Efficient Protection of the Pipeline Core for Safety-Critical Processor-Based Systems

E. Touloupis, J.A. Flint and V.A. Chouliaras
Department of Electrical and Electronic Engineering, Loughborough University
Email: {E.Touloupis, J.A.Flint, V.A.Chouliaras}@lboro.ac.uk

D.D. Ward
Electrical Group, MIRA Ltd.
Email: david.ward@mira.co.uk

Abstract— **The increasing number of safety-critical commercial applications has generated a need for components with high levels of reliability. As CMOS process sizes continue to shrink, the reliability of ICs is negatively affected since they become more sensitive to transient faults. New circuit designs must take this fact into consideration, and incorporate adequate protection against the effects of transient faults. This paper presents a novel method for protecting the pipelined execution unit of an embedded processor. It is based on a self-configured architecture with hybrid redundancy that can mask single and multiple errors, which can occur on storage elements due to transient or permanent faults. This concept can be easily applied to any processing architecture of this nature with a high safety integrity level. Results from error-injection experiments are also reported that show that this design can maintain a non-interrupted and failure-free operation under single and double errors with a probability that exceeds 99.4%.**

I. Introduction

The advances during the last decades in the development of large scale integrated circuits have encouraged the development of many computer-based systems and sub-systems in commercial products. A typical example is the increased amount of integrated electronics in modern road vehicles over the last few years. As this trend continues, electronic control is taking over safety-critical functions leading to the development of full authority drive-by-wire systems [1]. These follow the same concept as electronic fly-by-wire systems employed in the aerospace industry. However, common practices followed by the latter in order to guarantee high levels of reliability (e.g. the use of radiation-hardened components and large scale redundancy) cannot be applied to the automotive and other commercial sectors that require low-cost solutions due to large production volumes. A solution to this problem is the use of system-on-chip (SoC) fault-tolerant designs [2] that can also help in significantly reducing the component count.

There are a number of reliability issues associated with systems-on-chip and electronic systems in general. From the hardware point of view the most important is the sensitivity of ICs to radiation-induced soft errors. Radiation sources that can affect digital circuits include alpha particles from radioactive impurities contained in IC package materials, and high-energy neutrons from cosmic rays which constantly bombard the earth [3]. These errors have been studied thoroughly in the last few years and until recently the components that were considered to be most susceptible were memory devices [4]. However, the continued shrinkage of CMOS process feature sizes, along with lower voltages and higher operating clock frequencies, increase the occurrence of transient and intermittent faults that lead to data errors [5]. This has been demonstrated in experiments on commercial microprocessors [6]. Furthermore, electromagnetic disturbances within the chip, such as crosstalk, are becoming more significant due to the smaller distances between interconnects [7]. Recent experiments show that the single-event upset (SEU) rate that is caused by radiation increases with increasing clock frequency, since transients on combinational logic can overlap clock edge transitions [8]. It is therefore important that circuit designs use intelligent techniques in order to enable SoCs to tolerate these errors.

Previous research has focused on developing cost-effective solutions for the protection of memory blocks from soft errors, trying to achieve minimal performance penalty in the process. Typical examples can be found in many fault-tolerant computer systems (e.g. [7], [9], [10]) and vary from simple parity bits to long error detection and correction (EDAC) codes [11]. Fewer examples can be found on the protection of other parts of a processor-based system, such as control logic [7], [9], [12], [13] which in most cases don't take into account transient faults in combinational circuits.

This paper presents a novel architecture for protecting the execution pipeline of a RISC processor, typically used in commercial embedded systems and ASICs. The pipeline itself contains a significant amount of non-programmer-visible state and combinational logic and since it is a core part of the processor, it is essential to maintain fault-free operation at any time, especially when used in safety-critical applications. The following sections describe the basic features of this architecture and report some results from the error-injection experiments that have been performed.

