Center for Reliable Computing

TECHNICAL REPORT

Fault-Tolerant Computing for Radiation Environments

Philip P. Shirvani

01-6

June 2001

Center for Reliable Computing
Gates Room # 239, MC 9020
Gates Building 2A
Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305

Abstract:

This technical report contains the text of Philip Shirvani's Ph.D. thesis "Fault-Tolerant Computing for Radiation Environments."

Funding:

This work was supported in part by the Ballistic Missile Defense Organization, Innovative Science and Technology (BMDO/IST) Directorate and administered through the Department of the Navy, Office of Naval Research under Grant Nos. N00014-92-J-1782 and N00014-95-1-1047, and in part by the National Aeronautics and Space Administration (NASA) and administered through the Jet Propulsion Laboratory (JPL), California Institute of Technology under Contract No. 1216777.

FAULT-TOLERANT COMPUTING FOR RADIATION ENVIRONMENTS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Philip P. Shirvani

June 2001

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____
Edward J. McCluskey (Principal Advisor)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____
John T. Gill III

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____
Nirmal R. Saxena

Approved for the University Committee on Graduate Studies:

_____

To my parents, Satehe and Claude,
and to my sister, Vivian,
for their love and support

In memory of
my father, Claude and
my brother-in-law, Koorosh Malek

Abstract

Radiation, such as alpha particles and cosmic rays, can cause transient faults in electronic systems. Such faults cause errors called Single-Event Upsets (SEUs). SEUs are a major source of errors in electronics used in space applications. There is also a growing concern about SEUs at ground level for deep submicron technologies. In this dissertation, we compared different approaches to providing fault tolerance against radiation effects and developed new techniques for fault tolerance and radiation characterization of systems.

Estimating the SEU error rate of individual units of a digital circuit is very important in designing a fault-tolerant system and predicting its overall error rate. In this work, a new software method is developed that uses weighted test programs in conjunction with multiple linear regression for SEU characterization of digital circuits. We also show how errors in bistables can be distinguished from errors in combinational logic by operating a sequential circuit at different clock frequencies.

Radiation hardening is a fault avoidance technique used for electronic components used in space. However, these components are expensive and lag behind today's commercial components in terms of performance. Using Commercial Off-The-Shelf (COTS) components, as opposed to radiation-hardened components, has been suggested for providing the higher computing power that is required for autonomous navigation and on-board data processing in space. In this work, these two approaches are compared in an actual space experiment. We collected errors from two processor boards, one radiation-hardened and one COTS, on board the ARGOS satellite. We designed and implemented a software-implemented Error Detection and Correction (EDAC) scheme for main memory. We tested this scheme on the COTS board and showed that, in the absence of hardware EDAC, system reliability can be significantly improved by providing protection through software. We demonstrated that the reliability of COTS components can be enhanced by using software techniques for detecting, correcting and recovering from errors without changing the hardware. Despite the 170% time overhead of the software techniques used on the COTS board, the throughput of the COTS board was an order of magnitude higher than that of the radiation-hardened board. The throughput of the radiation-hardened board would be the same as that of the COTS board if the radiation-hardened board had cache memory.

We also developed a new technique for tolerating permanent faults in cache memories. The main advantage of this technique is its low performance degradation even in the presence of a large number of faults.

Acknowledgments

I express my deep gratitude to my advisor, Prof. Edward J. McCluskey for his guidance, support, and encouragement during my years at Stanford. His advice and insights guided me throughout my Ph.D. study and will continue to guide me through my professional career. I would also like to thank his wife, Mrs. Lois Thornhill McCluskey.

I would like to thank Prof. Oyekunle A. Olukotun for being my associate advisor, Prof. John T. Gill III and Dr. Nirmal R. Saxena for being on my orals and reading committee, and Prof. Umran S. Inan for being the chairman of my orals committee. Special thanks to Dr. Nirmal R. Saxena for his advice and guidance on my studies and research.

I wish to thank Dr. Kent Wood, Dan Wood, Mike Lovellette, and Ganwise Fewtrell of Naval Research Laboratory (NRL) for the collaboration on the ARGOS experiment and Dr. Alan Ross of Naval Post-graduate School (NPS) for his support in the ARGOS project and retrieval of a processor board.

I would like to thank my current and former colleagues at the Center for Reliable Computing (CRC) for helpful discussions, critical reviews of papers and technical reports, and companionship through the years: Khader Abdel-Hafez, Ahmad Al-Yamani, Mehmet Apaydin, LaNae Avra, Jonathan T.-Y. Chang, Ray Chen, Eddie Cheng, Kan-Yuan Cheng, Santiago Fernandez-Gomez, Vlad Fridman, Robert Huang, Rajesh Kamath, Erin Kan, Sam Kavusi, Chat Khunpitiluck, Wern-Yan Koe, Sunil Kosalge, Vincent Lo, James Li, Siyad Ma, Samy Makar, Subhasish Mitra, Robert Norwood, Nahsmuk Oh, Mehdi Tahoori, Rudy Tan, Nur Touba, Chao-Wen Tseng, Sanjay Wattal, Sungroh Yoon, Yoonjin Yoon, and Catherine Yu. I am greatly indebted to Siegrid Munda for her excellent administrative support. I wish to thank the CRC visitors Prof. Jacob Abraham and Prof. Bella Bose for their valuable suggestions and advice. I also thank Dr. Saied Bozorgui-Nesbat and Dr. Javad Khakbaz for their advice and encouragement.

I thank all my teachers and professors to whom I will always be in debt. I especially thank Dr. Parirokh Dadsetan.

I would like to thank all my friends for their support and encouragement. They have provided a refuge for me away from the pressures of work and research, and have helped me maintain a sense of balance in my life. Special thanks to Vida Vakilotojar for reviewing this dissertation and for her help in preparation for my orals. I also thank Stanford's Persian Student Association (PSA) for making our lives more colorful and enjoyable at Stanford.

