Single Event Effects

Benjamin William Mezger
Computer Engineering
University of Vale do Itajaí (UNIVALI)
Florianópolis, Brazil
me@benmezger.nl

***Abstract*—With the trend of higher density devices targeting faster processing and lower requirement of electric charge, a comparable amount of charge can be generated in the semiconductor by the passage of cosmic rays or alpha particles. This paper seeks to overview available mitigation for Single Event Effects, at the hardware- and software-level.**

***Index Terms*—Single Event Effect, Radiation**

I. Introduction

With the continuing trend of higher density devices for faster processing and lower requirement of electric charge, a comparable amount of charge can be generated in the semiconductor by the passage of cosmic rays or alpha particles. These charges may, for example, temporarily change memory contents or commands in a given instruction stream. The effects of radiation regarding space-borne electronic systems may penetrate sensitive nodes in these devices and affect its system functions and behavior [1].

The first satellite inconsistency was first reported in 1975, by D. Binder *et. al* on SEU in flip-flops. In 1978, the first SEU was first observed on earth by alpha particles, caused by packaging material in a chip and eventually affecting the Random Access Memory (RAM). In 1979, the first report on SEU due to comic rays was published, and in 1992, the first destructive SEE was observed in a memory on a space operating resource satellite [2].

The phenomenon of Single Event Effect (SEE) arises when a single energetic particle penetrates these sensitive nodes, causing glitches to the electronic system or catastrophic failures at the circuit level [1]. With a variety of possible SEE, they can either be transient, permanent, or intermittent.

Faults that may affect the system during its lifetime can be classified into eight basic viewpoints of phenomenological causes, being of: (i) natural faults, caused by natural phenomena without human interaction; (ii) human-made faults, resulted by human interaction such as production defects; (iii) transient faults, presented within a bounded time-frame; and (iv) permanent faults, given within a continuous time-frame [3]. This paper aims at reviewing transient fault mitigation.

During the system operation, natural faults can be either internal, due to the natural process of physical deterioration, or external, due to the natural process that happens outside the system boundaries and may cause hardware interference [3].

In fault-tolerant architecture, a fault is a physical defect, such as a broken transistor. These faults may manifest themselves as an error, such that having a bit 0 in place of a bit 1, or by not manifesting itself as an error. An error can be masked or can result in a user-visible failure [4].

A fault and/or error does not necessarily become an error and/or a fault, respectively. This can be mitigated by masking the system at the design level. The effect of an error at a logical level may not affect the system, and may not propagate to the architectural level either, as it depends on which instruction the error will impact. Errors that propagate to the application level may not be impacted by an error either, as the error may affect an unused memory location by the application and never gets triggered [4].

A transient fault may occur once and not persist across the system, these are often referred to as *soft error* or as Single Event Upset. Permanent faults are often called *hard fault*, and persists when the fault occurs and may manifest itself as a repeated error. An intermittent fault occurs repeatedly but not over the same place in the system [4].

Radiation device hardening and SEE fault tolerance approaches have been taken to mitigate these issues when they arise [1], however, the mitigation approaches are dependent on their fault duration, as tolerating a transient fault requires no self-repair due to its non-persistence. Fault tolerance schemes may treat intermittent faults as either transient or permanent, depending on how often they occur in the system [4].

Due to the many physical phenomena that may lead to a fault, a variety of techniques are available for mitigating these issues according to the environment they run. Due to the transient high-energy particles, cosmic rays may produce alpha particles or even electromagnetic interference from outside sources, generating transient faults to the devices [1].

The effects of the fault may change a value of a cell or transistor output. Due to the one-time disruption, the error will vanish once the cell or transistor's output is overwritten. [4] categorizes permanent phenomena into three categories: (i) permanent wear-out, making a processor fail due to several physical issues such as thermal cycling and mechanical stress; (ii) fabrication defects, by manufacturing chips with inherent defects; and (iii) design bugs, such as a chip not behaving correctly due to an internal bug. Some physical phenomena may lead to intermittent faults, such as loss of connection between two wires or devices [4].

This work aims at characterizing the types of SEE and the state-of-the-art that has been accomplished to mitigate these issues at the circuit- and software-level. The rest of the paper is organized as follows: section II gives a brief background over the types of SEE and how they may affect the system, among with fault metrics and types of errors, section III present some techniques for mitigating single events at the circuit level, section IV refers to software-based approaches for single event mitigation. Finally, section V provides final conclusions.

II. Background

With the decrease of dimension size of transistors, wires, and smaller chips, the tendency to transient and permanent faults are much higher, as the dimension of the chip may impact the temperature directly. Given Moore's law increase the number of transistors per chip, more opportunities arise for faults in the field of application and manufacturing. The complexity of processor design increases the likelihood of design bugs during production, which may bring permanent faults to the processor during execution time [4]. This section overviews the types of SEE and how they arise during the life-type of a system.

