Robust Register Files by Exploiting Asymmetric Soft Error Rate

Yohan Ko
Dept. of Computer Science
Yonsei Univeristy
Yohan.Ko@yonsei.ac.kr

Kyoungwoo Lee
Dept. of Computer Science
Yonsei Univeristy
klee@cs.yonsei.ac.kr

Abstract— As technology scaling, soft errors induced by external radiation or cosmic rays are becoming a serious concern in micro-architectures. In particular, soft errors in register files are critical in reliability since these errors are easily propagated to other components of processors, causing catastrophic system failures. To protect data in register files, there exist redundancy techniques such as Triple Modular Redundancy (TMR) and Error Correction Code (ECC). However, these techniques incur high overheads in terms of area cost, performance, and power consumption. In this paper, we increase reliability on data in register files by simply applying inverters since soft error rates are asymmetric, i.e., different between 0 and 1 in bit values. The main idea behind our approach is to increase the more stable bit values in register files by inverting bit values if it has more unstable bit values. Our experimental results show that our proposal can reduce soft error rates by up to 20% over a suite of benchmarks with minimal overheads due to inverters.

I. Introduction

With increased chip integration levels, reduced supply voltage, increased speed, transient faults are becoming a big threat for todays system reliability [1], [2], [13], [15]. Such transient fault, i.e., soft error, caused by cosmic ray or radiation particle is a temporary hardware defect in a signal or datum [14]. In this paper, our research is to reduce soft error rates (SER) in register files which are significantly sensitive to system reliability since they are frequently accessed by CPU.

When energetic particles strike the sensitive area of the silicon device, they generate electron-hole pairs in the wake. The source and diffusion nodes of a transistor can collect these charges, $Q_{collected}$. When $Q_{collected}$ becomes larger than critical charge, $Q_{critical}$, the state of the logic device may invert. For a long time, just high energy particles have caused soft errors but now low energy particles also cause soft errors. The effect is multiplied with the fact that there are a lot more lower-energy particles than higher-energy ones and the fact that $Q_{critical}$ is becoming low and low as technology scaling.

Soft errors have already been attributed to cause several fiscal damages [11], [20], e.g., Sun blamed soft errors for the crash of their million-dollar line SUN flagship servers in Nov. 2000 [3]. In one incident, a single soft error crashed an interleaved system farm. In another incident, a single soft error brought a billion-dollar automotive factory to halt every month [5]. At the current technology, a soft error may occur in a high-end server once every 170 hours, but it is expected to increase exponentially with advance of technology mainly because of lowering $Q_{critical}$.

In embedded systems, soft errors in register file are the important factor in terms of performance. Register file is an array of processor registers in a control processing unit (CPU). This is central to the architecture and often stores data for long periods of time. Moreover, if soft errors occur in register file, these soft errors quickly propagate to errors in other parts of processors [5], [6]. For these reasons, some commercial processor protects register file by using typical redundancy methods such as TMR and ECC. However, these techniques for register files incur high overheads in terms of area cost, access time, and power consumption [7], [8].

In order to reduce soft errors with minimal overheads, several techniques have been investigated. For example, Pablo Montesinos et al. [9] reduced soft error rates by using the observation that the reliability-sensitive time is limited during the register lifetime. For this reason, they proposed ParSHIELD that protects register files from soft errors with selective ECC codes. Jun Yan et al. [10] presented a compiler-guided method by exploiting the scheduling algorithms such as superblock and hyperblock to delay write operations as late as possible, which can alleviate the vulnerable time in register files. However, these methods still exploit expensive redundancy-based techniques for their protection. Therefore, we propose the software-based register file protection by exploiting unique feature of soft error rates, especially, rates for SRAM.

ASER implies that SER is asymmetric, i.e., the probability of bit flip form 0 to 1 is not equal to that from 1 to 0. Especially, they are different by up to several orders of

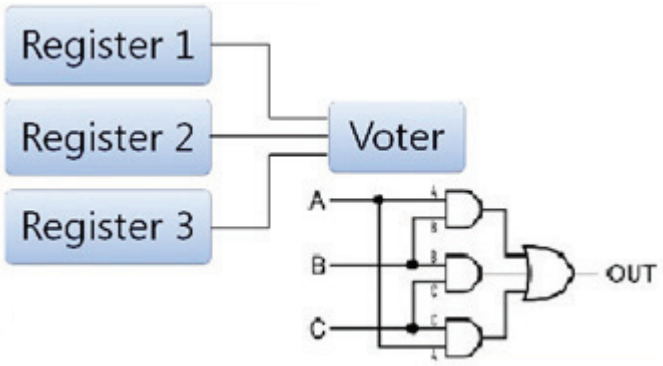

Fig. 1. TMR Architecture

magnitudes in low power SRAMs [11]. This is mainly because of the big difference of the critical charges between two types of flips. In most low power SRAM based register files, the SER of the 1 to 0 is much higher than that of 0 to 1 (about 20 times in $Q_{collected}$) [11]. In other words, bit value of 0 is much more robust against soft errors than that of 1, which implies that minimizing bit values of 1 in data makes register files more resilient against soft errors.