I am greatly indebted to Dr. Siavash Honari for his generosity, support and kindness without which my studies at Stanford might have never happened. Special thanks also go to Farideh and Masud Mehran for their generosity, support and care.

Finally, I would like to thank my parents and my sister for their endless love, support, and encouragement. I dedicate this dissertation to them. I thank my aunt, Nilla Fiouzi, for her love and her support during the application process of my graduate studies in the United States. I also thank my aunt, Francine Shirvani, for her love and care.

This work was supported in part by the Ballistic Missile Defense Organization, Innovative Science and Technology (BMDO/IST) Directorate and administered through the Department of the Navy, Office of Naval Research under Grant Nos. N00014-92-J-1782 and N00014-95-1-1047, and in part by the National Aeronautics and Space Administration (NASA) and administered through the Jet Propulsion Laboratory (JPL), California Institute of Technology under Contract No. 1216777.

Table of Contents

List of Figures

Number *Page*

List of Tables

Number *Page*

List of Acronyms

API	Application Program Interface
ARGOS	Advanced Research and Global Observation Satellite
CFCSS	Control Flow Checking by Software Signatures
COTS	Commercial Off-The-Shelf
EDAC	Error Detection And Correction
EDDI	Error Detection by Duplicated Instructions
FT	Fault Tolerance
ISA	Instruction Set Architecture
LEO	Low-Earth Orbit
LET	Linear Energy Transfer
MBU	Multiple-Bit Upset
MEO	Medium-Earth Orbit
OS	Operating System
PAD	Programmable Address Decoder
SAA	South Atlantic Anomaly
SEC-DED	Single-Error Correction, Double-Error Detection
SEE	Single-Event Effect
SEL	Single-Event Latchup
SER	Soft Error Rate
SEU	Single-Event Upset
SIHFT	Software-Implemented Hardware Fault Tolerance
SOI	Silicon On Insulator
WTP	Weighted Test Programs

Chapter 1
Introduction

1.1 Background

Electronic systems and computers have become an integral part of our everyday life. We rely on computers for doing many critical tasks. We rely on them when we fly in an airplane, ride a train, or drive a car. Banking systems use computers for storing records and executing transactions. Many medical instruments and devices have embedded computers. All these critical tasks require systems with high levels of reliability, availability and serviceability (RAS) that can provide continuous correct operation [Mueller 99]. RAS is also important in non-critical tasks for product quality and costumer satisfaction.

Failure of a device or component in a system might cause the system to function incorrectly and fail to provide the intended service. A *failure* is the deviation of a device from the specified characteristic. A *fault* models the effect of failure on logical signals. An *error* is the manifestation of a fault at system outputs. There are two approaches for avoiding system failures: fault avoidance and fault tolerance. *Fault avoidance* techniques try to reduce the probability of fault occurrence, while *fault tolerance* techniques try to keep the system operational despite the presence of faults. Since faults cannot be completely eliminated, critical systems always employ fault tolerance techniques to guarantee high reliability and availability.

Fault tolerance techniques are based on redundancy. Redundancy is provided by extra components (hardware redundancy), by extra execution time (time redundancy), or by a combination of both. For example, in the triple modular redundancy (TMR) technique, hardware is replicated so that errors can be detected and corrected. Cost, area, weight, speed (response time and throughput) and power consumption are constraints that limit the amount of fault avoidance and fault tolerance features that can be employed in a system. To maximize the reliability improvement obtained by fault avoidance and tolerance techniques, the most vulnerable (fault-prone) components should be addressed first. In this work, we take this approach and investigate fault tolerance techniques that could enhance system reliability without high cost. We also explore methods to identify the most vulnerable components.

Faults are either permanent or temporary. *Permanent faults* happen when a part fails and needs to be replaced. A system can be designed with spare parts to automatically replace the failed part, or it can be designed so that it can continue its operation without the failed part, perhaps at lower performance, a technique called

graceful degradation. The latter case is possible when the task of the failed part is not essential or can be carried out by another part. This usually happens when redundant parts are present in the system to provide higher performance. Cache memory of a processor is one such case. As part of the work in this dissertation, we present a new technique for tolerating permanent faults in caches that exploits the inherent redundancy in caches.

Transient errors occur in the system temporarily and are usually caused by interference. Radiation, such as alpha particles and cosmic rays, is one source of transient errors in electronic systems. These errors are called *Single-Event Upsets* (SEUs). SEUs are a major cause of concern in a space environment, and have also been observed at ground level [Ziegler 96a]. An example effect is a bit-flip — an undesired change of state in the content of a storage element. These errors are also called *soft errors* as opposed to permanent errors that are sometimes called *hard errors*. A single particle can upset multiple adjacent nodes in a circuit and cause a *Multiple-Bit Upset* (MBU)[1].

Radiation is a major source of transient and permanent faults in electronics used in environments such as space. Radiation hardening is a fault avoidance technique for reducing the sensitivity of electronic components to radiation. However, radiation hardened components are very expensive and lag behind today's commercial components in terms of performance. The need for low-cost, state-of-the-art high performance computing systems in space has created a strong motivation for investigating new fault tolerance techniques. Using unhardened *commercial off-the-shelf* (COTS) components, as opposed to radiation-hardened components, has been suggested for building cheaper and faster systems [LaBel 96]. The issue for COTS components is that they have limited fault avoidance and fault tolerance features. Software-implemented hardware fault tolerance (SIHFT) techniques provide low-cost solutions for enhancing the reliability of these systems without changing the hardware.

Many experiments have been conducted to test the feasibility of using COTS components in space. For example, the use of COTS for commercial microsatellites were studied at the University of Surrey through a set of satellite experiments [Underwood 98]. Microelectronics and Photonics Test Bed (MPTB) is another on-going satellite experiment for verifying the suitability of a 32-bit COTS microprocessor, a variety of memories, and photonic devices for space applications [Dale 95] [MPTB].

