RESEARCH ARTICLE OPEN ACCESS

A Systematic Methodology for the Detection and Correction of Soft Errors

K. Aruna, Mr. T. Suneel Kumar,
M.Tech,VLSI Design, PBR VITS, Kavali,
M.Tech, Asst. Professor, Dept of ECE, Nellore (D.T), A.P.

Abstract
Continuous shrinking in feature size, increasing power density etc, increases the vulnerability of microprocessors against soft errors even in terrestrial applications. The register file is one of the essential architectural components where soft errors can be very mischievous because errors may rapidly spread from there throughout the whole system. Thus, register files are recognized as one of the major concerns when it comes to reliability. The project introduces Self-Immunity, a technique that improves the integrity of the register file with respect to soft errors. Based on the observation that a certain number of register bits are not always used to represent a value stored in a register. The project deals with the difficulty to exploit this obvious observation to enhance the register file integrity against soft errors. It shows that our technique can reduce the vulnerability of the register file considerably while exhibiting smaller overhead in terms of area and power consumption compared to state-of-the-art in register file protection. For embedded systems under stringent cost constraints, where area, performance, power and reliability cannot be simply compromised, it proposes a soft error mitigation technique for register files. Xilinx-ISE tool for synthesizing and the VHDL language is used.
Key words: Error Correction Code, Self-Immunity Technique, Register file Integrity, Vulnerability.

I. Introduction

Over the last decade, and in spite of the increasingly complex architectures, and the rapid growth of new technologies, the technology scaling has raised soft errors to become one of the major sources for processor crashing in many systems in the nano scale era. Soft errors caused by charged particles are dangerous primarily in high-atmospheric, where heavy alpha particles are available. However, trends in today's nanometer technologies such as aggressive shrinking have made low-energy particles, which are more superabundant than high-energy particles, cause appropriate charge to provoke a soft error. On the other hand, relatively little work had been conducted for register files although they are very susceptible against soft errors. In fact, soft errors in register files can be the cause of a large number of system failures. Recently, Blome et al. showed that a considerable amount of faults that affect a processor usually come from the register file. Therefore, some processors protect their registers with Error Correction Code (ECC), but such solutions may be prohibitive in certain applications (like embedded) due to the significant impact in terms of area and power. Moreover, power consumption was conventionally a major concern in embedded systems due to their considerable power overhead. This project addresses this challenge by introducing a novel technique, called Self-Immunity to improve the resiliency of register files to soft errors, especially desirable for processors that demand high register file integrity under stringent constraints.

The contributions within this work are as follows:

(1) It presents a technique for improving the immunity of register files against soft errors by storing the ECC in the unused bits of a register.
(2) It solves the problem of the area and power overhead that typically comes as a negative side effect in register file protection by achieving high area and power saving with a slight degrading in the register file vulnerability reduction (7%) compared to a full protection scheme.

The rest of this project is organized as follows. Section 2 summarizes the previous work while Section 3 presents this proposed technique. Section 4 exhibits the implementation details and Section 5 evaluates the register file vulnerability reduction and gives a comparison to the state-of-the-art. Finally, Section 6 concludes the project.

II. Related work and background

The earliest schemes of register file protection such as Triple Modular Redundancy (TMR) and ECC can achieve a high level of fault tolerance but they may not be suitable solutions in embedded systems due to their power and area overheads. Recently, Fazeli et al. showed that protecting the whole register file with SEC-DED comes with about 20% power overhead.

The proposed approach utilizes the Cross-parity check as a method for correcting multiple errors in the register files. Spica et al. showed that there is a very little gain (just 2%) in fault tolerance for caches if they increase the protection to Double Error Correction while the overhead for that gain is considerable.

Thus, any soft error occurring during “write-write” or “read- write” intervals will have no effect on the system, because it will be corrected automatically by the next write operation. The “write-read” and “read-read” intervals are called as vulnerable intervals as shown in Fig. 1. The RVF of a register is defined as the sum of the lengths of all its vulnerable intervals divided by the sum of the lengths of all its lifetimes.

