DR.R HARIBABU,
DR.CH.SRAVAN KUMAR
DR.P SESHU KUMAR

Journal of Nonlinear Analysis and Optimization
Vol. 14(2) (2023), February 2023
https://ph03.tci-thaijjo.org/
ISSN : 1906-9685

J.Nonlinear Anal. Optim

Error protection scheme for registers

DR.R HARIBABU, Professor, Department of CSE CHADALAWADA VENKATA SUBBAIAH COLLEGE OF ENGINEERING, TIRUPATI, ragalahari789@gmail.com.
DR.CH.SRAVAN KUMAR, Professor, Department of CSE SHREE INSTITUTE OF TECHNICAL EDUCATION, RENIGUNTA, sravan.1948@gmail.com.
DR.P SESHU KUMAR, Professor, Department of CSE VAISHNAVI INSTITUTE OF TECHNOLOGY, TIRUPATI, seshu1243@gmail.com.

Abstract:-The Vulnerability of Microprocessors against soft errors increases due to reduce in feature size, increase in power density, etc. The register file is one of the essential architectural components where soft errors can be very mischievous because these errors can rapidly spread from register files throughout the whole system. Thus, register files are recognized as one of the major concerns when it comes to reliability. This project introduces Self-Immunity Technique; it improves the immunity of the register file against soft errors. Based on the observation, a certain number of register bits are not always used to represent a value stored in a register. This project deals with the difficulty to exploit this obvious observation to enhance the register file integrity against soft errors. It shows that this technique can reduce the register file vulnerability considerably while exhibiting smaller overhead in consumption of area and power compared to state-of-the-art in protection of register file. 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. To accomplish this project Modelsim for logical verification and Xilinx-ISE tool for synthesizing and the VHDL language is used.

Key words: Error Correction Code, Self-Immunity Technique, Register file Integrity, Vulnerability.

1. INTRODUCTION

Over the last decade, and in spite of the increasingly complex architectures, and the fast growth of new technologies, the technology scaling has raised soft errors to become one of the major sources for processor crashing in many systems. 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. The soft errors will become a cause of an inadmissible error rate problem in earthbound applications. Researchers have mainly and traditionally focused on reducing soft errors in memory and cache structures, due to their large sizes. On the other hand, relatively little work had been conducted for register files although they are very susceptible against soft errors. The corrupted data in any register, if not taken care of it, may propagate rapidly throughout the other parts of system's processor, leading to effective system reliability problems. In fact, soft errors in register files can be the cause of a large number of system failures. The considerable amount of faults that affect a system's 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. Power consumption was conventionally a major problem in embedded systems due to their considerable effects on the system. To bridge the gap, there is an aggravated need of techniques to increase integrity of register file against soft errors with a small effect on both area and power overhead. This project addresses this challenge by introducing Self-Immunity Technique to improve the

register file integrity to soft errors, especially desirable for processors that demand high register file immunity under stringent constraints. Self-Immunity technique used for improving the integrity of register files against soft errors by storing the ECC in the unused bits of a register. 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.

2. RELATED WORK AND BACKGROUND

The existing schemes of register file protection such as Triple Modular Redundancy (TMR) and ECC can achieve a high level of fault tolerance but they cannot be suitable solutions in embedded systems due to their power and area overheads. The 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. Building on the concept of Architectural Vulnerability Factor (AVF), it proposed the Register Vulnerability Factor (RVF) to describe the soft errors in registers can be spread to other system parts. In general, a value is written into a register, then it is read frequently, and later a new value is written again. 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.

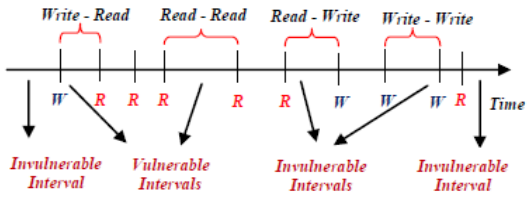


Fig. 1. Different Register Access Intervals.