[1] MBUs can occur in up to 10% of SEU events [Oldfield 98]. Therefore, they should be considered in the design of an error detection and correction technique.

The work presented in this dissertation was done under two fault tolerance projects at Stanford's Center for Reliable Computing (CRC): the ARGOS project and the REE project. The challenge that was given to us in the Advanced Research and Global Observation Satellite (ARGOS) project was to determine to what extent commercial electronics, e.g., microprocessors and RAMs, can be used in space. The approach we adopted for this challenge was, first, to design an experiment to collect data on board an actual satellite so that we can compare radiation-hardened components with COTS components and evaluate different fault tolerance techniques, and second, to develop techniques for fault tolerance that are software based and do not require any special hardware.

The other fault tolerance project at CRC was NASA's Remote Exploration and Experimentation (REE) project [Beahan 00] [Chen 00] [REE]. The goal of this project was to develop a fault-tolerant, high-performance, low-power supercomputer that enables the collected data on a spacecraft to be processed in space rather than on the Earth. Part of this project involved SEU characterization of processors and estimating the relative error rates in different functional units. The estimations would serve two purposes: (1) identification of the most SEU sensitive units, and (2) prediction of failure rate of different programs that have different utilization of the functional units. Another goal of the REE project was to develop and evaluate fault tolerance techniques.

This dissertation summarizes the results of my research in these two projects and my contributions to the field of fault tolerant computing. Section 1.2 briefly introduces each part of my work and presents a list of my contributions. The outline of the remainder of the dissertation is provided in Sec. 1.3.

1.2 Contributions

The contributions of this dissertation relate to both permanent and transient faults in computer systems. This section briefly explains these contributions.

Caches occupy a substantial area of a microprocessor chip and contain a high percentage of the total number of transistors on the chip. Consequently, the reliability of the caches has a big impact on the overall chip reliability. Previous fault-tolerance techniques for caches are limited either by the number of permanent faults that can be tolerated or by the rapid performance degradation as the number of faults increases. We present a new technique that overcomes these two problems. We propose a reconfiguration technique that can keep the cache operational at a high level of performance (low miss rate) even in the presence of a large number of faults. This technique uses a special *Programmable Address Decoder* (PAD) to disable faulty blocks

and to re-map their references to fault-free blocks. The main advantages of our technique over previous techniques are: (1) low performance (cache hit-rate) degradation of the cache even in the presence of a large number of faults, and (2) low sensitivity of performance to fault locations.

Memories contain most of the transistors in a computer system and are a major contributor to the reliability of the system. Bit-flips caused by SEUs are a well-known problem in memory chips, and *error detection and correction* (EDAC) codes have been an effective solution to this problem. These codes are usually implemented in hardware using extra memory bits and encoding & decoding circuitry. In systems where EDAC hardware is not available, the reliability of the system can be improved by providing protection through software. We look at the implementation requirements and issues and present some solutions. We discuss how system-level and chip-level structures relate to multiple error correction including MBUs[2]. A technique is proposed to make the EDAC scheme independent of these structures. The technique was implemented and used effectively in an actual space experiment (ARGOS experiment). The reliability improvement is demonstrated through both a satellite experiment and analytical estimates that are based on parameter values that closely match the environment of the satellite experiment.

Once fault tolerance is provided for memories, the focus turns to other components of the system, such as the processor. Processors are complex digital circuits with many different functional units. Estimating the SEU cross section[3] of individual units of a digital circuit is very important in designing a fault tolerant system and predicting its overall error rate. Hardware approaches to this SEU characterization are expensive and sometimes intrusive. Previous software approaches tend to use test programs targeting specific units [Hiemstra 99] [Kouba 97]. However, such methods are not suitable for all types of functional units because several units are always used during the execution of any program. Moreover, since the same error can be caused by many different faults, error classification can only do a crude association of errors with functional units. Because of these limits, previous methods do not always yield good estimates. In this thesis, a new method is proposed that uses weighted test programs in conjunction with multiple linear regression to alleviate some of the limits of previous software approaches. We also propose using the clock frequency dependency of SEU

[2] In some cases, hardware EDAC may fail in correcting MBUs.

[3] Cross section is a measure of sensitivity of a circuit to radiation. The precise definition is given in Chap. 2.

cross section in separating the cross section of bistables and combinational logic in a sequential circuit.

In the ARGOS project, we collected data on occurrence of SEU errors and the effectiveness of fault tolerance techniques in detection and recovery from errors. In this project, a radiation hardened processor board and a COTS processor board were tested simultaneously in space, on board the ARGOS satellite that was launched in February 1999 [Shirvani 98] [Shirvani 00a]. Earlier experiments on evaluation of SIHFT techniques have relied on artificial fault injection, which may not fully represent the condition in an actual space environment. In the ARGOS project, we collected data in an actual space environment thereby avoiding the necessity of relying on questionable fault injection methods. While the goal of most of the other experiments is to measure the error rate in different components, in the ARGOS project we also evaluated various SIHFT techniques to assess their effectiveness in providing reliability for COTS that are subject to SEU errors. We developed, implemented, and tested many software-implemented error detection techniques that do not require any hardware assistance. To enable continuous error data collection without ground intervention, we designed an error recovery mechanism without modifying the operating system of the board. Our results show that COTS with SIHFT are viable techniques for low radiation environments.

The contributions of this work are summarized below.