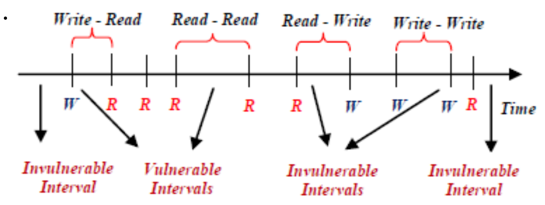


Fig. 1. Different Register Access Intervals.

The achieved RVF reduction is 23%, 41%, 67% and 93% for protecting 2, 4, 8 and 16 out of 32, 64 registers respectively. The maximizing register file integrity against soft errors by reducing the register file vulnerability. In either full or partial protection schemes, this reduction increases the area and power overheads.

III. Proposed Self-Immunity Technique

It proposes to exploit the register values that do not require all of the bits of a register to represent a certain value. Then, the upper unused bits of a register can be exploited to increase the register’s immunity by storing the corresponding SEC Hamming Code without the need for extra bits. The Hamming Code is defined by *k*, the number of bits in the original word and *p*, the required number of parity bits (approximately $\log_2 k$). Thus, the code word will be $(k + \log_2 k + 1)$s proposed technique, the optimal value of *k* is the value which guarantees that *w*, the bit-width of the register file, can cover both *k*, the required number of bits to represent the value, and the corresponding ECC bits of that value. In other words, the value and its ECC should be stored together within the bit-width of a register. Consequently, the following condition should be valid $(k + \log_2 k + 1 \le w)$e optimal value of *k* is 57 in 64-bit architectures.

When studying 64-bit architectures, where each register can represent a 64-bit value, it may exploits the register values, which require less than or equal to 57 bits by storing the corresponding ECC bits in the upper unused seven bits of that register to improve the register file integrity against soft errors. This technique is called as Self-Immunity Technique.

These values are called as “57-bit” values. It calls register values which need more than 57 bits to be represented “over-57-bit” register values. The percentage of register values usage for different applications of the MiBench Benchmark compiled for MIPS architecture is shown in Fig. 2.

A. Problem Description

1) **Goal**: The goal of our technique is to reduce the register file vulnerability with minimum impact on both area and power overhead.
2) **Effectiveness of our technique**: in a full protection scheme, an ECC generation is performed with each *write* operation and similarly ECC checking is performed with each *read* operation. This technique decides to protect the value depending if it is valid for *Self-Immunity*, then it activates the ECC generator to compute the ECC bits. Otherwise, the ECC generation is skipped. Similarly, on every register *read* operation, instead of always checking

ECC, our technique checks whether the ECC is being embedded in the register value, and only if it is, ECC checking & correction is performed. On average 12% of the data will be stored in the register file without protection. As a result, this technique reduces and it may lead to reduce the consumed power.

B. Architecture for Our Proposed Technique

The key challenge in distinguishing whether the ECC bits are embedded in the register value or not, is that the processor does not have sufficient information to make this decision when reading a value from a register. Consequently, it needs to distinguish “57-bit” register values from “over-57-bit” register values. To do that, a *self-π* bit is associated with each register and it initially clears all *self-π* bits to indicate the absence of any *Self-Immunity*. We explain the proposed architecture with the required algorithms in three different steps.

Writing, reading and correcting data into a register:

1. **Writing from a register:** Fig. 4 illustrates that whenever an instruction writes a value into a register it checks the upper seven bits of that value is '0' or not. If they are (57-bit register value case), the corresponding self-*π* bit is set to '1' indicating the existence of Self-Immunity. The ECC value is generated and stored in the upper unused bits of the register. Hence, the data value and its ECC are stored together in that register. In the second case (over-57-bit register value), the corresponding self-*π* bit is set to '0' and the value is written into the register without encoding.