The pure software approach at compile level, re-schedules the instructions in order to decrease the RVF of a register file but the proposed technique is not always very effective because it may increase the execution cycles and even the RVF in some benchmarks. To reduce the area and power, to protect a subset of the registers instead of full protection schemes and modify the register allocation algorithm to assign the most sensitive registers against soft errors to the protected registers. The achieved RVF reduction is 23%, 41%, 67% and 93% for protecting 2, 4, 8 and 16 out of 32 registers respectively. A compile technique to reduce RVF by protecting a small part of memory and write the vulnerable register values in this memory by inserting load/store instructions but it increases both runtime and code size.

Another important approach is In-Register Replication "IRR", which exploits the fact that a large fraction of register values are less than or equal to 16 bits wide for 32-bit architectures. Such values can be replicated in the same register for increasing the integrity against soft errors. 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.

3. 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 file integrity by storing the corresponding SEC Hamming Code without the need for extra bits. This Code is defined by k, the number of bits in the original word and p, the required number of parity bits (approximately). Thus, the code word will be $(k + \log_2 k + 1)$roposed technique, the optimal value of k is the value which guarantees that w (bit-width of register file), can cover both k, the required number of bits to represent the value, the corresponding ECC bits of that value. Consequently, following condition should be valid . The optimal value of k is 26 in 32-bit architectures and 57 in 64-bit architectures.

When $\log_2 k$ng 32-bit architectures, where each register can represent a 32-bit value, it may exploits the register values, which require less than $(k + \log_2 k + 1 \le w)$ bits by storing the corresponding ECC bits in the upper unused six 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 “26-bit” values. It calls register values which need more than 26 bits to be represented “over-26-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.

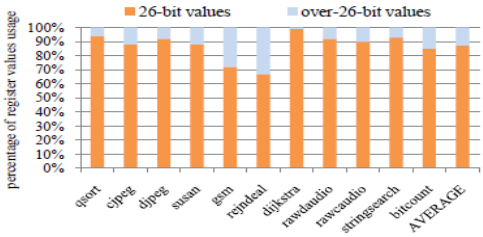


Fig. 2. “26-bit” register values and “over-26-bit” register values in different benchmarks.

Most of the register values are “26-bit” values. The upper six bits of 88% of the stored data in the register file are actually unused. It can stores the corresponding ECC in these available bits and increase the integrity of register files.

In addition to the previous key observation, the contribution of “26-bit” register values in the total vulnerable intervals is much more than the contribution of “over-26-bit” register values, The fraction of vulnerable intervals of each benchmark is reported is shown in Fig. 3. As is demonstrated, the fraction of vulnerable intervals of “26-bit” values is 93% on average.

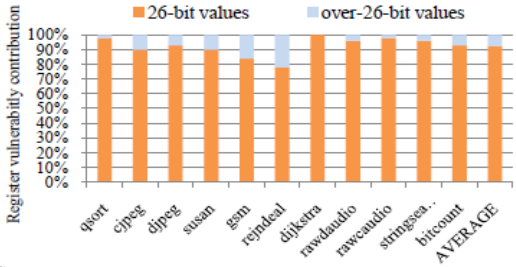


Fig. 3. The fraction of vulnerable intervals of “26-bit” register values and “over-26-bit” register values in different benchmarks.

A. Problem Description

1) **Goal**: The goal of this technique is to reduce the vulnerability of register file with minimum impact on both area and power overhead. Let N be the total number of registers and V the vulnerability of a register, then the vulnerability of the register file is . Let M is the number of protected register values and A the number of accesses, then the total power overhead can be estimated as . Thus, this goal can be formulated as:

M$(\sum_{i=1}^{N} Vi)$:e , Minimize . .

