ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 4, April 2013

Designing of Self Immunity Technique for 64bit Register File against Soft Errors

Supriya Sarkar[1] , Asst.prof.Tejaswini.R.Choudari[2]

Student, Dept. of ECE, RKDF Institute Of Science And Technology, Bhopal, India[1]

Assistant Professor, Dept. of ECE, RKDF Institute Of Science And Technology, Bhopal, India[2]

ABSTRACT: VLSI technology reduces the size of the digital components, reduce the power consumption and increase the speed of operation. In these criteria digital devices are more sensitively manufacturing. These type of devices mostly effected by soft errors, then it reduces the life of the devices (microprocessor, microcontrollers), it is nothing but vulnerability of microprocessors or microcontroller 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. This paper introduces *Self-Immunity*, a technique that improves the integrity of the register file with respect to soft errors. We show 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.

Keywords: Register file, Soft Error, Vulnerability Factor, Self Immunity

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 nanoscale era. Soft errors caused by charged particles are dangerous primarily in high-atmospheric, where heavy alpha particles are available [1].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. Furthermore, there is a prevailing prediction that soft errors will become a cause of an inadmissible error rate problem in the near future even in earthbound applications [2]. Researchers have mainly and traditionally focused on mitigating soft errors in memory and cache structures [4][5][13], 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 [8]. Despite the overall rather small area footprint of the register file, it is accessed more frequently than any other architectural component [6][9]. Thus, corrupted data in any register, if not taken care of, may propagate rapidly throughout the other parts of processor, leading to drastic system reliability problems [6]. In fact, soft errors in register files can be the cause of a large number of system failures [10]. Recently, Blome et al. [8] 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) [11], but such solutions may be prohibitive in certain applications (like embedded) due to the significant impact in terms of area and power [14]. Moreover, power consumption was conventionally a major concern in embedded systems due to their considerable effects on the system. To bridge the gap, there is an aggravated need of techniques to increase the register file integrity against soft errors with a small effect on both area and power overhead. This paper 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.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 4, April 2013

A. Our contributions within this paper are as follows:

(1) We present a technique for improving the immunity of register files against soft errors by storing the ECC in the unused bits of a register.
(2) We solve 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 paper is organized as follows. Section 2 summarizes the previous work while Section 3 presents our 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 paper.

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. [14] showed that protecting the whole register file with SEC-DED comes with about 20% power overhead.

The proposed approach in [15] utilizes the Cross-parity check as a method for correcting multiple errors in the register files. Spica et al. [16] 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.

Building on the concept of Architectural Vulnerability Factor (AVF), introduced by Mukherjee [3], Yan et al. [19] proposed the *Register Vulnerability Factor* (RVF) to describe the likelihood that a soft error 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 durin
"write-write" or "read- write" intervals will have no effect on the system, because it will be corrected automatically by the next write operation. On the other hand, "write-read" and "read-read" intervals are considered vulnerable intervals as is depicted 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 [19]. $( ) = \Sigma\sum$ Finally, the total vulnerability of the register file is assumed as the sum of vulnerability of all registers [21].

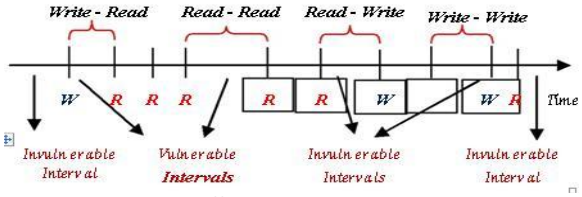

Fig. 1.Different Register Access Intervals [19].

The pure software approach at compile level introduced by Yan et al. [19] 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 [19].

In a bid to reduce the area and power penalties, Yan et al. [19] proposed 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 64 registers respectively.

Montesinos et al. [9] make a decision of which register values should be protected at runtime by hardware logic but the runtime prediction is very costly in terms of energy [22].

Lee et al. [7] presented 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" [17], which exploits the fact that a large fraction

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 4, April 2013

of register values are less than or equal to 16 bits wide for 64-bit architectures. Such values can be replicated in the same register for increasing the immunity against soft errors.

The fundamental conflict is that, while maximizing register file immunity against soft errors by reducing the vulnerability of the register file, this reduction (either with full or partial protection schemes) increases the area and power overheads.

III. PROPOSED SELF-IMMUNITY TECHNIQUE

We propose 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 [11] 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

wordnumberwillofparitybe bits (approximatelyr[23].log In) .Thus,ourproposedthecode guaranteestechnique, thatthe optimal(w,+logthebitvalue.width+1)ofofkthe is theregistervalue file, which 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 (+log +1 ≤). Thus, the optimal value of *k* is 52 in 64-bit architectures and 57 in 64-bit architectures.