2. **Reading from a register**: In read Operations , the self-*π* bit is used to distinguish between a Self-Immunity case and a non self-Immunity

case. In the first case, the value and the corresponding ECC are stored together in that register and the read value should be decoded. In another case, the stored value is not encoded and as a result there is no need to be decoded as is shown in Fig5.

3. **Corrected data:** In correction, it will check Encoder hamming code and decoder hamming code, two hamming codes are equal there is no correction otherwise it will check the Self-Immunity case and correct the data

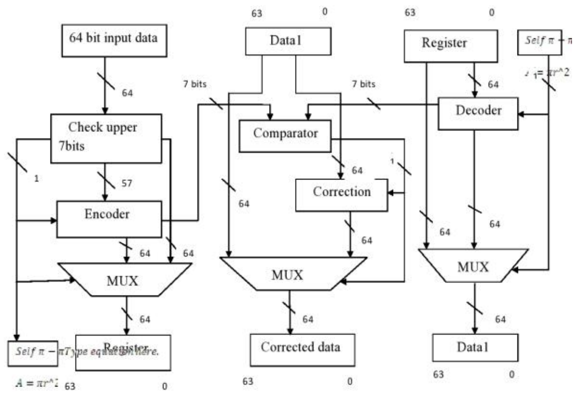


Fig. 2. Microarchitectural support for reading a register value.

C. Potential Power Saving

This proposed architecture promises to consume less power. In this architecture, "over-57-bit" register values are neither encoded nor decoded and consequently the encoding and decoding operations are not performed with each read and write operation as it happens in a full protection scheme. This may reduce the power consumption of this proposed architecture because the encoding and decoding operations are performed only in the case of "57-bit" register values. Fig4 shows that on average 12% and 13% of the total number of read and write operations are occurred in the case of "over-57-bit" register values. As a result, proposed architecture may consume less power because the encoder and decoder are lesser times accessed.

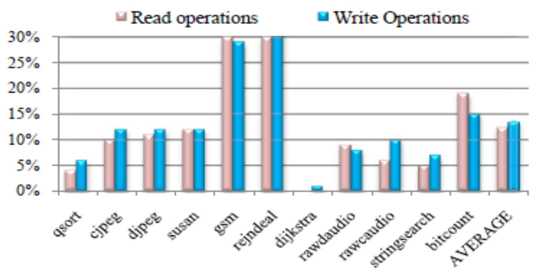


Fig4. The percentage of read and write operations in the case of "over-26-bit" register values

IV. Implementation details

Since the probability of multiple bit-errors is largely lower than the single bit-error, a single bit-rror model has been considered in this project. In this fault injection environment, faults are injected on the fly while the processor executes an application. In each fault injection simulation, one of the 64 registers is selected randomly and a bit in that register is chosen randomly and then flipped. Notice that a *write* operation clears out the previous injected error into that register. Likewise, by using a uniform distribution, a random cycle is chosen as the time that soft error occurs. This makes sure that the faults will be injected only when the program is executed. Since an injected fault might produce an infinite loop, a watchdog timer was implemented for the required number of execution cycles. It stops the simulation when the cycle count exceeds two times the number of cycles in the fault-free case. Towards evaluating this proposed technique, it uses different applications from MiBench Benchmark compiled for MIPS architecture to take into account different possible scenarios for register utilization. Simulations were conducted using the MIPS model simulator. When a simulation terminates, the corresponding output information (final results, content of the register file, execution time and state of the processor) are stored and used to classify the simulation.