2) **Effectiveness of our technique**: In a full protection scheme, an ECC generation is performed with each write operation and ECC checking is perfo$(\sum_{i=1}^{M} Ai)$ 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 g$(V = \sum_{i=1}^{N} Vi)$ is skipped. Si$(P = \sum_{i=1}^{M} Ai)$ every register read operation, instead of always checking ECC, this technique checks whether the ECC is being embedded in the register value, and only if it is, ECC checking is performed. An average 12% of the data will be stored in the register file without protection as shown in fig 2. From Fig3, when studying 32-bit architectures, around 93% of vulnerable intervals will potentially be invulnerable. Thus, this technique promises to reduce the register file vulnerability.

B. Architecture for Our Proposed Technique

It needs to distinguish “26-bit” register values from “over-26-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. For the sake of simplicity, it explains the proposed architecture with the required algorithms in two steps.

Writing into a register- Fig. 4 illustrates that whenever an instruction writes a value into a register it checks the upper six bits of that value is '0' or not.

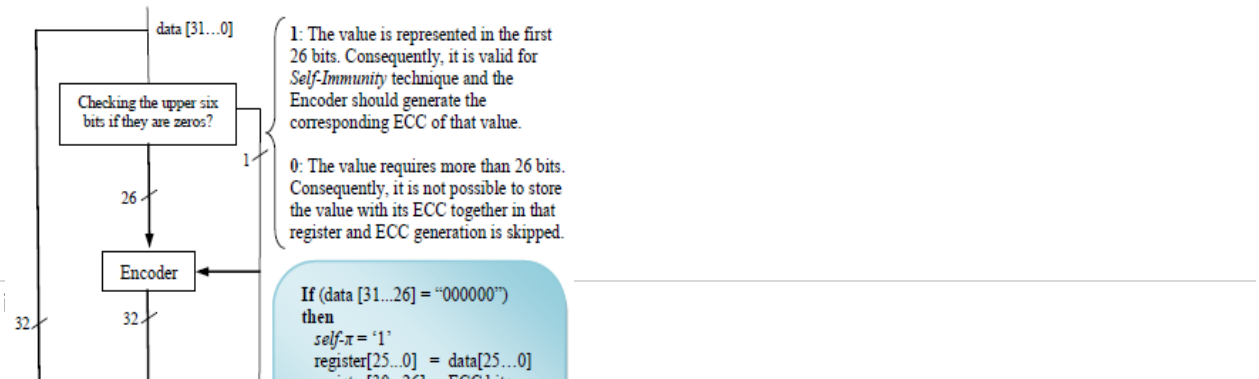


Fig4. Microarchitectural support for writing a register value.

If they are (26-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-26-bit register value), the corresponding self-π bit is set to '0' and the value is written into the register without encoding.

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.

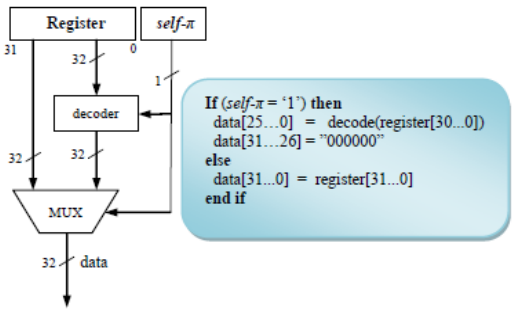


Fig5. Microarchitectural support for reading a register value.

C. Potential Power Saving

This proposed architecture promises to consume less power. In this architecture, "over-26-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 "26-bit" register values. Fig6 shows that on average 12% and 13% of the total number of read and write

operations are occurred in the case of "over-26-bit" register values. As a result, proposed architecture may consume less power because the encoder and decoder are lesser times accessed.

Fig6. The percentage of read and write operations in the case of "over-26-bit" register values.

This proposed architecture used a less complex encoder and decoder. This may also lead to a further saving in the terms of the power consumption. Finally, this proposed architecture reduced the total number of bits of a protected register from 38 bits to 33 bits and as a result the consumed switching power is lower. The power saving is mainly due to the fewer ECC operations, the usage of a less complex ECC generator and checker, and the absence of additional storage for ECC.