In this paper, we propose a simple profiling-based approach to maximize the number of more stable bit values in register files by exploiting the feature of ASER. Moreover, we consider the concept, Architectural Vulnerability Factor (AVF) [4], to obtain more accurate decline in SER. Over the benchmarks we used for our experiments, our proposal can reduce SER by up to 25% for the AVF-based profiled methods.

The paper is organized as follows. Section 2 presents related works; Section 3 describes design and implementation idea of our approach; Section 4 shows our efficacy in soft error robustness from simulation-based experiments; Section 5 concludes our paper and discusses future works.

II. Related Work

To mitigate soft errors in register files, redundancy-based techniques have been investigated. These techniques include TMR and Hamming Code based ECC. The former exploits three functionally equivalent replicas of a component with a majority voting mechanism as shown in Figure 1. This method detects and corrects most cases of soft errors in one replica out of three. However, this method incurs 204% area costs, 19% access time as compared to the default one without any replica [7], [8], [19]

Another redundancy-based technique is a Hamming code based ECC, which is a linear ECC to correct errors by exploiting additional parity bits. The number of required parity bits for 2n data bits is n+1 if it is designed for combatting single-bit errors. For instance, we need 6 parity bits for 32 bits register , which code is called (38, 32) Hamming code which is able to correct single bit errors and to detect up to double bit errors. This approach is more resource efficient than TMR since most soft errors are single bit errors. However, fully Hamming code without any optimization still incurs more than 27% area costs, 113% access time overhead, 110% power consumption overhead as compared to the normal register files without any protection [7].

For these reasons fully hardware-based protection is significantly costly. Jun Yan et al. [10] presented the register file protection by exploiting selective partial ECC. In [10], after the modification of conventional register allocation algorithm is done, ECC protects selectively based on Register Vulnerability Factor (RVF) similar with Architectural Vulnerability Factor (AVF). However, this method has still hardware overhead because of exploiting ECC.

Pablo Montesinos et al. [9] proposed selective hardware protections of the registers that contribute the most to the overall vulnerability of the register file. In [9], this method only protects the useful lifetime of these registers. Their approach, Shield, stores the ECCs of these registers and checks their integrity offline when they are read. However, these methods didnt consider the concept, ASER that means the bit value of 0 is more stable than that of 1.

III. Our Approach

In this paper, we exploit an interesting feature of soft errors, ASER. ASER means that the bit value of 0 is more stable than that of 1. So, our goal is to increase the number of the bit value 0 in register file. Suresh Srinivasan et al [16] proposed an interesting approach in FPGA. By increasing the number of zeros in the bit stream by flipping bits in internal look-up table in case that the bit value of 0 is more than that of 1, they can reduce the failure in time compared to the original design. For these reasons, our profiling method is acceptable to improve reliability in register files against soft errors.

To exploit our suggestion, we have two registers equipped with inverters in the register files as shown in Figure 2, which invert bit streams in the registers if our profiling results show more 1s than 0s. (Note that just two registers are equipped with inverters since other numbers of inverter-equipped registers do not show significant difference in experimental enviroments.). During the profiling, the data in register files have been read. Register profiling is to count the number of bit values in register files. At the end, we calculate the accumulated number of bit values of each register. And the two registers with maximum number of bit value 1 have been chosen and replaced (renamed) with two inverter-equipped registers (in case of our study, the last two registers, R30, R31). Note that all the benchmarks can find out more than two registers with more 1s in our profiled experiments. And the compiler re-compiles the programs by using the reallocated register files.