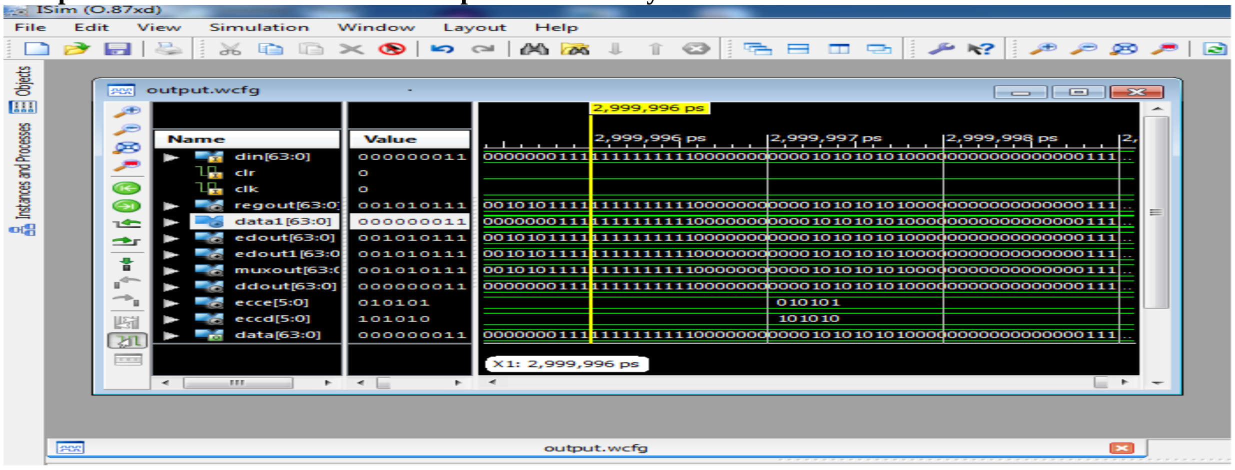
V. Experimental Results and Evaluation

As is depicted in Table 1, this proposed technique improves the register file integrity effectively by reducing largely the number of errors in each category. On average, this proposed technique reduces the number of error by 100%, 87%, 93%, 93%, and 100%

Table.1. Processor behavior for single error injection after implementing this proposed technique.

		cjpeg	djpeg	bitcount	qsort	rawdaudio	rawca-udio
Effect-Less	Base	5934	5801	5118	6415	5607	2655
	SI	9011	8961	8501	8751	7842	7768
Latent	Base	1113	1241	826	1164	896	1916
	SI	965	965	953	1249	2006	2062
Wrong Answer	Base	24	42	872	192	294	1434
	SI	2	13	0	0	0	2
Timed-Out	Base	180	736	801	22	1038	713
	SI	15	44	377	0	152	11
Stalling	Base	117	10	5	57	285	1917
	SI	1	0	0	0	0	0
Exception	Base	1646	1461	1567	1063	1464	335
	SI	0	0	0	0	0	0
Crashing	Base	926	621	811	1105	416	1030
	SI	6	17	169	0	0	157

Complete Simulation Result of Implemented System:



```
Name            Value
din[63:0]       000000011   0000000111|11111111110000000|0000101010101000|000000000000000111..
clr             0
clk             0
regout[63:0]    001010111   0010101111|11111111110000000|0000101010101000|000000000000000111..
data1[63:0]     000000011   0000000111|11111111110000000|0000101010101000|000000000000000111..
edout[63:0]     001010111   0010101111|11111111110000000|0000101010101000|000000000000000111..
edout1[63:0     001010111   0010101111|11111111110000000|0000101010101000|000000000000000111..
muxout[63:(     001010111   0010101111|11111111110000000|0000101010101000|000000000000000111..
ddout[63:0]     000000011   0000000111|11111111110000000|0000101010101000|000000000000000111..
ecce[5:0]       010101                           010101
eccd[5:0]       101010                           101010
data[63:0]      000000011   0000000111|11111111110000000|0000101010101000|000000000000000111..
```

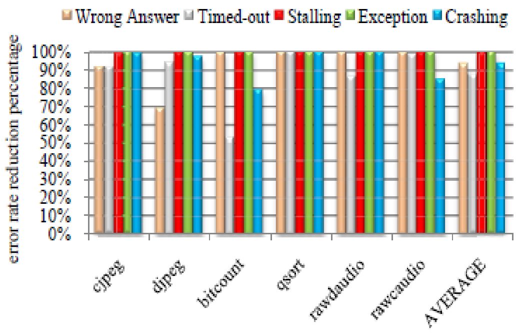


Fig.5. Percentage of error rate reduction after implementing this proposed technique.

