ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Self-Immunity Technique to Improve Register File Integrity Against Soft Errors

Parasa Ram Babu
M.Tech(VLSI & ES),
SR International Institute of Technology.

Mr.K.Kotaiah, M.Tech
Asst Prof,
SR International Institute of Technology.

Abstract:

Continuous shrinking in feature size, increasing power density etc, increase 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 paper 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 paper deals with the difficulty to exploit this obvious observation to enhance the register file integrity against 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. 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.

Keywords:

Self-Immunity, State-of-the-art, File Protection.

I.INTROCUTION:

In the early days of computers, glitches were an accepted way of life. Since then, as computers have become more reliable (and more relied upon), glitches are no longer acceptable yet they still occur. One of the most intractable sources of glitches has been the transient bit-flip, or soft memory error: a random event that corrupts the value stored in a memory cell without damaging the cell itself.

Soft errors in electronic memory were first traced to alpha particle emissions from chip packaging materials. Since then, memory manufacturers have eliminated most alpha particle sources from their materials, changed their designs to make them less susceptible (e.g., moved ball-grid bumps farther away from memory cells), and even added shielding (usually internal die coatings). Tests and standards have been developed to measure and improve the resistance of memory chips to alpha particles – but soft errors have not disappeared. Further testing, mostly performed by avionics and space organizations, pinpointed a more pernicious source of soft errors: cosmic rays. At ground level, cosmic radiation is about 95% neutrons and 5% protons.These particles can cause soft errors directly; they can also interact with atomic nuclei to produce troublesome short-range heavy ions. Cosmic rays cannot be eliminated at their source, and effective shielding would require meters of concrete or rock.

To eliminate the soft memory errors that are induced by cosmic rays, memory manufacturers must either produce designs that can resist cosmic ray effects or else invent mechanisms to detect and correct the errors. 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. 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. Researchers have mainly and traditionally focused on mitigating soft errors in memory and cache structures, due to their large sizes.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Devices are increasingly vulnerable to soft errors as their feature sizes shrink. Previously, soft error rates were significant primarily in space and high-atmospheric computing. Modern architectures now use features so small at sufficiently low voltages that soft errors are becoming important even at terrestrial altitudes. Due to their large number of components, supercomputers are particularly susceptible to soft errors. Since many large scale parallel scientific applications use iterative linear algebra methods, the soft error vulnerability of these methods constitutes a large fraction of the applications' overall vulnerability. Many users consider these methods invulnerable to most soft errors since they converge from an imprecise solution to a precise one. However, we show in this paper that iterative methods are vulnerable to soft errors, exhibiting both silent data corruptions and poor ability to detect errors.

II. SOFT ERROR BACKGROUND AND TERMINOLOGY:

A.MTBF and FIT

Vendors express an error budget at a reference altitude in terms of Mean Time Between Failures (MTBF). Errors are often further classified as undetected or detected. The former are typically referred to as silent data corruption (SDC); we call the latter detected unrecoverable errors (DUE). Note that detected recoverable errors are not errors. For example, for its Power4 processor-based systems, IBM targets 1000 years system MTBF for SDC errors, 25 years system MTBF for DUE errors that result in a system crash, and 10 years system MTBF for DUE errors that result in an application crash. Note that the processor MTBF must be significantly higher than the system MTBF, particularly for large multiprocessor systems.

Another commonly used unit for error rates is FIT (Error in Time), which is inversely related to MTBF.One FIT specifies one failure in a billion hours. Thus, 1000 years MTBF equals 114 FIT (109 / (24×365×1000)). A zero error rate corresponds to zero FIT and infinite MTBF. Designers usually work with FIT because FIT is additive, unlike MTBF. To evaluate whether a chip meets its soft error budget possibly via the use of error protection and mitigation techniques microprocessor designers use sophisticated computer models to compute the FIT rate for every device RAM cells, latches, and logic gates on the chip.