- Fault tolerance for cache memories:
  - We developed a new fault-tolerance technique for cache memories called *PADded Cache*. We evaluated this technique by simulation and estimated its area overhead for an example design.
- Fault tolerance for main memories:
  - We designed and implemented a software-implemented EDAC technique that combines vertical codes and interleaving to tolerate single and multiple errors, as well as MBUs, independent of system-level and chip-level structures.
  - We designed a self-repairing and recovery mechanism for the software-implemented EDAC program that provides protection for the program itself and also recovers from hang-ups in this program.
  - We demonstrated the effectiveness of software-implemented EDAC in an actual space experiment.
- SEU characterization of digital circuits:
  - We developed a new method for SEU characterization of digital circuits using weighted test programs. This method can provide better estimation of individual

cross sections than previous methods, it does not depend on error classification, and can use linear regression to provide more accurate estimations.

- We developed a technique for separating the cross section of bistables and combinational logic in SEU characterization of sequential circuits using clock frequency dependency of SEU cross section.

- ARGOS experiment:
  - We designed and implemented the experiment that was conducted on board the ARGOS satellite.
  - We collected and analyzed the data from the ARGOS satellite. We measured the error detection coverage and error recovery success rate of software-implemented hardware fault tolerance (SIHFT) techniques and demonstrated the effectiveness of SIHFT in an actual space experiment. We measured the SEU and MBU rates of memories and showed the pattern dependency of SEUs
  - We evaluated and compared different fault tolerance technique in an actual space environment. We compared a radiation-hardened processor board and a COTS processor board in space and demonstrated that COTS components can be a low-cost substitute for radiation-hardened components in a low radiation environment.

The trend in integrated circuits technologies has been towards smaller feature sizes, lower supply voltages, and higher frequencies. There is a growing concern about the sensitivity of deep submicron technologies to radiation. The issue of soft errors on ground is currently being addressed by industry for commercial parts [Dai 99][Hareland 00][Seifert 01]. Therefore, the SEU related work presented in this dissertation is important for terrestrial applications as well as avionics and space applications.

1.3 Outline

This dissertation summarizes my work in fault tolerance techniques for radiation environments. Detailed description of each topic can be found in the appendices. The appendices are technical reports at Stanford Center for Reliable Computing (CRC), some of which are extended versions of published conference and journal papers.

This dissertation is organized as follows. Chapter 2 provides the background on radiation-related issues and terminology in electronic circuits. Chapter 3 reviews previous fault tolerance techniques for cache memories and introduces PADded caches. We discuss the advantages of PADded caches over previous techniques and show a selection of the simulation results. More detailed analysis and simulation results are presented in Appendix A.

Chapter 4 discusses implementation of EDAC in software and presents our technique along with analytical and experimental results. Detailed analysis and discussion of software-implemented EDAC is provided in Appendix B.

In Chap. 5, a new software technique for SEU characterization of digital circuits is introduced. Previous work in this field is reviewed and the advantages and limits of the proposed technique are discussed. Detailed discussion and the simulation setup and results can be found in Appendix C.

In Chap. 6, the experimental results of our project on board the ARGOS satellite is presented. More detailed explanation of these experiments can be found in Appendix D. Chapter 7 concludes the dissertation.

Chapter 2
Radiation and Its Effects on Electronic Systems

This chapter presents a summary of the basics of radiation science and its relation to electronic systems. The objective of this chapter is threefold. First, it provides background information on radiation-related issues in electronic systems. Second, it puts the thesis in perspective and shows how each of the following chapters addresses the radiation issues. Third, it describes the environment of the ARGOS satellite that was used for the space experiments described in Chap. 6.

2.1 Radiation Types

Radiation can be classified into two major types: (1) energetic particles such as electrons, protons, neutrons, α-particles and heavy ions, and (2) electromagnetic radiation (photons), which can be X-ray, gamma ray, or ultraviolet light. Each of these radiations exists in different densities and energy ranges in different environments (energy is typically measured in electron volts, eV). Electrically charged particles such as electrons, protons, α-particles and heavy ions interact with electromagnetic fields. One such field is Earth's magnetic field, which diverts and traps charged particles. On the other hand, neutrons and photons are not affected by Earth's magnetic field.

2.2 Radiation Environments

We distinguish two radiation environments: terrestrial (on the Earth) and extra-terrestrial (space), and look at sources of radiation that we should be concerned about in each of them.

2.2.1 Terrestrial Environments

There are two main sources of radiation on the Earth: radioactive material and cosmic rays. The α-particles produced during a radioactive decay process can affect electronic circuits. Residual traces of radioactive material may be found in integrated circuits (ICs), for example, Polonium 210, Uranium 238 and Thorium 232 have been found in aluminum, gold, processing chemicals, ceramic packages or lead solder bumps [Hasnain 92][Ziegler 96a]. Electronic circuits may be exposed to radioactive material if they are operated in a nuclear power plant. The strong radiation blast produced by a nuclear weapon may also be a cause of concern.

Cosmic rays consist of galactic cosmic rays, which are very energetic particles from distant sources in the galaxy, and solar wind, which is a flood of relatively low-

energy particles emitted from the Sun. The Earth's magnetosphere shields part of the radiation that hits the Earth. The part that passes through this shield hits the atmosphere and interacts with molecules in the atmosphere. Secondary particles are produced as a result of these interactions, which may lead to further interactions and particles. The atmosphere shields the Earth from primary particles. However, the induced secondary particles may reach ground levels and affect electronic circuits. Particles of interest at high altitudes are protons and heavy ions. The flux of these particles increases with altitude. Aircrafts flying at high altitude experience radiation levels that are 100 times more than that at the sea level [Normand 96].

A significant source of ionizing particles at ground level is the secondary radiation induced from the interaction of cosmic ray thermal neutrons and boron. It has been shown that a large part of the upsets on ground are due to neutrons [O'Gorman 94][Tosaka 96].

A comprehensive assessment of terrestrial cosmic radiation is found in [Ziegler 98]. The data show that radiation intensities vary with geographical position, i.e., longitude, latitude and altitude.