For instance, when studying 64-bit architectures, where each register can represent a 64-bit value, we may exploit the register values, which require less than or equal to 52 bits by storing the corresponding ECC bits in the upper unused six bits of that register to enhance the register file immunity against soft errors[1]. We call this technique *Self-Immunity* and we call such values"52-bit"values. On theother hand, we call register values which need more than 52 bits to be represented "over-52-bit" register values. Fig. 2 shows the percentage of register values usage for different applications of the MiBench Benchmark [12] compiled for MIPS architecture.

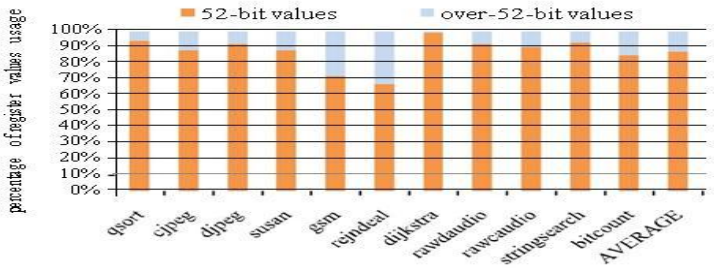

Fig. 2. "52-bit" register values and "over-52-bit" register values in different benchmarks.

As it can be noticed, in all benchmarks most of the register values are "52-bit" values. In other words, the upper six bits of 88% of the stored data in the register file are actually unused. Consequently, we can store the corresponding ECC in these available bits and increase the register's immunity.

In addition to the previous key observation, the contribution of "52-bit" register values in the total vulnerable intervals is much more than the contribution of "over-52-bit" register values. In Fig. 3, the fraction of vulnerable intervals of each benchmark is reported. As is demonstrated, the fraction of vulnerable intervals of "52-bit" values is 93% on average.

[1] Note that the proposed technique will not work with the negative values.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 4, April 2013

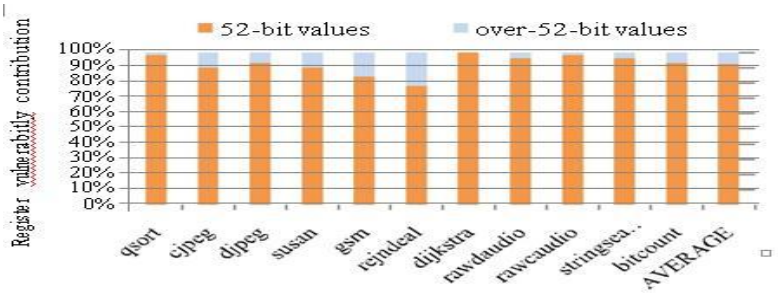

Fig. 3.The fraction of vulnerable intervals of "52-bit" register values and "over-52-bit" register values in different benchmarks

A. Problem Description

1)Goal*:*
The goal of our technique is to reduce theregister file vulnerability 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 (∑). Since the power overhead[2] mainly stems from accessing the encoder and decoder, it can approximately be modeled through the number of accesses [22]. Let *M* is the number of protected register values and *A* the number of accesses, then the total power overhead can

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. Our 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 is performed. As is demonstrated in Fig. 2, on
average 12% of the data will be stored in the register file without protection. As a result, our technique reduces *M* and it may lead to reduce the consumed power.

As is shown in Fig. 3, when studying 64-bit architectures, 93% (on average) of the total *vulnerableintervals* are *vulnerable intervals* of valid register values forour technique. In other words, around 93% of *vulnerableintervals* will potentially be invulnerable. Thus, ourtechnique promises to reduce the vulnerability of the register file considerably.

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, we need to distinguish "52-bit" register values from "over-52-bit" register values. To do that, a *self-π* bit is associated with each register and we initially clearall *self-π* bits to indicate the absence of any *Self - Immunity*. For the sake of simplicity, we explain the proposed architecture with the required algorithms in two different steps.

[2] We consider only the dynamic power and more details will be shownlater.

[3] We formulate our problem similarly to [22].