4. IMPLEMENTATION DETAILS

The probability of multiple bit-errors is largely lower than the single bit-error, a single bit-error 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 32 bit registers is selected randomly and a bit in that register is chosen randomly and then flipped. The write operation clears out the previous injected error into that register. 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. 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 are stored and used to classify the simulation. For the classification, it exploits the following categories proposed in An Accurate Analysis of the Effects of Soft Errors in the Instruction and Data Caches of a Pipelined Microprocessor, Efficient protection of the pipeline core for safety-critical processor-based systems:

- **Wrong Answer:** The application terminates normally but the results are not correct.
- **Latent:** The application terminates normally, but the results are correct but at the end of simulation the content of the register file is different from that of fault-free case.
- **Effect-Less**: The application terminates normally, the results are correct, and the content of the register file is similar to that of fault-free case.
- **Exception:** The processor detected the injected fault and generated an exception (e.g., invalid address exception).
- **Timed-Out:** The application failed to terminate and produce results with a predefined time limit.
- **Stalling:** The processor computed the expected results in a time greater than the time of fault-free case.
- **Crashing:** The processor fails to terminate normally. Each benchmark was simulated 10,000 times. As a result, 10,000 soft errors were injected randomly in the register file. This number compiles with those used by other research to keep the total time of simulations reasonable. For a fair comparison, it considers three models of the processor:

Base: a normal processor (without implementing any protection technique).
IRR: a fault tolerant model, where an In-Register Replication technique is implemented. This technique has been chosen here because it tries to achieve a similar goal as this proposed technique.
SI: a fault tolerant version, where our proposed technique, Self-Immunity, is implemented.

5. EXPERIMENTAL RESULTS AND EVALUATION

As is shown in Table1, this proposed technique improves the register file immunity effectively by reducing number of errors. The number of errors reaches zero in some benchmarks. On average, this proposed technique reduces the number of error by 100%, 87%, 93%, 93%, and 100% for the following categories: Exception, Timed-Out, Crashing, Wrong Answer, and Stalling respectively as is shown in Fig. 7.

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

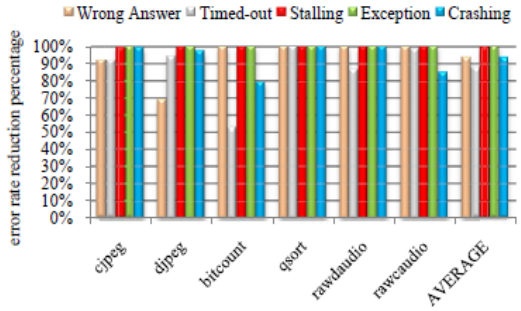


Fig.7. Percentage of error rate reduction after implementing this proposed technique.

In this case, on average, the system fault coverage after implementing this technique reaches on average 98% and up to 100% as is shown in Fig8.

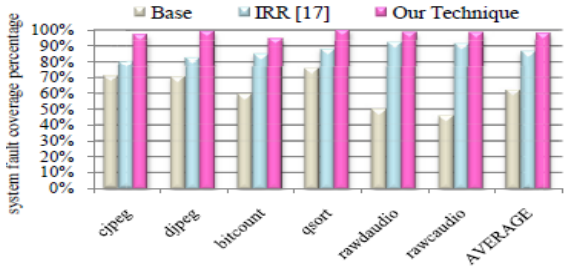


Fig.8. System fault coverage comparison for different benchmarks.

Fig. 9 shows that the potential RVF reduction is always very high. It reaches 93% on average and up to 100%. It achieves the best result compared to the IRR technique. Partial ECC protection technique instead of protecting the whole register file "Fully ECC" to achieve area and power saving while increasing the register file vulnerability reduction.