2.2.2 Space Environments

Spacecrafts and satellites do not benefit from the shielding provided by the atmosphere. The shielding provided by the magnetosphere depends on the position of the spacecraft relative to the Earth. There are three main sources of radiation in space: Van Allen radiation belts, solar winds, and galactic cosmic rays. Charged particles such as electrons and protons get trapped in Earth's magnetic field. There are regions in this magnetic field where the density of high-energy charged particles is higher. These regions form two belts around the Earth and are called the *Van Allen* belts: the *inner belt* and the *outer belt*. It is mainly the high-energy protons in these belts that can cause upsets in electronic circuits. The trapped electrons do not have sufficient energy to penetrate the spacecraft casing.

Solar winds are composed of mostly protons and some α-particles, heavy ions and electrons. The Sun has a very active surface that can be seen in the form of spots on its surface. From time to time, the Sun ejects plasma from its corona, a phenomenon known as a coronal mass ejection (CME). Some of these CMEs escape the Sun's gravity and form strong solar winds that can hit the Earth. These long bursts of high-energy charged particles can disrupt satellite operations. Solar activity is cyclic and is measured by the number of sunspots. It goes through a maximum (solar max) and a minimum (solar min), on average, every 11 years. The proton flux of the belts is affected by solar activity.

Proton flux in the radiation belts is lower during solar max and higher during solar min. A system may have to be designed for an average flux or worst-case flux depending on the application.

Galactic cosmic rays are mainly very-high-energy heavy ions and have an isotropic (omni-directional) flux. Spacecrafts that are outside the magnetosphere are exposed to these radiations.

Space missions may be divided into three types depending on their trajectories: Earth-orbiting, solar system, and deep space. Satellites are Earth-orbiting spacecrafts and can have a variety of orbits. The main parameters of an orbit are shown in Fig. 2.1. Hp and Ha are the altitude of the perigee and apogee of the orbit, respectively. An orbit can be circular (Ha = Hp) or elliptical (Ha ≠ Hp). Orbits with high inclination are called polar orbits. The main types of satellite orbits are Low-Earth Orbit (LEO), Medium-Earth Orbit (MEO), and Geosynchronous orbit (GEO) (Fig. 2.2).

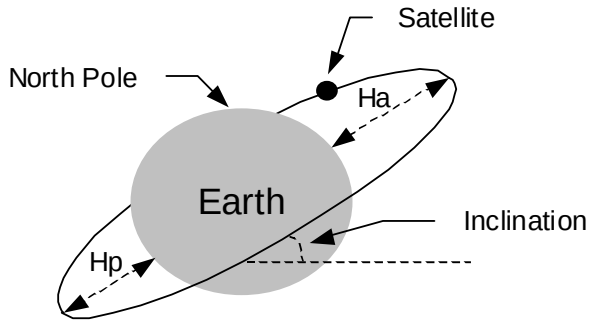

Figure 2.1 Satellite orbit parameters.

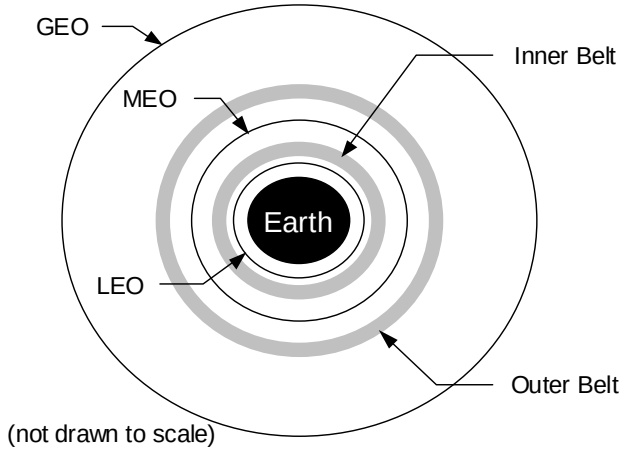

Figure 2.2 Satellite orbits and radiation belts.

LEOs are orbits with about 1,000km altitude. The ARGOS satellite has a LEO orbit with 834km altitude. Its orbit is a special kind of LEO called Sun-synchronous. A

Sun-synchronous orbit is a special polar LEO in which every day, the satellite passes over the same point at the same time of the day.

A LEO orbit is below the inner belt. However, due to nonuniformity of Earth's magnetic field that is partially caused by the displacement of Earth's magnetic dipole relative to the center of the Earth, there is a region with high density of radiation over South America. This region is known as the *South Atlantic Anomaly* (SAA). Due to higher particle flux in SAA, satellite electronics get more upsets in this region than in other parts of the orbit. We observed this effect in the ARGOS project (Sec. 6.1 and 6.2).

The Earth's magnetic lines bend and enter the Earth at North and South poles. The lack of magnetic shielding over these areas creates two regions (the polar cusps) where there is higher density of charged particles. A satellite with a polar orbit is exposed to this radiation.

A MEO orbit has an altitude of about 10,000km and lies between the two belts. However, solar storms affect the positions of the belts and recent studies show that at times a third high-density region may form temporarily. Similar to LEOs, polar cusps are exposed regions for polar MEOs.

Geosynchronous (GEO) orbits have an orbit altitude of exactly 35,790km. These satellites retain their position relative to the Earth and therefore stay over a certain point on the Earth. A geosynchronous orbit is too high to get any radiation shielding from the magnetosphere. Therefore, the electronics of a satellite with GEO orbit is almost fully exposed to cosmic rays and should be designed for a higher level of radiation tolerance than the ones for LEO or MEO orbits.

The conditions of spacecrafts for solar system and deep space missions are, for the most part, unknown. Therefore electronic systems that go into spacecrafts that explore these environments are designed for the worst case.