A. Types of SEE

SEE depends on the interaction of a single particle penetrating the device, which can be caused by the passage of a single heavy ion by a cosmic ray. As cosmic rays are highly energetic in space, they may pass through the device and be collected in the device's electrodes. The ion produces an electric pulse that may appear to the device as if it should respond and eventually causing a failure. High energy protons can also be a cause of failure, as a proton may have a nuclear reaction in the silicon device [1].

SEE has a variety of possible effects, each of which is important, as they cause malfunctioning of devices in space ionizing radiation environment [1]. These SEE is illustrated in table II-A with their respective description.

TABLE I
Types of Single Event Effect [5]

Term	Definition
Single event upset	A change of state or transient induced by an energetic particle
Single hard error	Causes permanent changes to the operation of the device
Single event latch-up	Loss of device functionality induced by high current
Single event burnout	A condition which causes a device to destruct due to high current state in a power transistor
Single event effect	A measurable effect to a circuit due to an ion strike
Multiple bit upset	Event induced by a single energetic particle which may cause multiple upsets or transient
Linear energy transfer	A measure of energy deposited per unit length

B. Fault tolerance metric

Fault tolerance solution requires experiments to test a hypothesis or compare with previous works and knowing which errors may apply within the system. [4] covers several important metrics on fault tolerance systems, those including (i) the availability of the system, by verifying the system is functioning correctly at a specific time; and (ii) reliability, is the probability that the system has been functioning correctly from time zero to a specific time.

C. Error detection

Error detection provides a measure of safety, as it is an important aspect of fault tolerance since the processor cannot tolerate a problem it is not aware of. Redundancy is fundamental to error detection, as it helps the processor detect when a given error occurs. There are three classes of redundancy, spatial, temporal and information redundancy [4].

Spatial redundancy adds redundant hardware to the system. The Dual Modular Redundancy (DMR) is a simple form of spatial redundancy, which provides error detection by using a voter system, which then receives the output of all modules and checks for any error [4].

Temporal redundancy may perform redundant operations, by requiring a unit to operate twice and finally compare the results. Temporal redundancy doubles the amount of time for each operation. However, in comparison to Spatial redundancy, there is no extra hardware or power cost involved. For reducing performance cost, some schemes may use pipelining to hide the latency of a redundant operation [4].

Finally, information redundancy detects when a datum has been affected by adding bits to it. Schemes such as Error-detecting Code (EDC) can be used for such redundancy, for example, by adding a parity bit to a data word and convert into a codeword. The parity scheme is popular, due to its simplicity and inexpensive implementation [4].

D. Error recovery

Error detection is enough for providing safety to the system, but not recovering from the error. By recovering from the error, it hides the effect of the error from the end-user and allows the system to resume operation [4]. Two primary approaches to error recovering is Foward Error Recovery (FER) and Backward Error Recovery (BER).

FER corrects the error without having to revert to a previous state. FER can be implemented through physical, temporal, and information means of redundancy. In FER, if a specific amount of redundancy is required to determine if an error has occurred, then additional redundancy is required to correct the error [4].

BER restores the state of the system to a previously known good state, known as recovery point on single-core systems and recovery-line on multi-core systems. The system architect should think through what state it should be saved for recovery, where and when to deallocate, the algorithm, and what to do after the system has been restored [4].

III. Hardware Mitigation

A. Soft errors

[6] explores four single transient mitigations by evaluating four techniques that can be applied at the circuit level. These techniques are covered in the next sub-sections.

1) Schmitt Triggers: In high noise applications, the Schmitt Triggers (ST) works as a replacement for the internal inverter of a circuit. The ST has a higher dependency over a source-gate voltage of its P1 and N1 transistors, resulting in an enhanced robustness over a Voltage Transfer Curve (VTC) deviation [6].

2) Decoupling Cells: By connecting capacity elements to the most exposed nodes, one can mitigate transient effects. The use of decoupling cells increases the total capacity in the output of a node of the NAND2 gate, resulting in a decrease of critical charge required to produce a single transient pulse, which by effect improves signal degradation [6].

3) Sleep Transistors: Circuit blocks that are not in use can be shut off by using the power-gating strategy, widely used in low-power designs for reducing chip's power consumption. Sleep transistors act as a supply-voltage regulator. When a sleep transistor is in active mode, it improves the process variability of a typical logic gate connection to the ground rail by acting as a voltage regulator. While in standby, the sleep transistor disconnects the virtual ground from the physical ground [6].

4) Transistor Reordering: Optimizing transistors arrangements allows reducing current leakage or dealing with bias temperature instability. This technique modifies the transistor arrangement by still keeping the same functionality that was aimed at. The transistor reordering swaps the electrical and physical characteristics of the logic cells, resulting in susceptibility to soft errors. The robustness of complex gates where can be improved up to 8% by using this approach and can be favorable to improve single effect stability of circuits without including area penalty in complex gates [6].