II. Proposed pipeline architecture

A. On-chip redundancy

The concept of triple modular redundancy (TMR) was first described by Von Neumann [14] in 1956. Since then, it has been used in many life-critical applications since it allows fault-masking, something that is not possible in redundant systems that only use pairs of components. TMR is considered an expensive solution because it introduces a large hardware overhead. It is however possible to implement a TMR system at a very low level on a single piece of silicon, offering most of its fault tolerant properties on core functionality of the processor. The overhead in this case is not a real issue in state-of-the-art system-on-chip designs. In our work we enhance the attributes of TMR in order to make it more robust against multiple errors. Whilst it is not claimed that a single-chip redundant configuration could replace a similar system based on discrete components, there are some clear benefits in using it as a low-cost, highly reliable unit.

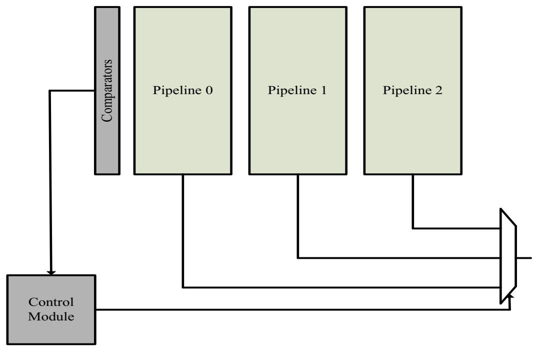

Fig. 1. The triplicated pipeline architecture

The architecture proposed in this paper (Figure 1) is based on an open-source configurable, 32-bit processor that implements the SPARC V8 instruction-set and it is distributed in synthesisable VHDL. The whole pipelined execution unit is triplicated and the three replicas work concurrently, executing the same code on a per-cycle basis, and sharing the same register file, instruction and data cache. All the outputs of the pipelines (data and control signals to register file, cache memories and other parts of the processor) are passed through checking circuits. The checking circuits do not perform a traditional majority vote, but only detect disagreements by producing the following output signals to the control module:

- *dis*: A signal that indicates whether one or more pipelines produces different outputs from the others.
- *dis_vector*: A 3-bit vector where each bit corresponds to one pipeline and becomes ‘high’ when the pipeline in question disagrees with the other two.
- *match_vector*: A 3-bit vector where each bit represents one pipeline pair and becomes ‘high’ when the outputs of the respective pair agree.

Multi-bit signals are not checked on a bit-by-bit basis since this may lead to an error if a particular bit in two of the three modules is erroneous, as has been pointed out in [15].

The control module is a simple state machine that changes the system configuration in the event of a detected error, by checking the above signals. The exact behavior of the system is described in detail in the following section.

B. System behavior

The reliability of a TMR system under the presence of a fault in one of the three modules drops significantly, since an additional fault will result in the generation of an erroneous output. In the case of a triple-pipeline system this scenario

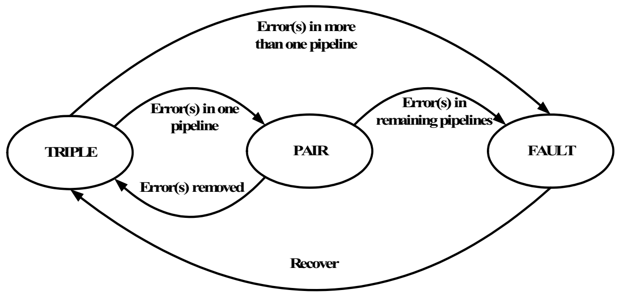
Error(s) in more than one pipeline
Error(s) in one pipeline
Error(s) in remaining pipelines
TRIPLE
PAIR
FAULT
Error(s) removed
Recover