Another factor that should be counted when designing an electronic system that is going to be used in space is the mission time. The length of the mission, as well as its position relative to the solar activity cycle (considering 1-2 year launch delay) determines the total amount of radiation as well as the peak radiation rate that the electronics should be able to tolerate.

There are numerical models for the ionizing radiation environment in near-Earth orbits and for evaluating radiation effects in spacecrafts based on the collected data from space experiments. Two such models are CREME96 [Tylka 97] and APEXRAD [Gussenhoven 97].

2.3 Radiation-Matter Interactions

There are two basic mechanisms through which radiation interacts with matter: (1) atomic displacement, and (2) electronic charge displacement (also called ionization) [Ohring 98]. In atomic displacement, the energetic particle hits an atom and knocks it out of its original position (this is similar to the ion implantation that occurs during the fabrication process of an integrated circuit). If the atom is part of a crystal structure, the displacement disrupts the structure and can change the properties of the matter. This effect is important in insulator and semiconductor material. As defect concentration increases, the lifetime, mobility and concentration of charge carriers change. The result is a change in the electrical conductivity of the material that can affect circuit operations.

In the second form of interaction, atoms are excited resulting in production of electron-hole pairs. Figure 2.3 shows the electrons and holes generated when an ion (heavy ion or α–particle) hits a CMOS transistor. When the circuit is powered, there is an electric field between the active areas and the substrate. Due to this field, the generated electrons and holes move in the opposite direction, for example, in Fig. 2.3, electrons will move up and into the active area, creating a current pulse[1]. Depending on the magnitude and timing of this current, this may disrupt the proper operation of the circuit and cause a *Single-Event Upset* (SEU). For example, the voltage of the corresponding node may change and represent an incorrect logical value. If the node is in a memory element, it can result in a bit-flip (an undesired change from 0 to 1 or 1 to 0) in the memory. This is an example of a transient failure caused by radiation. It is also known as a *soft error* because no permanent damage is caused in the circuit; the memory element can be restored to its correct value and can operate correctly afterwards.

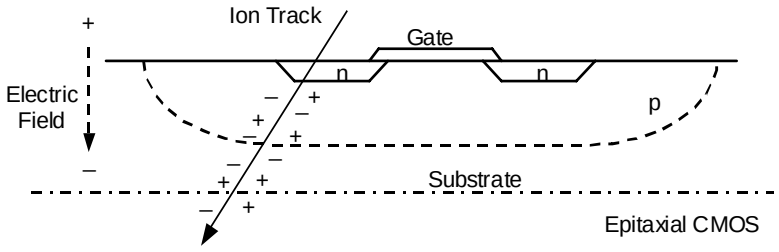

Figure 2.3 A heavy ion hitting a CMOS transistor.

In silicon-on-insulator (SOI) CMOS technologies, a thin substrate is built on an insulating layer. The electron-hole pairs generated in the insulating layer cannot move

[1] The energy deposited in a node by an ionizing particle can be modeled as a voltage (or current) pulse. The duration of this pulse depends on the particle energy and the affected circuit, and can vary from a few hundreds of picoseconds for Si transistors [Wagner 88] to microseconds for certain GaAs transistors [Buchner 91].

due to the electric field and will recombine on the spot. Therefore, the generated current in the active area is relatively smaller and these technologies tend to show less sensitivity to particle hits.

Both energetic particles and photons are capable of causing ionization damage. The excitation of valance-band electrons to the conduction band creates charge carriers that are mobile in electric fields. Simultaneously, slower-moving holes remain behind in the valence band. This interaction produces trapped holes in thin dielectrics, which can degrade the dielectric behavior. Ionization in insulating layers can cause charge build-up, for example, in the gate of an n-p-n CMOS transistor. The excess charge can shift the threshold voltage (V_T) of the transistor, which may result in excess current or functional failure of the transistor. This damage occurs over a long period and is an example of permanent damage or a hard error. This permanent damage is both radiation dose and dose rate dependent.

Heavy ions and α-particles interact with matter through direct ionization. Protons, on the other hand, mainly cause ionization indirectly through nuclear collisions. Neutrons do not have any electric charge and do not cause direct ionization. High-energy neutrons transfer their energy through elastic collision and cause damage by atomic displacement. Low-energy neutrons can have non-elastic collisions with the nucleus of an atom and get absorbed. The excited nucleus loses its energy through an atomic reaction. For example, thermal (low-energy) neutrons can combine with a boron atom and produce an α-particle: $n + B^{10} \rightarrow Li^{7} + \alpha + \gamma$ (the lithium ion can also cause an SEU) [Baumann 95]. This particular combination has become an issue in the CMOS technologies which use boron as a dopant[2].

In the next section, we define some related terminology that will be used throughout this thesis. In Sec. 2.5, a classification of different radiation damages is presented.

2.4 Terminology

In Sec. 2.3, we saw that radiation can cause transient as well as permanent damage to an electronic circuit. In this section we look at the terminology used in quantifying this damage.

A *Single Event Upset* (SEU) is a change of state or a transient error induced in a device by an ionizing particle such as a cosmic ray or proton. Circuit parameters and

[2] Texas Instruments Corp. eliminated the borophosphosilicate glass (BPSG) passivation layer in their process due to this problem [Baumann 00].

particle energy determine whether a particle can cause an SEU or not. The minimum charge that a particle must deposit in a node to cause an SEU is denoted by Q_{crit} and depends on node parameters such as capacitance and voltage. *Linear Energy Transfer* (LET) is a measure of the energy transferred to the device per unit length of the material through which the ionizing particle travels[3]. The minimum LET that can deposit the Q_{crit} and cause an SEU is called *LET threshold* (LET_{th}).

