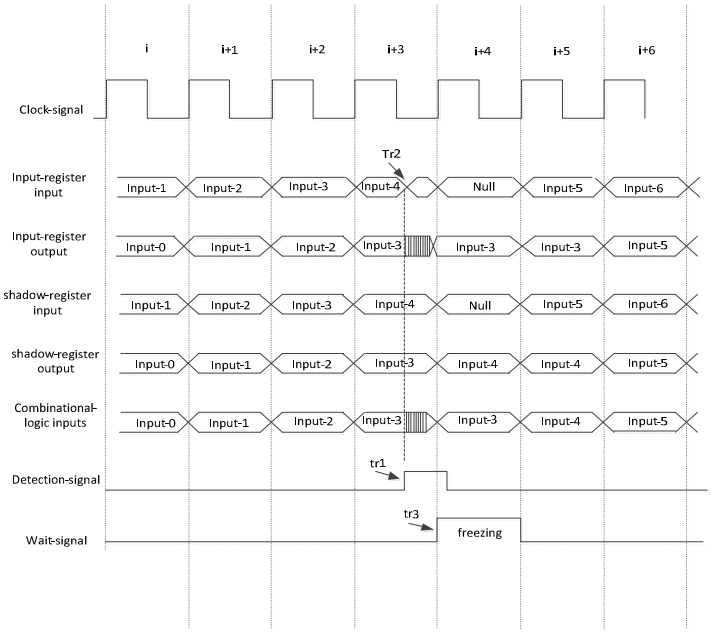


Figure 8. Timing diagram of the recovery mechanism

### A. Area overhead and Performance Degredation

To assess the area overhead and performance degradation induced by the presented methods, a fault-tolerant version of the DSP Xentium processor was developed in the RTL VHDL code. An aggregation test using 20 DSP workloads was carried out to assure its correct functionality. Subsequently, the synthesis tool Synopsys DC was used to synthesize the RTL design using the UMC 90nm technology. The implementation data has been divided into two parts: the area/timing overhead induced by the LCU masking mechanism and the overhead in the combinational logic. Doing so allows us to compare the efficiency of our method with others.

The achieved results are shown in Table 1. The *original Xentium* is the original implementation of the processor. The *FT LCU* is a Xentium processor in which the LCUs of all functional units have been enhanced by the method of Section IV, while the rest of the functional units is identical to the original design. The next two rows show a Xentium processor in which the combinational logics have been modified on the basis of Section V. Moreover, the reported area for the combinational logic has been divided into two parts: including the detection mechanism and disregarding the detection mechanism. As detection mechanism for combinational logic, a partial DWC approach has been employed. For example, for the 32*32-bit multipliers inside a functional unit, two 8-LSB of each input are concurrently multiplied by a smaller redundant multiplier and then the calculated result will be compared to the 8-LSB of the 32*32 bit multiplier, in which any mismatch indicates an error.

Table 1. Area and system clock for the proposed approaches

| | **total cell area ($\mu^2$)** | **area overhead (%)** | **critical-path (ns)** | **performance degradation** |
|---|---|---|---|---|
| original Xentium | 293462 | 0 | 7.87 | 0 |
| FT LCU | 304785 | 4 | 8.70 | 10 |
| FT comb. (including detection) | 341316 | 16 | 7.87 | 0 |
| FT comb. (excluding detection) | 325247 | 10 | 7.87 | 0 |

### B. Achieved Fault tolerance

A simulation-based fault study at the gate-level implementation has been conducted to assess the achieved fault tolerance of the enriched processor. Regarding the simulation-model of SETs, the most recent model of SETs presented in [3] and [19] have been employed. Regarding the workload, a signal processing program, named Finite Impulse Response (FIR) has been used as a workload during fault injection. Using additional DSP workloads were desirable, however the computation time to conduct FIR experiments was already more than several days and involving more workloads was not feasible at this time.