Fig. 2. System’s transitions in the case of single or multiple errors

does not hold because each pipeline replica is a very complex system which makes the probability of having a common-mode error in two of them practically zero. Nevertheless the occurrence of an error could possibly lead to a general system failure, since the affected component will malfunction. For that reason, when an error is detected in our system further action is taken apart from only masking it. The pipeline that produced the faulty output is temporarily disabled by the control module while the remaining two pipelines operate as a self-checking pair [16]. This transition is performed with virtually no delay (one clock cycle) on the processor’s operation, by deactivating all the pipelines for one clock cycle through their hold signal (the hold signal in a RISC processor is used to cease the pipeline’s operation, so that it can be synchronized with the cache memories or co-processor). At the same time the contents of all the registers of the disabled pipeline are updated with the correct data from one of the two operating “healthy” pipelines. When this transfer is completed, the system reintroduces the disabled pipeline on-the-fly, thus switching back to triple mode. If another error is detected while in self-checking pair mode, the system activates a *‘fault’* signal that can optionally be used to generate an exception which can initiate a system restore process (e.g. through a trap). During this mode errors cannot be masked but they can be detected by monitoring the appropriate signal of the *‘match_vector’* that indicates whether the outputs of the remaining pipelines agree.

In the case of multiple errors in more than one pipeline, it is not possible to detect whether any of them is error free, but the presence of errors can still be detected, leading again to the activation of the *‘fault’* signal. The possible system’s transitions are shown in Fig. 2. Overall, this architecture can tolerate any number of errors in one pipeline and can also remain fail-silent in the case of multiple errors in different pipelines. Error-masking and recovery is made with minimal delay making it suitable for hard real-time applications and the cases where the errors cannot be masked, can be handled with many different strategies such as a system reset or a roll-back mechanism. Note that an error on any part of the control logic (including the checking circuits) can only potentially generate unnecessary reconfiguration actions, without any implication on the overall reliability and safety. Even when considering multiple errors in both the pipelines and the control logic, the likelihood of the system not detecting the erroneous condition is extremely low. This is a unique characteristic of the proposed architecture.

III. Implementation

We implemented both the default, single pipeline configuration as well as the redundant configuration on a high performance, 8-copper layer silicon process from UMC. Both designs were read into Synopsys Design Compiler and we obtained an optimized logical netlist. The netlist was subsequently read into Synopsys Physical Compiler where the instruction cache, data cache and register file RAMs were placed and fixed into place in a Minimum-Physical-Constraints (MPC) flow. The optimized, placed netlist was finally read into Cadence SoC Encounter where power planning and detailed routing took place. Fig. 3 and 4 depict the MPC floorplan and the final routed design of the redundant configuration and Table I shows the statistics of both VLSI macrocells.

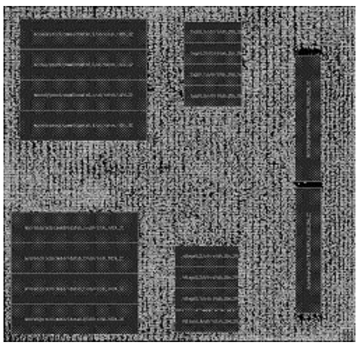

Fig. 3. Floorplan of the redundant configuration

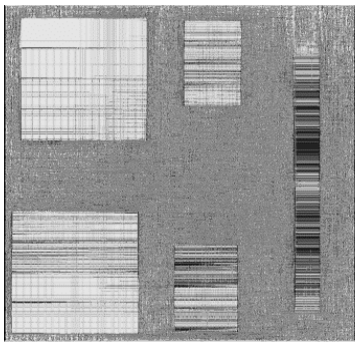

Fig. 4. VLSI macrocell of the redundant configuration

As can be observed in Table I, the fault-tolerant configuration introduces an area overhead of around 26.6% and also a performance penalty in the maximum clock frequency compared to the non-fault-tolerant configuration, for the chosen

TABLE I
Macrocell results

Parameter	Single configuration	Redundant configuration
Instances (Macros)	29132 (18)	71718 (18)
Area (μm^2)	3586685	4539518
Core util.	64.2%	67.4%
F_{max} (MHz)	229.3	174.8

technology. These are expected trade-offs in the design of fault-tolerant systems and their impact must be considered according to the specifications of the targeting application. The following section shows that the reliability and availability levels of this solution are very high.