One important metric for radiation damage is the rate at which errors occur in the circuit. For transient errors, this rate is often referred to as *soft error rate* (SER). To calculate the rate at which SEUs occur in the circuit, we start with the rate at which particles hit the circuit. Particle hit rate, or *flux*, is measured in (particles/cm^2)/sec. Depending on the particle energy and the sensitivity of the circuit, a particle may or may not cause an SEU in the circuit. For a fixed particle energy, we designate the sensitivity of the circuit with P_{err} = *Prob* (error | particle hit). This sensitivity is less than one because not all parts of a chip are sensitive to particles (e.g., empty areas or areas with only wires and no transistors underneath). If the chip area is A_{chip}, the total sensitive area of the chip will be $\sigma = P_{err} \times A_{chip}$. σ is known as the *cross section* and is the device upset response to ionizing radiation. For an experiment with a specific LET, σ = # errors / (particle fluence), where particle fluence is particles/cm^2. The units for cross section are cm^2 (number of errors and particles are unitless). A typical cross section versus LET graph is shown in Fig. 2.4. If we know the cross section of the chip, then the error rate will be $\lambda = flux \times \sigma$.

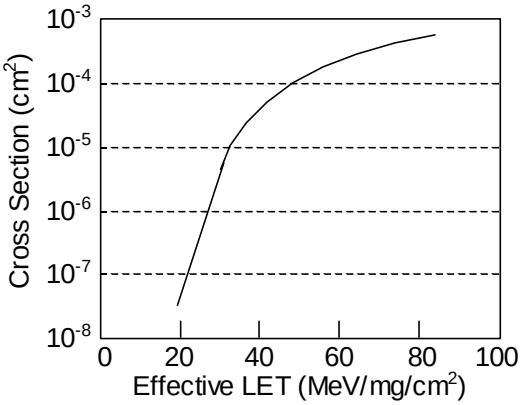

Figure 2.4 A typical cross section versus LET graph.

When a device is exposed to radiation over a long period, it accumulates a total dose of radiation. This radiation *dose* is defined as $D = dE/dm$ (E = energy, m = mass)

[3] The common unit for LET is MeV/mg/cm^2 of material (Si for substrate and SiO_2 for transistor gate in MOS devices); MeV = Million electron-Volts, mg = milligram.

and is measured in *rad* (radiation absorbed dose)[4]. Since the accumulated radiation is material dependent, it is stated as, for example, rads(Si) or rads(SiO_2). In many applications, *dose rate* (dD/dt) is also an important parameter. At higher dose rates, devices may fail at lower total doses because there is less time for charge to anneal [Fleetwood 94].

2.5 Radiation Damage Classification

Radiation damages can be classified into two major categories:

1. Long-term cumulative degradation: These damages have two types:
   - Total Ionizing Dose (TID): the accumulation of ionizing radiation over time, typically measured in rads. The accumulation of charge in different parts of a circuit causes a gradual degradation and eventually failure of the circuit.
   - Displacement Damage Dose (DDD): another cumulative long-term degradation of the circuit due to displacement of atoms in circuit material.
2. Single-Event Effects (SEEs): These are the local damages that are caused by a single incident ionizing particle. The effects can be of many types and are explained next.

SEUs are one type of SEEs. It has been observed that a single particle can affect multiple adjacent memory cells and cause multiple bit-flips [Ziegler 96b][Liu 97][Reed 97][Hosken 97]. These events are called *multiple-bit upsets* (MBUs). If the upset bits happen to be in the same word, it is called a *single-word multiple-bit upset* (SMU).

The *n*-channel and *p*-channel transistors in CMOS structures form a parasitic bipolar transistor that is normally off. However, a current pulse generated by an ionizing particle can trigger this transistor to turn on, creating a low resistance path from power to ground. This effect is called a *Single-Event Latchup* (SEL). SEL is a potentially destructive condition. The resulting large current cannot only drop the supply voltage in the whole circuit, but can also burn out the device due to the excess heat. A current sensor may be able to detect the occurrence of an SEL and shut down the power to prevent permanent damage. SEL is mostly an issue for bulk CMOS devices. SOI technologies are inherently immune to this effect because of the insulating layer that prevents the parasitic transistor from turning on. There are also layout techniques to isolate the transistor structures and avoid the formation of parasitic transistors.

[4] 1 rad = 100 ergs of energy per gram of absorber.

A particle hit can reduce the breakdown voltage in a power MOSFET and cause an effect similar to SEL called *snapback*. The excess current of a snapback is usually not high and the circuit can resume its normal operation after a power cycle.

Another type of SEE is a *Single-Event Burnout* (SEB) where the drain-source channel in a power MOSFET burns out due to excess current. A *Single-Event Gate Rupture* (SEGR) also occurs in power MOSFETs when the gate shorts to the channel due to a particle hit. SEGRs have recently been observed in CMOS structures in logic, memory and FPGAs, too.

The trends in new technologies, namely, smaller feature sizes, lower supply voltages, and smaller threshold voltages, have led to a new type of radiation damage mechanism [Swift 94]. The charge deposited in a CMOS transistor gate due to a single particle hit has become comparable to the amount of charge that is needed on the gate for turning on the transistor. The result is called a *microdose* damage [Oldham 93]. This is local permanent damage that can cause a stuck-at-on fault in a transistor.

Radiation effects are either permanent or transient and can be local or global. Table 2.1 summarizes the radiation effects based on these properties.

Table 2.1 Classification of radiation effects.

	Local	**Global**
Transient	SEU, MBU	SEL
Permanent	SEL, snapback, SEB, SEGR, microdose	TID, DDD, SEL

Radiation can cause degradation of operating characteristics of electro-optical devices, such as laser diodes, LEDs (light emitting diodes), photodetectors and CCDs (charge-coupled devices) [Ohring 98]. In optical fibers, radiation can cause luminescence, which is a spurious light signal (a transient effect), or increase absorption. The latter is a permanent damage that causes attenuation of light pulses.