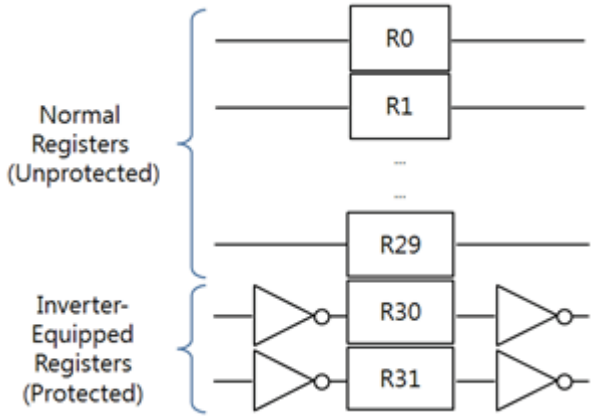

Fig. 2. Inverter-equipped Register Files for our Approach

TABLE I
Architecural Vulnerability Factor

Case	t_m	t_n	Vulnerability
0	R	R	Vulnerable
1	R	W	Non-vulnerable
2	W	R	Vulnerable
3	W	W	Non-vulnerable

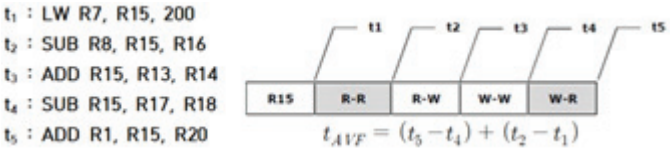

Fig. 3. Example of AVF

A. *Per-registers and Per-instruction profiling*

In this step, we decrease the number of bit values of 1 due to the reverse of the bit values of 1 in two selected registers if they have more 1s than 0s. Hence, the reliability of the program is improved in terms of soft error rates. Moreover, the overhead from the recompiling the instructions is negligible. And the hardware overhead for inverters in two registers is much lower than that of ECCs or TMR. Note that our technique implements two inverters at input and output for specific registers as shown in Figure 2. Figure 2 shows the outline of our register files where R30 and R31 are equipped with inverters and they are substituted (renamed) for two most 1s registers.

Another proposal is to profile and rename registers per each instruction. Former method is carried out statically after profiling. In contrast, this approach looks into each instruction, counts the number of bit value 1 at each register file, and dynamically renames registers with two registers with the largest and the second largest number of bit value 1. Note that our solution is applicable only if there is more 1s than 0s in registers. This per-instruction approach has more profiling and compiling overheads than those of former per-register approach while it can result in higher.

B. *AVF-Based Profiling*

However, this simple method to mitigate SER in register file is not accurate. Thus we consider about Architectural Vulnerability Factor (AVF) [4]. AVF expresses the probability that a user-visible error will occur given a bit flip in a storage cell. In register file, we can divide the accesses to register files into four different patterns (or intervals), namely, the write-read (W-R), read-read (R-R), read-write (R-W) and write-write (W-W) patterns (note that the read/write mentioned in this paper refers to the corresponding operations on register values, including but not limited to the load/store instructions, which operate on the data from the memory hierarchy). Among these four patterns, the register file is only susceptible to soft errors during the W-R and R-R intervals. In contrast, the soft errors occurred during the R-W and W-W intervals can be overlapped by the latter write operations, and hence will not impact other system components as shown in Table 1.

Figure 3 is the example of AVF of register file. In t0 time, LW(load word) instruction is executed, and then loads the value in memory(R15+200) to R7. In other words, R7 is write status, and R15 is read status. In t1 time, SUB(subtract) instruction is executed, and then the (R15-R16) is stored in R8. In other words, R8 is write status, and R15 and R16 are read status. In t2 time, ADD(add) instruction executed, and then the (R13+R14) is stored in R15. In other words, R15 is write status, and R13 and R14 are read status. In t4 time, R15 is write, and R17 and R18 are read. In t5 time, R1 is write, and R15 and R20 are read.

Then, we explain the AVF concept to exploit R15 register in t0-t4 times. In t0-t1, R15 is read-read and vulnerable. In t1-t2, R15 is read-write and non-vulnerable. In t2-t3, R15 is write-write and non-vulnerable. In t3-t4, R15 is write-read and vulnerable. Therefore, R15 is vulnerable in t0-t1 and t3-t4, and total AVF time is (t1-t0) + (t4-t3).

In our approach, time unit is defined by instruction numbers. Figure 4 is our algorithm to define AVF in register file. Line 2-4 means that default AVF in register file is non-vulnerable. Line 5-13 is the process that determines which register is vulnerable. This process means that the former operation doesnt affect vulnerability. However, if the present operation is read, the time that former operation to the present method is vulnerable time.