Since latent errors have no effect on the output of an application, they are less harmful. This means that it can safely add the "Latent" category to the "Effect-Less" category since in both categories the final results are still completely correct. In this case, on average, the system fault coverage after implementing our technique reaches on average 98% and up to 100%.

To investigate the advantages of using this proposed technique in terms of area overhead against "Fully ECC" and against the partially protection, it implemented and synthesized for a Xilinx XC2V600 different versions of a 64-bit, 64-entry, dual read ports, single write port register file. Fig. 6 shows the comparison results in terms of RVF reduction and area overhead. As is noticed, this technique achieves a good area saving with slight degradation (7%) in the register file vulnerability reduction compared to "Fully ECC". Furthermore, protecting 16 out of 64 registers "16ECCs" can achieve similar RVF reduction to our result but this technique occupies 31% less area. On the other hand, protecting 4 registers "4ECCs" comes with an area overhead similar as this technique but this technique achieves 1.3X improvement in terms of RVF reduction.

Since the main target of this project are 64-bit embedded processors, a synthesizable VHDL model of the DLX processor is used to investigate the performance and power penalties for each technique.

Also the Xpower tool from Xilinx is used to estimate the total power consumption. Since the used encoder and decoder are less complex as explained earlier, the critical path in this proposed architecture is shorter. Consequently, this technique improves the performance compared to other competitors. As shown Fig.7, this technique comes with a minimum impact on both performance and power. It achieves 54% delay reduction and consumes with 94% less power compared to "Fully ECC". Furthermore, protecting 16 out of 64 registers "16 ECCs" achieves similar RVF reduction as mentioned before, but this technique achieves a 47% performance improvement and consumes 87% less power. On the other hand, this technique consumes 75% less power and achieves 29% improvement in terms of delay overhead compared to "4ECCs". It can be concluded that this technique achieves the best overall result compared to state-of-the-art in register file vulnerability reduction.

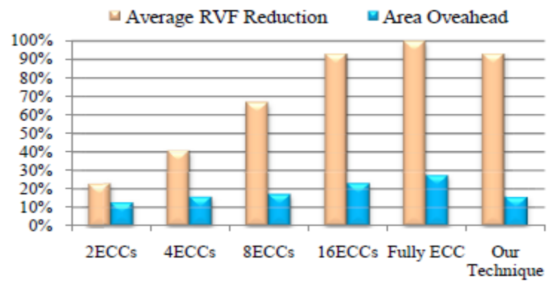


Fig.7.The performance and power overhead comparison

VI. CONCLUSION

For embedded systems under stringent cost constraints, where area, performance, power and reliability cannot be simply compromised, it proposes a soft error mitigation technique for register files. This experiment on different embedded system applications demonstrate that this proposed *Self-Immunity* technique reduces the register file vulnerability effectively and achieves high system fault coverage. Moreover, this technique is generic as it can be implemented into diverse architectures with minimum impact on the cost.

REFERENCES

[1] Greg Bronevetsky and Bronis R. de Supinski,"Soft ErrorVulnerability of Iterative Linear Algebra Methods," in the 22ndannual international conference on Supercomputing, pp. 155-164,2008.

[2] J.L. Autran, P. Roche, S. Sauze, G. Gasiot, D. Munteanu, P. Loaiza,M. Zampaolo and J. Borel, "Real-time neutron and alpha soft-errorrate testing of CMOS 130nm SRAM: Altitude versus undergroundmeasurements," in ICICDT'08, pp. 233–236, 2008.