IV. System validation

A. Error-injection environment

In order to test the system during the design stages and also to validate and demonstrate its error-masking capabilities, error-injection experiments are necessary. The use of software-based error-injection [17], [18] was considered insufficient since the majority of the pipeline registers are non-programmer-visible. Based on the idea of 'saboteurs' [19], error-injection support was implemented through a non-synthesisable, transparent VHDL entity that can have access to all registers and alter their contents at specified times. The error-injection platform is based on a VHDL simulator and its Foreign Language Interface (FLI) capability. The basic steps of an error-injection campaign are the following:

1. Generate a list of errors that define time (clock cycle) and location (register bit) of the injection.
2. Define the number of errors to be injected in each simulation and the total number of simulations for the campaign.
3. Define various parameters that depend on the executed program with a golden run (a simulation without any error injection).
4. Run the simulations
4. Analyse the output data.

The output data consist of information about the correctness of results, the presence of latent errors inside the microarchitecture after the end of the program execution, the status of the processor at the end of the simulation and the execution time. Each simulation falls into one of the following categories:

- No Effect: The program terminates normally, the results are correct and the contents of the pipeline registers and the register file are the same with those of the golden run.
- Latent: The program terminates normally, the results are correct but the contents of the pipeline registers and the register file are not the same with those of the golden run.
- Wrong Result: The program terminates normally but the results are wrong.
- Timed Out: The program failed to terminate within a predefined time limit and the simulation was halted externally.

- Exception: The processor detected the error and created a trap that made it enter in error mode.
- Invalid: The program terminated before reaching its end.

B. Error Model

The error model used in these experiments is a bit-flip of a memory element, which means inverting the logic value from ‘0’ to ‘1’ or from ‘1’ to ‘0’. This model has been traditionally used by other researchers when injecting errors in microprocessor systems. In our error injection platform, errors are injected on the fly while the processor executes the program without stoping the simulation or changing the processor’s mode, and are synchronous with the clock. This means that these errors can be seen as modelling transient faults on combinational circuits during the clock cycle (i-1) that last long enough to be latched on a register at the beginning of cycle (i), or as a direct register hit that occurs during the clock cycle (i), assuming that the error is injected at the beginning of clock cycle (i).

The bit to be affected by the fault is picked randomly. All the registers are grouped according to the pipeline stage they belong to: fetch (fe), decode (de), execute (ex), memory (me), write (wr) and a group of special state/status registers (sregs). We performed error-injections in each group separately and all the bits in each group have equal probability of being selected. This means the probability of a particular register being selected for error-injection depends on the number of bits it contains. The timing of the error-injection is also randomly selected, using a uniform distribution, making sure that the error is injected only when the application program is executed and not during the boot program that is located in the ROM.

C. Results

1) Single errors: We injected 1000 errors in each of the register groups in two processor configurations: a normal, non-fault-tolerant pipeline configuration (referred as ‘Single Core’) and a fault-tolerant version using the architecture described in this paper (referred to as ‘Redundant Core’). The application program used is ‘bitcount’ from the MiBench benchmark suite. The results are depicted in Fig. 5 (the percentages shown are rounded to one decimal place). The ‘Invalid’ category has been omitted from the graphs since it occurred only a few times, and only in the single core.

As expected, the redundant configuration maintains very high levels of fault-tolerance compared to the single. One very important characteristic is that errors are not only detected but also masked without causing any interruptions to the program flow. Simple error detection, which is represented by the “Exception” category, needs the use of a recovery mechanism which introduces delays.

Latent errors in the redundant core are less dangerous because more than 99% of them were observed in the pipeline registers only. If they propagate to the pipeline’s output they will be detected and masked. This means that we can safely add the “Latent” category to “No Effect”, since in both cases

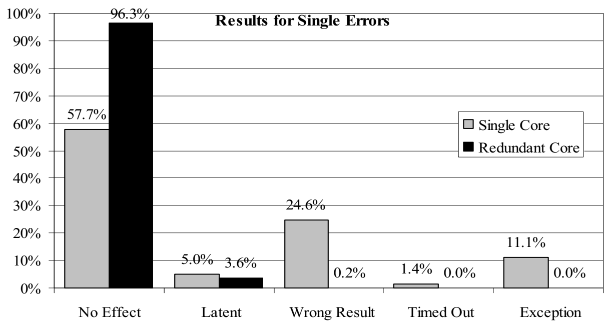

Fig. 5. Processor behaviour for single error-injection

the results produced are correct. In this case the system fault coverage of the redundant configuration reaches almost 99.9%.