The effective FIT rate for a structure is the product of its raw circuit FIT rate and the structure's vulnerability factor, i.e., an estimate of the probability that a circuit fault will result in an observable error. The overall FIT rate of the chip is calculated by summing the effective FIT rates of all the structures on the chip. Current predictions show that typical raw FIT rate numbers for latches and SRAM cells vary between 0.001 -0.01 FIT/bit at sea level.

The FIT/bit is projected to remain in this range for the next several technology generations, unless microprocessors aggressively lower the supply voltage to reduce the overall power dissipation of chip. The total FIT contribution of logic gates today is a negligible fraction of the FIT contribution from latches, so we concern ourselves only with faults due to strikes on latches and SRAM cells. In the future, if the contribution of logic becomes non-negligible, we could incorporate the effective FIT rate due to a logic block into the FIT rate of the latch that it feeds.

B.Vulnerability Factors:

The effective FIT rate per bit is influenced by several vulnerability factors (also known as derating factors or soft error sensitivity factors). In general, a vulnerability factor indicates the probability that an internal fault in a device's operation will result in an externally visible error. For example, when a level-sensitive latch is accepting data rather than holding data, a strike on its stored bit may not result in an error, because the erroneous stored value will be overridden by the (correct) input value. If the latch is accepting data 50% of the time, this effect results in a timing vulnerability factor for the latch of 50%. For simplicity, we assume this timing vulnerability factor is already incorporated in the raw device fault rate.

The computation of the device fault rate also includes some circuit-level vulnerability factors. The architectural vulnerability factor (AVF) expresses the probability that a visible system error will occur given a bit flip in a storage cell. The AVF can have a significant impact on the effective error rate of a processor. Prior studies with statistical fault injection into RTL models have demonstrated AVFs of 1%- 10% for latches and 0% - 100% across a range of architectural and micro architectural state bits.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

III.PROPOSEDSELF-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 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 log2k). Thus, the code word will be (k + log2k + 1). In our 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 + log2k + 1 ≤ w). Thus, the optimal value of k is 26 in 32-bit architectures and 57 in 64-bit architectures. For instance, when studying 32-bit architectures, where each register can represent a 32-bit value, we may exploit the register values, which require less than or equal to 26 bits by storing the corresponding ECC bits in the upper unused six bits of that register to enhance the register file immunity against soft errors1. We call this technique Self-Immunity and we call such values 26-bit values. On the other hand, we call register values which need more than 26 bits to be represented over-26-bit register values. Figure-1 shows the percentage of register values usage for different applications of the MiBench Benchmark compiled for MIPS architecture.

Fig1: “26-bit” register values and “over-26-bit” register values in different benchmarks.

As it can be noticed, in all benchmarks most of the register values are 26-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 26-bit register values in the total vulnerable intervals is much more than the contribution of over-26-bit register values. In Figure-1, the fraction of vulnerable intervals of each benchmark is reported. As is demonstrated, the fraction of vulnerable intervals of 26- bit values is 93% on average.

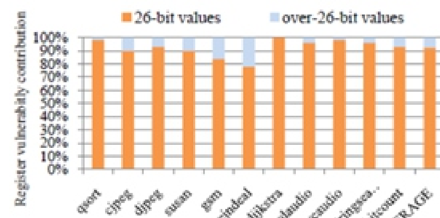

Fig2. The fraction of vulnerable intervals of “26-bit” register values and “over-26-bit” register values in different benchmarks.

A. Problem Description

1. Goal:

The goal of our technique is to reduce the register 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 (Σ i=1 Vi). Since the power overhead mainly stems from accessing the encoder and decoder, it can approximately be modeled through the number of accesses. Let M is the number of protected register values and A the number of accesses, then the total power overhead can be estimated as (Σ i=1 Ai). Thus, our goal can be formulated as:

$$minimize\ (v = \sum_{i=1}^{N} vi), minimize\ (v = \sum_{i=1}^{N} ai)\ (1)$$

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 32-bit architectures, 93% (on average) of the total vulnerable intervals are vulnerable intervals of valid register values for our technique.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

In other words, around 93% of vulnerable intervals will potentially be invulnerable. Thus, our technique promises to reduce the vulnerability of the register file considerably.