IV. Software Mitigation

Software approaches can also be used for hardware errors. The primary interest of using a software redundancy is that it brings no hardware cost and requires no hardware modification. The software approach may provide good coverage of possible errors and can be easily tested comparing to hardware approaches. The cost of software redundancy may be significant, as performance may be lost depending on the core model and software workload, as instruction duplication requires more processing [4]. The following presents some solutions for software-based mitigation

A. Selective Code Duplication

In Selective code duplication, only parts of the code are duplicated, and their results are compared, which reduces fault coverage but improves code size and execution time overhead. Multiple techniques use selective code duplication, such as SWIFT, VAR3+, CDB, and SEDSR.

1) Error detection by duplicated instructions: Error detection by duplicated instructions (EDDI) consists of inserting redundant instructions and instructions that also compares the results produced by the original instruction and the redundant instructions [4]. The SWIFT scheme by [7] improved upon EDDI by combining with the control flow checking and optimizing the performance by reducing the number of comparison instructions.

[8] provides a tool named Compiler Assisted Software Fault Tolerance (COAST), which provides an automated compiler modification of software to insert a dual- or triple-modular redundancy. The approach adds data flow protection to user-provided programs. By default, the tool replicates all compute operations and memory loads/stores. while keeping a single set of control flow operations. The COAST tool provides Duplicate With Compare (DWC) and TMR protection mode. The replication and/or synchronization of instructions is fully automated as part of the compilation process. The produced software executable is are more tolerant to SEU, ideal for processing in a high radiation environment. Experiments were conducted in 30R flight path at the Los Alamos Neutron Science Center (LANSCE), and neutron beam tests have shown that COAST provides a significant increase in Mean Work To Failure (MWTF).

2) Error detection by diverse data and duplicated instructions: Error Detection by Diverse Data and Duplicated Instructions (ED^{4}I) is a full code duplication technique, where all instructions in a block are duplicated. Comparison instructions are placed after each original and duplicated instruction in each block to compare their results [9].

3) Overhead reduction: In a Overhead Reduction (VAR3+) technique, all instructions in a block, except for branch and store instructions are duplicated. The comparison instructions have to be placed before load, store, and branch instructions to compare the results [9].

4) Critical block duplication: In Critical Block Duplication (CBD) technique, critical blocks have to be identified in the control flow graph. Any block which has the highest number of fan-outs in the control flow graph is considered a critical block. If any mismatch of results is detected, an error is reported [9].

B. Soft error detection using software redundancy

Soft Error Detection Using Software Redundancy (SEDSR) in an extended version of CBD, however, comparison instructions have to be added after the original and duplicated instruction in each identified block block for comparing results [9].

V. Conclusions

With the continuous trend of smaller chip sizes, the tendency of transient and permanent faults are much higher. This paper sought to characterize the types of SEE and how they affect a system according to the environment, and what metrics are important when considering a fault-tolerant design. By understanding the difference between error detection and error recovery allows one to seek a solution which may fit their requirements. Multiple fields of mitigation's have been reviewed, from a circuit-level techniques to software-level approaches. Although software mitigation usually impacts performance, it is a cheaper alternative in comparison to hardware alternatives.

References

[1] E. Petersen, *Single event effects in aerospace*. John Wiley & Sons, 2011, ch. 1, pp. 1–12. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118084328.ch1

[2] S. Buchner, "Overview of single event effects," in *Proc. 11th Int. School Effects Radiation Embedded Syst. Space Appl.(SERESSA)*, 2015, pp. 62–69.

[3] A. Avizienis, J. . Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan 2004.

[4] D. J. Sorin, *Fault Tolerant Computer Architecture*. Morgan and Claypool Publishers, 2009.

[5] The Radiation Effects and Analysis Group (REAG). (2017) Draft - single event effects specification. National Aeronautics and Space Administration (NASA). Last accessed on 24/01/2021. [Online]. Available: https://radhome.gsfc.nasa.gov/radhome/papers/seespec.htm

[6] R. Reis, C. Meinhardt, A. L. Zimpeck, L. H. Brendler, and L. Moraes, "Circuit level design methods to mitigate soft errors," in *2020 IEEE Latin-American Test Symposium (LATS)*, 2020, pp. 1–3.

[7] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August, "Swift: software implemented fault tolerance," in *International Symposium on Code Generation and Optimization*, 2005, pp. 243–254.

[8] B. James, H. Quinn, M. Wirthlin, and J. Goeders, "Applying compiler-automated software fault tolerance to multiple processor platforms," *IEEE Transactions on Nuclear Science*, vol. 67, no. 1, pp. 321–327, 2019.

[9] V. B. Thati, J. Vankeirsbilck, N. Penneman, D. Pissoort, and J. Boydens, "Cdfedt: Comparison of data flow error detection techniques in embedded systems: An empirical study," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ser. ARES 2018. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3230833.3230854