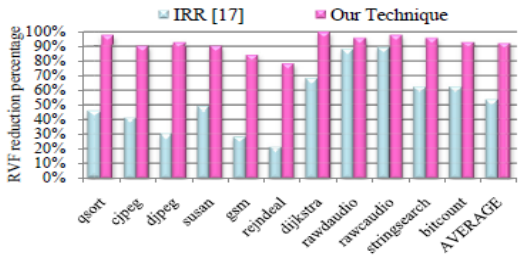
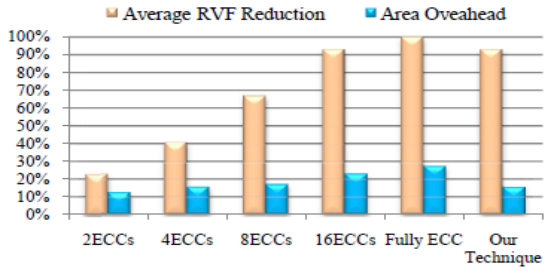


Fig.9. Comparison of Register Vulnerability Factor Reduction.

An 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 32-bit, 32-entry, dual read ports, single write port register file. Fig. 10 shows the comparison results in terms of RVF reduction and area overhead. 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 32 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 32-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 in each of the different processor versions for the adcpm decoder benchmark application. Since the used encoder and decoder are less complex as explained earlier, the critical path in this proposed architecture is shorter. This technique improves the performance compared to other

competitors. As shown Fig. 11, 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 32 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"4.



F **ea overhead.**

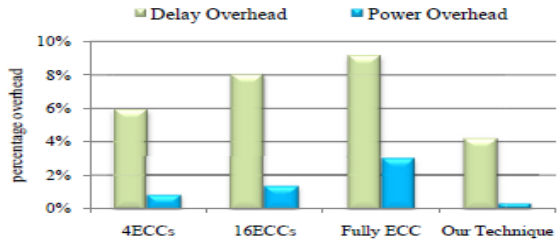


Fig. 11. The performance and power overhead comparison.

6. CONCLUSION

For embedded systems under stringent cost constraints, where area, performance, power and reliability cannot be simply compromised, it proposed 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. This technique is generic as it can be implemented into diverse architectures with minimum impact on the cost.

7. REFERENCES

[1] Greg Bronevetsky and Bronis R. de Supinski,"Soft Error Vulnerability of Iterative Linear Algebra Methods," in the 22nd annual 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 Architectural Vulnerability Factors for a High-Performance Microprocessor," in International Symposium on Microarchitecture (MICRO-36), pp.29-40, 2003.

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

[5] S. Kim and A.K. Somani, "An adaptive write error detection technique in on-chip caches of multi-level cache systems," in Journal of 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 in Europe, pp. 586-591, 2005.

[7] Jongeun Lee and AviralShrivastava, "A Compiler Optimization to Reduce 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 lifetime predictions to protect register files against soft errors,” in Dependable Systems and Networks, pp.286–296, 2007.
[10] M. Rebaudengo, M. S. Reorda, and M.Violante, “An Accurate Analysis of the Effects of Soft Errors in the Instruction and Data Caches 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 efficient circuit level technique to protect register file from MBUs and SETs in embedded processors,” in Dependable Systems & Networks 2009, pp.195–204,DNS’09.
[15] K. Walther, C. Galke and H.T. VIERHAUS, “On-Line Techniques for Error Detection and Correction in Processor Registers with Cross-Parity Check,” in Journal of Electronic Testing: Theory and Applications 19, pp.501-510, 2003.
[16] M. Spica and T.M. Mak, “Do we need anything more than single bit error correction (ECC)?,“ in Memory Technology, Design and Testing, 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-Microarchitecture 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 file protection for energy-efficient soft error reduction,” in ASP-DAC, pp.618–623, 2009.
[23] RiazNaseer, RashedZafarBhatti, and Jeff Draper, “Analysis of Soft Error Mitigation Techniques for Register Files in IBM Cu-08 90nmTechnology,” in MWSCAS’06, pp. 515-519, 2006.