3. Architecture for Our 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 26-bit register values from over-26- bit register values. To do that, a self-π bit is associated with each register and we initially clear all 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.

i) Writing into a register:

Figure-3 illustrates that whenever an 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 (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.

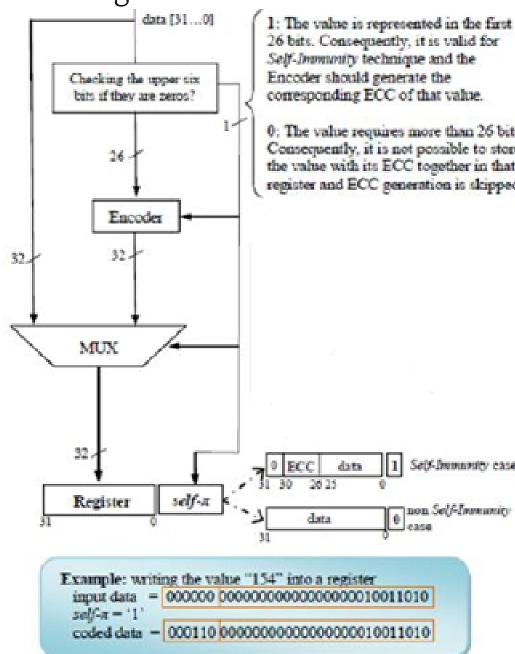

Fig3. Micro-architectural support for writing a register value.

ii) 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 Figure-4.

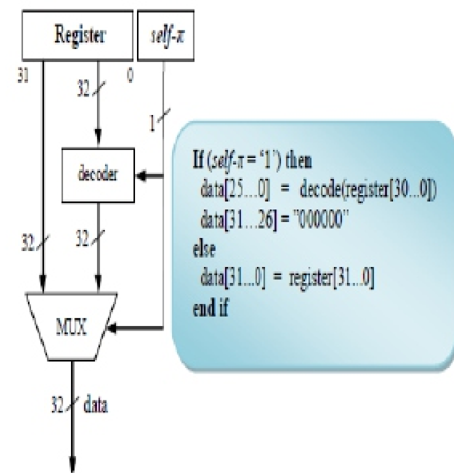

Fig4. Micro architectural support for reading a register value.

4. Potential Power Saving:

In our proposed 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 our proposed architecture because the encoding and decoding operations are performed only in the case of 26-bit register values.

Figure-5 demonstrates that on average 12% and 13% of the total number of read and write operations, respectively, are occurred in the case of over-26-bit register values. As a result, our proposed architecture may consume less power because the encoder and decoder are lesser times accessed.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

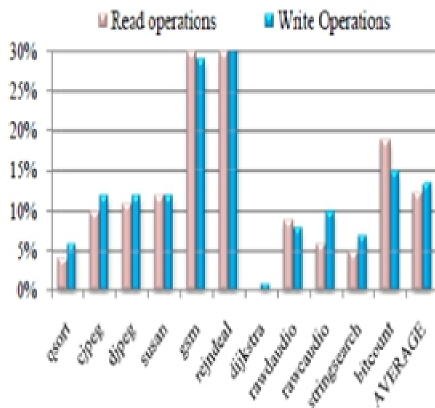

Fig5. The percentage of read and write operations in the case of “over-26-bit” register Values.