Writing into a register- Fig. 4 illustrates that wheneveran instruction writes a value into a register it checks the upper six bits of that value if they are '0' or not. If they are (52 -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 - 52-bit register value), the corresponding *self-π* bit is set to '0' and the value is written into the register without encoding.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 4, April 2013

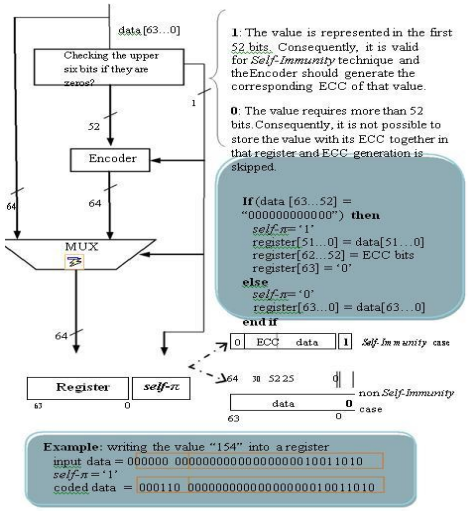

Fig. 4.Microarchitectural support for writing a register value.

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 consequently the read value should be decoded. In the second case, the stored value is not encoded and as a result there is no need to be decoded as is demonstrated in Fig. 5.

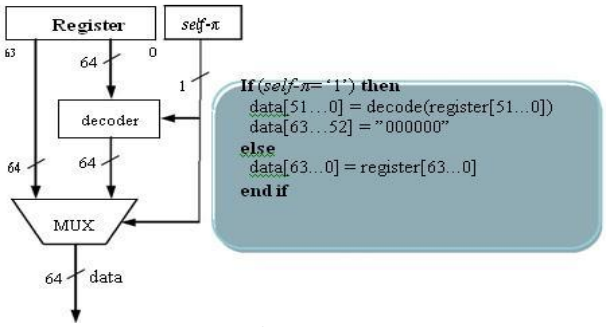

Fig. 5.Microarchitectural support for reading a register value.

C. Potential Power Saving

In this section we explain why our proposed architecture promises to consume less power. In our proposed architecture, "over-52-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 our proposed architecture because the encoding and decoding operations are performed only in the case of "52-bit" register values.

Fig. 6 demonstrates that on average 12% and 13% of the total number of read and write operations, respectively, are occurred in the case of "over-52 -bit" register values. As a result, our proposed architecture may consume less power because the encoder and decoder are lesser times accessed.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 4, April 2013

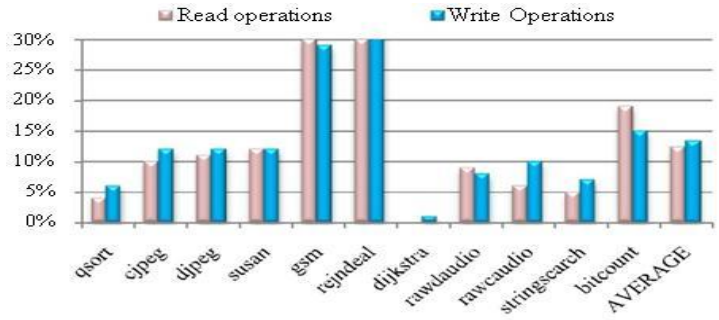

Fig. 6. The percentage of read and write operations in the case of "over-52-bit" register values.

Since the input of the deployed encoder in our architecture is 52 bits instead of 64 bits, it generates 5 parity bits instead of 6 parity bits. Likewise, the used decoder in our architecture takes 63 bits (52 bits for data + 5 bits for ECC) as an input instead of 38 bits. In other words, our proposed architecture uses a less complex encoder and decoder. This may also lead to a further saving in the terms of the power consumption. Finally, our proposed architecture reduces 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. In short, 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.

IV. IMPLEMENTATION DETAILS

Since the probability of multiple bit-errors is largely lower than the single bit -error [20], a single bit-error model has been considered in this paper. In our 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 [20]. Since an injected fault might produce an infinite loop, a watchdog timer was implemented for the required number of execution cycles. We stop the simulation when the cycle count exceeds two times the number of cycles in the fault-free case.

Towards evaluating our proposed technique, we use different applications from MiBench Benchmark compiled for MIPS architecture [12] to take into account different possible scenarios for register utilization. Simulations were conducted using the MIPS model simulator [18]. 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. For the classification, we exploit the following categories proposed in [10][20]:

- *Wrong Answer*: The application terminates normallybut the results produced are not correct.
- *Latent*: The application terminates normally, the resultsare 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, theresults are correct, and the content of the register file is similar to that of fault-free case.
- *Exception*: The processor detected the injected fault andgenerated an exception (e.g., invalid address exception).
- *Timed-Out*: The application failed to terminate andproduce results with a predefined time limit.
- *Stalling*: The processor computed the expected resultsin 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 complies with those used by other research to keep the total time of simulations reasonable. For a fair comparison, we consider three models of the processor:

Base: a normal processor (without implementing anyprotection technique).