[3] S.S. Mukherjee, C. Weaver, J. Emer, S.K. Reinhardt and T. Austin,"A Systematic Methodology to Compute the ArchitecturalVulnerability Factors for a High-Performance Microprocessor," inInternational Symposium on Microarchitecture (MICRO-36), pp.29-40, 2003.

[4] T.J. Dell, "A whiteproject on the benefits of Chippkill-Correct ECC forPC server main memory," in IBM Microelectonics division Nov1997.

[5] S. Kim and A.K. Somani, "An adaptive write error detectiontechnique in on-chip caches of multi-level cache systems," in Journalof microprocessors and microsystems, pp. 561-570, March 1999.

[6] G. Memik, M.T. Kandemir and O. Ozturk, "Increasing register file immunity to transient errors," in Design, Automation and Test inEurope, pp. 586-591, 2005.

[7] Jongeun Lee and AviralShrivastava, "A Compiler Optimization toReduce Soft Errors in Register Files," in LCTES 2009.

[8] Jason A. Blome, Shantanu Gupta, ShuguangFeng, and Scott Mahlke,"Cost-efficient soft error protection for embedded microprocessors,"in CASES '06, pp. 421–431, 2006.

[9] P.Montesinos, W.Liu, and J.Torrellas, "Using register lifetimepredictions to protect register files against soft errors," in DependableSystems and Networks, pp.286–296, 2007.

[10] M. Rebaudengo, M. S. Reorda, and M.Violante, "An AccurateAnalysis of the Effects of Soft Errors in the Instruction and DataCaches of a Pipelined Microprocessor," in DATE'03, pp. 602-607,2003.

[11] I. Koren and C. M. Krishna, Fault-Tolerant Systems. San Mateo, CA:Morgan Kaufmann, 2007.

[12] MiBench (http://www.eecs.umich.edu/ mibench/).

[13] T. Slegel et al, "IBM's S/390 G5 microprocessor design," in IEEEMicro, 19, pp. 12-23, 1999.

[14] M. Fazeli,A. Namazi, and S.G. Miremadi "An energy efficientcircuit level technique to protect register file from MBUs and SETs inembedded processors," in Dependable Systems & Networks 2009, pp.195–204,DNS'09.

[15] K. Walther, C. Galke and H.T. VIERHAUS, "On-Line Techniquesfor Error Detection and Correction in Processor Registers with Cross-Parity Check," in Journal of Electronic Testing: Theory andApplications 19, pp.501-510, 2003.

[16] M. Spica and T.M. Mak, "Do we need anything more than single biterror correction (ECC)?," in Memory Technology, Design andTesting, Records of the International Workshop on 9-10, pp. 111–116, 2004.

[17] M. Kandala, W. Zhang, and L. Yang, "An area-efficient approach to improving register file reliability against transient errors," in Advanced Information Networking and Applications Workshops, AINAW '07, pp. 798–803, 2007.

[18] http://archc.sourceforge.net/.

[19] Jun Yan and Wei Zhang, "Compiler-guided register reliability improvement against soft errors," in EMSOFT '05, pp. 203–209,2005.

[20] E. Touloupis, J.A. Flint, V.A. Chouliaras and D.D. Ward, "Efficient protection of the pipeline core for safety-critical processor-based systems," in IEEE workshop on Signal Processing Systems Design and Implementation, pp. 188-192, 2005.

[21] Jongeun Lee and A. Shrivastava, "A Compiler-Micro architecture Hybrid Approach to Soft Error Reduction for Register Files," in Computer-Aided Design of Integrated Circuits and Systems, pp.1018-1027, 2010.

[22] Jongeun Lee and A. Shrivastava, "Compiler-managed register fileprotection for energy-efficient soft error reduction," in ASP-DAC, pp.618–623, 2009.

[23] RiazNaseer, RashedZafarBhatti, and Jeff Draper, "Analysis of SoftError Mitigation Techniques for Register Files in IBM Cu-08 90nmTechnology," in MWSCAS'06, pp. 515-519, 2006.