The induced effect of a fault can be classified as *wrong-results*, which means the injected fault has been propagated into the system while *correct-behaviour* indicates the injected fault has been masked before propagating into the system [20]. The behavior of the processor has been indicated in percent, for example if 10 out of 100 fault injections produce *wrong-results*, the sensitivity of the processor is 10%.

The number of fault injections in each set of experiments has been increased from an initial value (200 for LCU and 500 for combinational logic) until a clear convergence can be recognized in the obtained sensitivity level of the processor. The mathematical details of calculating the convergence point is out of the scope of this paper and it has been thoroughly discussed in [19].

Table 2 shows the results of the sensitivity analysis. It can be seen that for the original processor, the percentage of propagated faults in the LCUs is 40% while this number is

30% for combinational logic. The sensitivity of the enriched LCUs has decreased to 5.4% with a detection-latency of 0 clock-cycles (faults will be masked). Further investigation showed that undetected faults have escaped from the detection mechanism as they occurred in the *opcode-to-address-convertor* in the look-up table scheme. For FT-combinational logic, 15% of injected faults could escape from the detection mechanism. However, as long as a fault is detected, the recovery mechanism blocks the fault from propagating through the rest of the processor and recovers the processor within one clock-cycle.

Table 2. Achieved fault tolerance

| | **# fault injections (k)** | | **# wrong answers (k)** | | **Sensitivity (%)** | | **Detection latency (Clk)** |
|---|---|---|---|---|---|---|---|
| | LCU | Comb. logic | LCU | Comb. logic | LCU | Comb. logic | |
| Original xentium | 12.8 | 32 | 5.12 | 9600 | 40 | 30 | N.A |
| FT LCU | 12.8 | 32 | 0.7 | 9600 | 5.4 | 30 | 0 |
| FT comb. logic | 12.8 | 32 | 5.12 | 5100 | 40 | 15 | 1 |

### C. *Comparison of the presented methods with other available mechanisms*

Table 3 shows the comparison of our proposed methods with some available solutions of soft-error mitigation in either LCU or combinational logics. A thorough comparison is very hard as the imposed overhead depends on many parameters such as the exact architecture of the case study, the workloads etc.

Starting with the mitigation methods in the control unit, our work has been compared with [4]. It can be seen that even though the area-overhead of [4] is similar to ours, our method will cause less performance degradation. Comparing our combinational logic mitigation method with one presented in [7], our method is quite competitive in terms of area overhead.

Table 3. Comparison of our method versus other referred methods

| | **Area overhead (%)** | | **Frequency degradation (%)** | |
|---|---|---|---|---|
| | **LCU** | **Comb. logic** | **LCU** | **Comb. logic** |
| Our Method | 4 | 10 | 10 | 0 |
| [4] | 3.4 | N.A | 17 | N.A |
| [7] | N.A | 15 | N.A | 0 |

## VII. CONCLUSIONS

As DSP processors emerge in diverse domains, traditional redundancy methods are not affordable for current-day applications. In this paper two novel solutions to mitigate soft-errors in DSP processors were introduced. The first method employed iterative execution of DSP kernels to organize a look-up table and a cache structure to mitigate soft-errors in the control unit of a functional unit. The second approach benefits from the inherent architecture of DSP processors to isolate faulty functional units from the fault-free ones in order to carry out a fast recovery. Our simulation results showed that the proposed methods are able to successfully mitigate SETs, while their area overhead/performance degradation are better compared to current available methods.


## ACKNOWLEDGMENT

The authors would like to thank G. Rauwerda and S. Baillou of Recore Systems for their valuable suggestions during this work as well as their kind contribution to provide VHDL code of the processor.


## REFERENCES


[1] A. Miele, C. Sandionigi, M. Ottavi, S. Pontarelli, A. Salsano, C. Metra and et al., "High-reliability fault tolerant digital systems in nanometric technologies: characterization and design methodologies," IEEE Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, 2012, pp. 121-125.