IRR: a fault tolerant model, where an In-RegisterReplication technique [17] is implemented. This technique has been chosen here because it tries to achieve a similar goal as our proposed technique.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 4, April 2013

SI: a fault tolerant version, where our proposed technique, Self-Immunity,is implemented

V. EXPERIMENTAL RESULTS AND EVALUATION

As expected, our proposed technique maintains very high levels of fault tolerance compared to the "Base" case. As is depicted in Table 1, our proposed technique improves the register file integrity effectively by reducing largely the number of errors in each category. Furthermore, the number of errors reaches zero in some benchmarks. On average, our proposed technique reduces the number of error by 100%, 87%, 93%, 93%, and 100% for the following categories: simulation and synthesis reports r shown below fig.

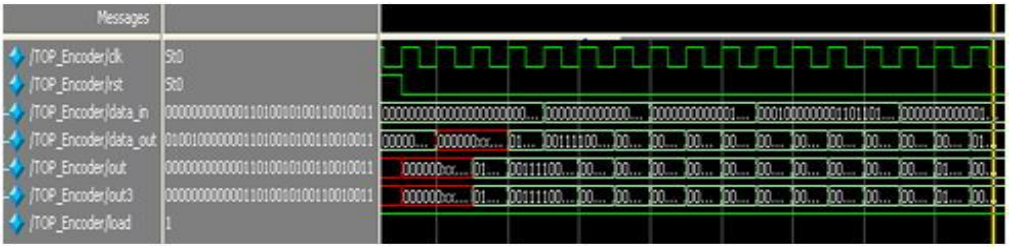

Fig. 7.Test wave forms for Encoding block

Fig. 8.Test wave forms for Decoding block

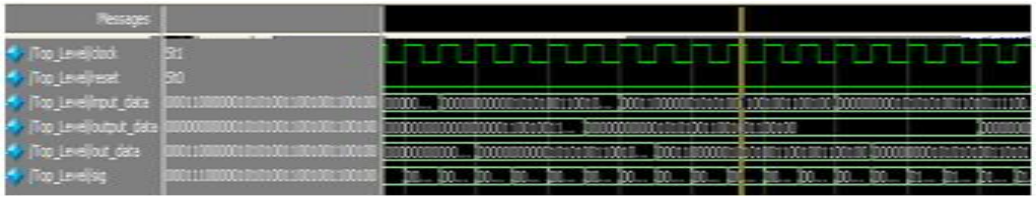

Fig.9.Wave forms for Top-level block

To investigate the advantages of using our proposed technique in terms of area overhead against "Fully ECC" and against the partially protection, we implemented and synthesized for a Xilinx XC2V600 different versions of a 64-bit, 64-entry, dual read ports, single write port register file. Fig. 10 shows the comparison results in terms of RVF reduction and area overhead. As is noticed, our 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 our technique occupies 63% less area. On the other hand, protecting 4 registers "4ECCs" comes with an area overhead similar as our technique but our technique achieves 1.3X improvement in terms of RVF reduction. Since the main target of this paper 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 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 our proposed architecture is shorter. Consequently, our technique improves the performance compared to other competitors. As shown Fig. 11, our technique comes with a minimum impact on both performance and power. It achieves 54% delay reduction

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 4, April 2013

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 our technique achieves a 47% performance improvement and consumes 87% less power. On the other hand, our technique consumes 75% less power and achieves 29% improvement in terms of delay overhead compared to "4ECCs" [4]. It can be concluded that our technique achieves the best overall result compared to state-of-the-art in register file vulnerability reduction.

VI. CONCLUSION

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

VII.ACKNOWLEDGMENT

I would like to thank P.Srinivas, HOD, ECE Department who had been guiding through out to complete the work successfully, and would like to thank ECE Department Professors for extending their help & support in giving technical ideas about the paper and motivating to complete the work effectively & successfully.

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.[4] We assume that generating/checking ECC operations are not performed inparallel to the write/read operations.

[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-error rate testing of CMOS 130nm SRAM: Altitude versus underground measurements," 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 whitepaper on the benefits of Chippkill-Correct ECC for PC server main memory," in IBM Microelectonics division Nov 1997.

[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. 421–463, 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 IEEE Micro, 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.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 4, April 2013

[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,

[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 90nm Technology," in MWSCAS'06, pp. 515-519, 2006.

BIOGRAPHY

Supriya sarkar was born in India, Tripura, in 1980. He received the B.E degree from SRTMU , Nanded, Maharashtra, India in 2006, he has six year teaching experince, presently preserving his M.Tech from RGPV, Bhopal, India. His research interests include VLSI and embedded systems **.**