Table 2 shows that portion of the bit value of 1in the each register in vulnerable time. In these benchmarks,

```
1 : PROCEDURE SetAVF
2 :  FOR i = 1 to last_inst
3 :     AVF[i] = N
4 :  END FOR
5 :  FOR x = 1 to last_inst
6 :     IF OP = 'write' THEN
7 :        temp = x
8 :     ELSE
9 :        FOR y = temp to x−1 DO
10 :          AVF[y] = V
11 :       END FOR
12 :       temp = x
13 :    END IF
14 :  END FOR
15 : END PROCEDURE
```

Fig. 4. Algorithm of defining AVF

TABLE II
PORTION OF THE BIT VALUE OF 1 IN VULNERABLE TIME

	susan (%)	sha (%)
1st	85.60	81.25
2nd	74.99	80.77
3rd	72.92	64.47
4th	68.17	62.79
5th	59.13	61.50
6th	53.20	58.57

there are at least 6 registers that have more the number of the bit value of 1 than that of 0. This means our inverter-equipped profiling method is efficient to mitigate soft error rates.

IV. EXPERIMENTS

A. Experimental Setup

For experimental setup, we exploit the SimpleScalar simulator [17] with a suite of benchmarks from MiBench [18]. SimpleScalar is a set of tools that model a virtual computer system with XScale-based processor, cache and memory hierarchy. Using the SimpleScalar tools, we can build modeling applications that simulate benchmarks running on a range of modern processors and systems to quickly evaluate our solutions [12]. For soft error rate evaluations, we conservatively assume that bit value 0 is 20 times lower soft error rate than that of bit value 1 [11].

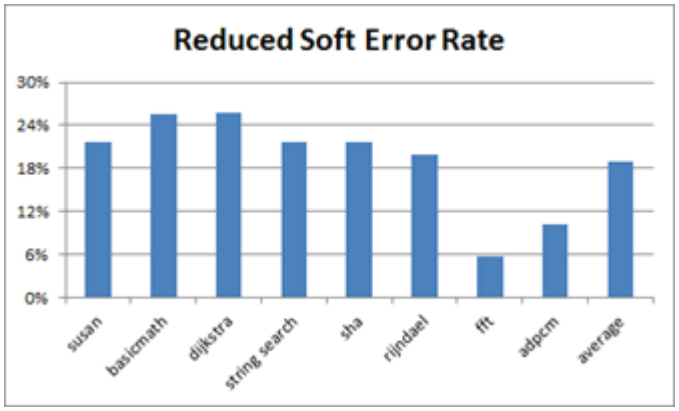

Fig. 5. AVF-base Profiling SER

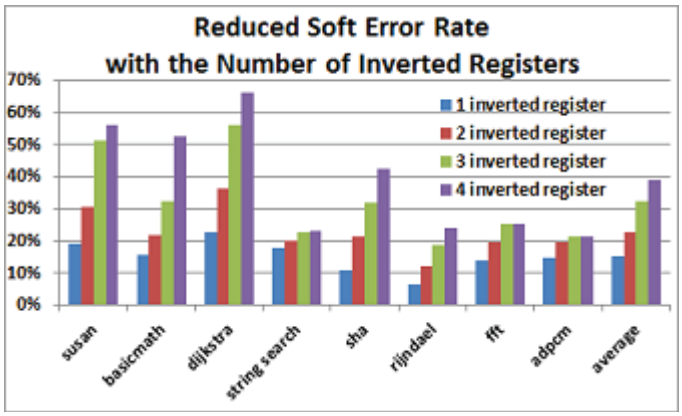

Fig. 6. AVF-base Profiling SER

B. Experimental Results

Figure 5 shows the efficacy of AVF-based profiling method by up to 25% (18% on average). AVF of the register file is 38 40% on average. And Figure 6 shows the reduced soft error rates with the number of inverted register between 1 and 4. The method by exploiting 8 inverters is most dependable, but this method has huge overhead because of adding inverters. Moreover, some benchmarks, such as *sha* and *stringserach*, show the similar dependability more than 2 inverted registers. These results show that 2 inverted register based register file is effective since that has only simple 4 inverters, which is negligible in hardware overheads. And, more than 5 inverted registers have similar reduced soft error rates in this experiment.

V. CONCLUSION

As technology scaling, soft errors are becoming a challenging concern in embedded system designs. We propose profile-based register files equipped with simple inverters by exploiting asymmetric soft error rates. Our

solutions are very effective to protect register files against soft errors by exploiting minimal hardware inverters and software-based method. Moreover, AVF-based consideration is more stable than normal approach.

Our future work includes that consideration of dynamic register renaming techniques and approaches for other micro-architectures by exploiting the characteristic of asymmetric soft error rates. In other words, our AVF-based method is not realistic, because the real processor exploits register renaming technique to mitigate W-R, R-R dependency in out-of-order processing. And, ASER is available to apply to other parts of computer science to increase reliability against soft error rates.