Since the input of the deployed encoder in our architecture is 26 bits instead of 32 bits, it generates 5 parity bits instead of 6 parity bits. Likewise, the used decoder in our architecture takes 31 bits (26 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. RESULTS:

We have coded the self immunity technique in Verilog HDL using the proposed self immunity technique design for bit-width 32. All the de-signs are synthesized in the Xilinx Synthesis Tool and Simulated using Xilinx ISE simulator, the area, delay and power values are compared with conventional self immunity technique.

The synthesis result confirms that the proposed self immunity technique involves significantly high performance and accurate than the existing designs.

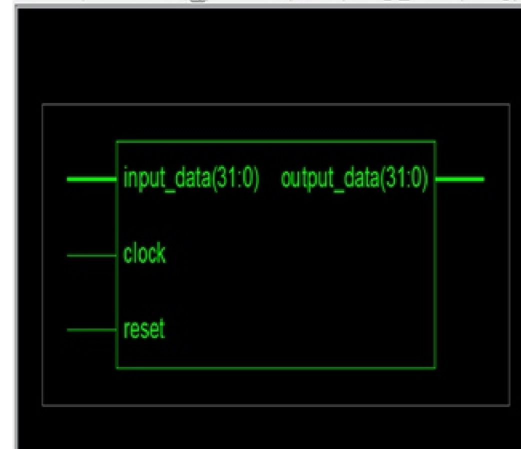

Fig6. RTL Schematic ofthe proposed self immunity technique.

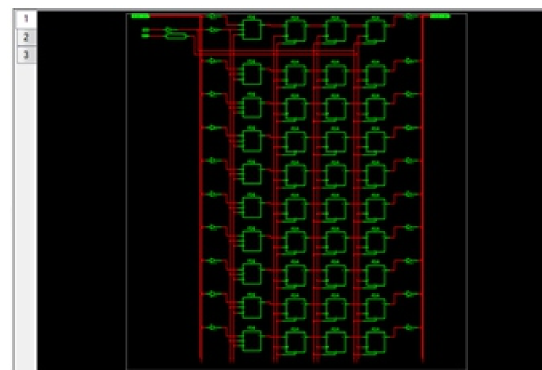

Fig7. RTL Schematic of the proposed self immunity technique.

SELFIMMUNITY Project Status			
Project File:	selfimmunity.ise	Current State:	Synthesized
Module Name:	Top_Level	• Errors:	No Errors
Target Device:	xc3s500e-5fg320	• Warnings:	97 Warnings
Product Version:	ISE 9.2i	• Updated:	Wed Jul 1 23:34:33 2015

SELFIMMUNITY Partition Summary
No partition information was found.

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	60	4656	1%
Number of Slice Flip Flops	104	9312	1%
Number of 4 input LUTs	1	9312	0%
Number of bonded IOBs	60	232	25%
Number of GCLKs	1	24	4%

Detailed Reports					
Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Wed Jul 1 23:34:29 2015	0	97 Warnings	4 Infos
Translation Report					
Map Report					

Fig8. Summary of the proposed self immunity technique.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

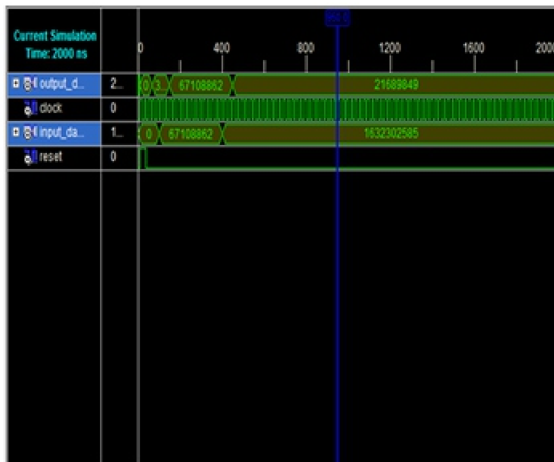

Fig9. Summary of the proposed self immunity technique.

V. 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-error rate testing of CMOS 130nm SRAM: Altitude versus underground measurements, in ICICDT08, 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.