The most important class of errors in safety-critical systems are those where wrong results are generated, since they are difficult to detect. As it can be seen in Fig. 5, approximately a quarter of errors in the single core resulted in normal program termination but generated an incorrect result. In the redundant core the equivalent percentage is more than two orders of magnitude lower, although according to the design concept it should be zero. This problem is mainly associated with the instruction cache and errors on the program counter. In particular, errors on low significant bits of the program counter cause an incorrect reference to an instruction that is inside the instruction cache. As a result no cache miss is generated, and the incorrect instruction is passed to the pipeline system in the next clock cycle since the cache memories have a synchronous operation and the transition time lasts only one clock cycle. Similar behaviour is observed in a much smaller extend with the data cache. There are two different solutions to overcome this problem. The first is to extend the transition time to two cycles. The second is the use of a voting circuit for the program counters. The second solution does not have a negative impact on the average delay like the first, but introduces further hardware overhead.

2) Double errors: Under the same conditions described above, we performed more error injection experiments, this time by injecting 2000 double errors in each register group. The results are shown in Fig. 6.

The performance of the single core drops significantly in all categories. Here the probability of a double error generating a wrong result rises to more than one out of three. On the other hand no noticeable difference is observed in the performance of the redundant core, apart from an increase in the occurrence of exceptions and latent faults, which have no implication on the system’s safety. The disproportionate increase in latent errors for the two configurations happens because in the single core configuration, the probability of both errors remaining latent is very small. On the other hand in the redundant configuration this probability is very high since one latent error may occur in one replica and another in a different replica.

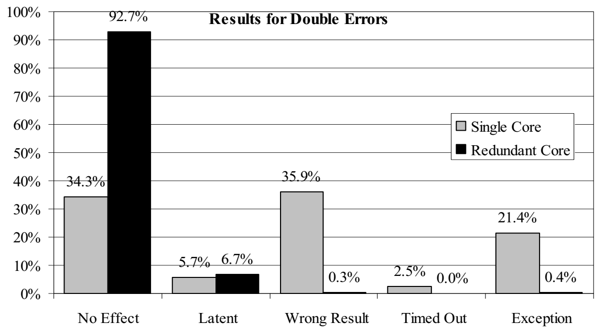

Fig. 6. Processor behaviour for double error-injection

Even if both errors occur in the same replica, the simulation outcome will be a latent error if the first is masked and the second remains latent. Again, by adding the "Latent" category to "No Effect" but also the "Exceptions" that signify error detection but no correction, the system fault coverage of the redundant configuration becomes again roughly 99.8%. If we don't take into account the "Exceptions" category, in the cases of which a timing penalty for recovery actions is introduced, this figure drops only to 99.4%.

V. Conclusion

The soft error rate is expected to significantly increase in near-future CMOS technologies, not only in memory components but also in other parts of logic such as latches, flip-flops and combinational circuits. This paper presents a fault-tolerant architecture that protects the execution pipeline of a RISC processor against soft errors by applying dynamic redundancy techniques that extend the capabilities of a TMR architecture. It is based on triplication with low level voting, and allows dynamic self-reconfiguration in the event of an error in order to maintain high levels of reliability and availability. This configuration can be used in processing units of safety-critical and mission-critical systems. Preliminary results from error-injection experiments confirm the robustness of this architecture against not only single but also double errors.

The paper also highlights the trade-offs between the overhead and the high reliability of this solution by analysing the results of synthesising the design in an ASIC. This architecture is a step towards the implementation of an ultra-reliable microprocessor that can satisfy the very strict failure-rate requirements for high integrity systems, as defined by relevant standards. One big advantage of this design is that it can be easily applied to different processor architectures. Future work will focus in further validation of the system, by performing error-injection experiments with different benchmark programs and fault scenarios through hardware based error-injection on an FPGA prototype.

Acknowledgment

The authors wish to thank the management of MIRA Ltd for funding this research.