References

[1] Semiconductor, T., "Soft errors in electronic memory-a white paper," *URL: http://www. tezzaron. com/about/papers/Papers. htm*, 2004.

[2] Weaver, C. and Emer, J. and Mukherjee, S.S. and Reinhardt, S.K., " Techniques to reduce the soft error rate of a high-performance microprocessor ," *ACM SIGARCH Computer Architecture News*, Vol. 32, Num. 2, pp. 264, ACM, 2004.

[3] D.Lyons, " Sun screen ," *Forbes Magazine*, 2000.

[4] Mukherjee, S., " Architecture design for soft errors ," *Morgan Kaufmann*, Vol. 2008.

[5] Tremblay, M. and Tamir, Y., " Support for fault tolerance in VLSI processors ," *Circuits and Systems, 1989., IEEE International Symposium on*, pp. 388–392, IEEE, 1989.

[6] Rebaudengo, M. and Sonza Reorda, M. and Violante, M., " An accurate analysis of the effects of soft errors in the instruction and data caches of a pipelined microprocessor," *Proceedings of the conference on Design, Automation and Test in Europe-Volume 1*, pp. 10602, IEEE Computer Society, 2003.

[7] Naseer, R. and Bhatti, R.Z. and Draper, J., "Analysis of Soft Error Mitigation Techniques for Register Files in IBM Cu-08 90nm Technology," *Circuits and Systems, 2006. MWSCAS'06. 49th IEEE International Midwest Symposium on*, Vol. 1, pp. 515–519, IEEE, 2006.

[8] Fazeli, M. and Ahmadian, SN and Miremadi, SG, "A Low Energy Soft Error-Tolerant Register File Architecture for Embedded Processors," *High Assurance Systems Engineering Symposium, 2008. HASE 2008. 11th IEEE*, pp. 109–116, IEEE, 2008.

[9] Montesinos, P. and Liu, W. and Torrellas, J., "Using register lifetime predictions to protect register files against soft errors," *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on*, pp. 286–296, IEEE, 2007.

[10] Yan, J. and Zhang, W., "Compiler-guided register reliability improvement against soft errors," *Proceedings of the 5th ACM international conference on Embedded software*, pp. 203–209, ACM, 2005.

[11] Degalahal, V. and Vijaykrishnan, N. and Irwin, MJ, "Analyzing soft errors in leakage optimized SRAM design," *VLSI Design, 2003. Proceedings. 16th International Conference on*, pp. 227–233, IEEE, 2003.

[12] Ray, J. and Hoe, J.C. and Falsafi, B., "Dual use of superscalar datapath for transient-fault detection and recovery," *Microarchitecture, 2001. MICRO-34. Proceedings. 34th ACM/IEEE International Symposium on*, pp. 214–224, IEEE, 2001.

[13] Kim, S. and Somani, A.K., "Area efficient architectures for information integrity in cache memories," *ACM SIGARCH Computer Architecture News*, Vol. 27, Num. 2, pp. 246–255, ACM, 1999.

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

[15] Hazucha, P. and Svensson, C., "Impact of CMOS technology scaling on the atmospheric neutron soft error rate," *Nuclear Science, IEEE Transactions on*, Vol. 47, Num. 6, pp. 2586–2594, IEEE, 2000.

[16] Srinivasan, S. and Gayasen, A. and Vijaykrishnan, N. and Kandemir, M. and Xie, Y. and Irwin, M.J., "Improving soft-error tolerance of FPGA configuration bits," *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pp. 107–110, IEEE, 2004.

[17] Austin, T. and Larson, E. and Ernst, D., "SimpleScalar: An infrastructure for computer system modeling," *Computer*, Vol. 35, Num. 2, pp. 59–67, IEEE, 2002.

[18] Guthaus, M.R. and Ringenberg, J.S. and Ernst, D. and Austin, T.M. and Mudge, T. and Brown, R.B., "MiBench: A free, commercially representative embedded benchmark suite," *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pp. 3–14, IEEE, 2001.

[19] Balkan, D. and Sharkey, J. and Ponomarev, D. and Ghose, K., "Selective writeback: reducing register file pressure and energy consumption," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, Vol. 16, Num. 6, pp. 650–661, IEEE, 2008.

[20] Wang, N.J. and Quek, J. and Rafacz, T.M. and Patel, S.J., "Characterizing the effects of transient faults on a high-performance processor pipeline," *Dependable Systems and Networks, 2004 International Conference on*, pp. 61–70, IEEE, 2004.