[2] A. Eghbal, P. M. Yaghini, H. Pedram and H. R. Zarandi, "Fault injection-based evaluation of a synchronous NoC router," IEEE International On-Line Testing Symposium, 2009. Pp. 212-214.

[3] D. Alexandrescu, E. Costenaro and M. Nicolaidis, "A practical approach to single event transients analysis for highly complex designs," IEEE Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, 2011, pp. 155-163.

[4] H. Ghasemzadeh-Mohammadi, H. Tabkhi, S. G. Miremadi and A. Ejlali, "A cost-effective error detection and roll-back recovery technique for embedded microprocessor control logic," International Conference on Microelectronics, 2008, pp. 470-473.

[5] E. Touloupis, J. A. Flint, V. A. Chouliaras and D. D. Ward, "Study of the effects of SEU induced faults on a pipeline protected microprocessor," IEEE Transaction on Computers, 2007, pp. 1585-1596.

[6] N. J. Wang and S. J. Patel, "ReStore: symptom based soft error detection in microprocessors," IEEE Transactions on Dependable and Secure Computing, vol. 3, no. 3, 2006, pp. 188-201.

[7] Y. Y. Chen, K. L. Leu and C. S. Yeh, "Fault-Tolerant VLIW processor design and error coverage analysis," Conference of Embedded and Ubiquitous Computing, 2006, pp. 754-765.

[8] T. Li, R. Ragel and A. Parameswaran, "Reli: Hardware/software checkpoint and recovery scheme for embedded processors," Design, Automation and Test in Europe, 2012, pp. 857-880.

[9] S. Kim and A. K. Somani, "On-Line integrity monitoring of microprocessor control logic," International Conference on Computer Design, 2001, pp. 314-319.

[10] T. S. Ganesh, V. Subramanian and A. Somani, "SEU mitigation techniques for microprocessor control logic," European Dependable Computing Conference, 2006, pp. 77-86.

[11] S. Shyam, S. Phadke, B. Lui, H. Gupta, V. Bertacco and D. Blaauw, "VOLTaiRE: low-cost fault detection solutions for VLIW microprocessors," Workshop on Introspective Architecture, 2006, pp. 20-27.

[12] N. D. P. Avirneni and A. K. Somani, "Low Overhead Soft Error Mitigation Techniques for High-Performance and Aggressive Designs," IEEE Transactions on Computers, vol. 61, no. 4, 2012, pp. 488-501.

[13] Radiation Effects Mitigation handbook, ESA handbook, 2011.

[14] X. Wendling, R. Rochet and R. Leveugle, "ROM-Based synthesis of fault-tolerant controllers, "Proceedings of the Workshop on Defect and Fault-Tolerance in VLSI System, 1996, pp. 304-308.

[15] G.J. Smit, A. Kokkeler, P. T. Wolkotte, P. K. F Hölzenspies, D. Marcel and P. M. Heysters, "The chameleon architecture for streaming DSP applications," EURASIP Journal on Embedded Systems, 2007, pp. 11.

[16] H. Zarandi, S. G. Miremadi and A. Ejlali, "Dependability analysis using a fault injection tool based on synthesizability of HDL Models," Symposium on Defect and Fault Tolerance in VLSI Systems, 2003, pp. 485-492.

[17] iRoC Technologies, 2012. http://www.iroctech.com/

[18] Recore Systems, 2011, http://www.recoresystems.com/.

[19] A. Rohani, H. G. Kerkhoff, D. Alexanderscu and E. Costenaro, "Pulse-length determination techniques in the rectangular single event transient fault model," International Conference on Embedded Computer Systems: Mirchitectures, Modeling, and Simulation, in Press.

[20] S Mukherjee, "Architecture Design for Soft Errors," ISBN: 978-0-12-369529-1, Morgan Kaufmann Publishers, 2008.