References

[1] R. Iserman, R. Schwarz, and S. Stölzl, "Fault-Tolerant Drive-by-Wire Systems," *IEEE Control Systems Magazine*, vol. 22, no. 5, pp. 64–81, Oct 2002.

[2] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vicentelli, M. Peri, and S. Pezzini, "Fault-Tolerant Platforms for Automotive Safety-Critical Applications," in *Proc. of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, 2003, pp. 170–177.

[3] R. Baumann, "Soft Errors in Advanced Semiconductor Devices-Part I: The Three Radiation Sources," *IEEE Transactions on Device and Materials Reliability*, vol. 1, no. 1, pp. 17–22, Mar 2001.

[4] J. F. Ziegler *et al.*, "IBM experiments in soft fails in computer electronics (1978-1994)," *IBM Journal of Research and Development*, vol. 40, no. 1, pp. 3–18, Jan 1996.

[5] C. Constantinescu, "Trends and Challenges in VLSI Circuit Reliability," *IEEE Micro*, vol. 23, no. 4, pp. 14–19, Jul / Aug 2003.

[6] F. Irom, F. Farmanesh, G. Swift, A. Johnston, and G. Yoder, "Single-Event Upset in Evolving Commercial Silicon-on-Insulator Microprocessor Technologies," *IEEE Transactions on Nuclear Science*, vol. 50, no. 6, pp. 2107–2112, Dec 2003.

[7] E. Dupond, M. Nicolaidis, and P. Rohr, "Embedded Robustness IPs for Transient-Error-Free ICs," *IEEE Design & Test*, vol. 40, no. 3, pp. 56–70, May 2002.

[8] F. Irom and F. Farmanesh, "Frequency Dependence of Single-Event Upset in Advanced Commercial PowerPC Microprocessors," *IEEE Transactions on Nuclear Science*, vol. 51, no. 6, pp. 3505–3509, Dec 2004.

[9] J. Gaisler, "A Portable and Fault-Tolerant Microprocessor Based on the SPARC v8 Architecture," in *Proc. of the International Conference on Dependable Systems and Networks*, 2002, pp. 409–415.

[10] N. Quach, "High Availability and Reliability in the Itanium Processor," *IEEE Micro*, vol. 20, no. 5, pp. 61–69, Sep / Oct 2000.

[11] C. L. Chen and M. Y. Hsiao, "Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review," *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 124–134, Mar 1984.

[12] M. Pflanz and H. T. Vierhaus, "Efficient Backup Schemes for Processors in Embedded Systems," *Solid State Electronics*, vol. 44, no. 5, pp. 791–796, May 2000.

[13] L. Spainhower and T. A. Gregg, "IBM S/390 Parallel Enterprise Server G5 Fault Tolerance: A Historical Perspective," *IBM Journal of Research and Development*, vol. 43, no. 5/6, pp. 863–873, Sep / Nov 1999.

[14] J. V. Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," in *Automata Studies*. Princeton University Press, 1956.

[15] S. Mitra and E. J. McCluskey, "Word Voter: A New Voter Design for Triple Modular Redundant Systems," in *Proc. of the 18th IEEE VLSI Test Symposium*, 2000, pp. 465–470.

[16] N. Storey, *Safety-Critical Computer Systems*. Addison Wesley Longman, 1996.

[17] G. A. Kanawati, N. A. Kanawati, and J. A. Abraham, "FERRARI: A Flexible Software-Based Fault and Error Injection System," *IEEE Trans. on Computers*, vol. 44, no. 2, pp. 248–260, Feb 1995.

[18] R. Velazco, S. Rezgui, and R. Ecoffet, "Predicting Error Rate for Microprocessor-Based Digital Architectures through C.E.U. (Code Emulating Upsets) Injection," *IEEE Trans. on Nuclear Science*, vol. 47, no. 6, pp. 2405–2411, Dec 2000.

[19] E. Jenn, J. Arlat, M. Rimén, J. Ohlsson, and J. Karlsson, "Fault Injection into VHDL Models: The MEFISTO Tool," in *Proc. of the 24tth Annual International Symposium on Fault-Tolerant Computing*, 1994, pp. 66–75.