2.6 Dealing with Radiation Effects

As for other kinds of faults, the techniques for dealing with faults caused by radiation fall into one of the two basic approaches: fault avoidance, or fault tolerance. Even if a device is used in a very low radiation environment, the radiation sources inside the chip can still cause an unacceptable SER. Therefore, the first step in fault avoidance is to use material with a minimum amount of radioactive impurities. Also, elimination of materials that have high probability of interaction with neutron, such as boron 10, significantly reduces the neutron-induced SER [Baumann 00].

Shielding is the oldest way of avoiding external radiation. Enclosing the electronics in metal foils such as lead or aluminum can keep most of the radiation particles from hitting the devices. The biggest drawback of shielding is the weight and volume of the shielding material. This makes shielding very unattractive for space applications, especially with the new space programs that demand smaller and cheaper spacecrafts.

Shielding is effective for α-particles and low-energy protons but not very effective for galactic heavy ions, high-energy trapped protons, gamma rays and X-rays. In some cases, the shield can reduce the particle energy to its most effective energy level. The absorption of the particle by the shield may result in secondary particles that can hit the electronics and cause upsets. Therefore, not all the radiation can be shielded out.

Radiation hardening of ICs is another fault-avoidance technique. Radiation hardness of components can be classified as shown in Table 2.2. In this context, commercial off-the-shelf (COTS) components are components that have no special features to reduce radiation effects[5]. The rad-tolerant category includes commercial components that are inherently hard to a specific level of radiation [Lum 97]. Here, radiation hardness is a by-product of the design and/or technology. This category can be used in low to medium radiation environments. Rad-hard components are specially designed and fabricated components that can withstand relatively harsh radiation environments.

Table 2.2 Classification of components according to their radiation hardness [Hash 97].

	COTS	**Rad-Tolerant**	**Rad-Hard**
Total Dose ***krad (Si)***	< 20	20 to 100	100 to 1000
Transient ***rad (Si)/s***	$< 10^7$	10^7 to 10^9	$> 10^9$
SEL	Customer evaluation and risk	Customer evaluation and risk	Not allowed or low risk
SEE ***MeV-cm^2/mg***	< 20	20 to 80	> 80

Hardening is done by using: (1) a special fabrication process, (2) circuit techniques, (3) layout techniques, or a combination of these. As mentioned in Sec. 2.5, SOI technologies are more immune to SEL. GaAs technologies have no gate insulators to experience charge buildup and are more rad-tolerant compared to Si technologies.

[5] Sometimes COTS is used in contrast with "custom" components that are not commercially available. In this case, for example, a radiation-hardened R6000 is also COTS [Winokur 99].

Addition of a shielding layer in the fabrication process is a simple way of increasing the radiation hardness of ICs. It is documented that a triple well structure can reduce SER in SRAMs [Burnett 93].

Radiation hardness of commercial components is very fabrication process dependent and may vary across the fabrication lots [Shaneyfelt 94]. Variation in SEU sensitivity has also been observed in parts that are from the same lot [Underwood 98]. Therefore, to assure quality, either the manufacturer or the customer may have to evaluate each lot using a sampling method.

Circuit design techniques usually make a trade-off between radiation hardening of the circuit and circuit speed. For example, using larger capacitances makes Q_{crit} larger and consequently, the circuit will be less vulnerable to SEUs. However, larger capacitances take longer time to charge and discharge, reducing the circuit speed. Moving more charge around the circuit also translates to more power.

Addition of a pair of resistors to SRAMs cells is a circuit technique for hardening SRAMs against SEUs [Wang 99]. The resistors reduce the gain of the feedback loop of the back-to-back inverters, reducing the chance that a transient pulse caused by a particle will flip the state of the SRAM bit. The drawback is that these resistors increase the write time of the memory. Some techniques for SEU hardening of circuits can be found in [Kang 86] [Calin 96] [Zhang 98] [Nicolaidis 99].

Hardening is a specialized process that has been used only for military and space applications. Due to the limited market for radiation hardened parts, they are very expensive, not many parts are available in radiation-hardened format, and the parts that are available are usually old designs and at least two generations behind commercial components in terms of performance. Therefore, it is desirable to use commercial components if radiation effects can be tolerated using fault tolerance techniques.

A fault-tolerant system allows for the occurrence of faults in the system but prevents the fault from producing a system error by detecting and recovering from the error. Error detecting and correcting (EDAC) codes are the most commonly used technique for protecting memories. Duplication and triple modular redundancy (TMR) are well-known fault tolerance techniques that can be implemented in time (repeated execution) or in hardware (replicated components). Spare components are always needed if permanent faults are to be tolerated. The advantage of cold spares over hot or warm spares in a radiation environment is that a cold spare accumulates a smaller radiation dose while it is not powered. This is due to the absence of the electric field that separates the electrons and holes generated during a particle hit.

Software-implemented hardware fault tolerance (SIHFT) techniques provide alternatives to costly hardware techniques [Shirvani 00a] [Oh 01a] [Oh 01b]. The cost is performance since part of the processing power is used for redundant calculations. However, a high-performance state-of-the-art commercial system with SIHFT techniques could outperform a customized hardware that is expensive and old technology.

2.7 Summary

There are different types of radiation in space and on the Earth that cause faults in electronics. Some faults, such as SEUs, are transient, and some faults are permanent. Transient faults outnumber permanent faults by at least an order of magnitude. Meanwhile, if the mission time is long enough, permanent faults cannot be ignored.

More information on radiation effects on electronic systems, related research papers and radiation effects data can be found in [IEEE-TNS][Messenger 92][NSREC] [Radhome][RADECS][Radnet][SEECA].

2.8 Relation to the Following Chapters

Microdose, SEB and SEGR are three types of permanent faults that are caused by radiation[6]. We address permanent faults in Chap. 3 and present a new technique for tolerating permanent faults in cache memories.

In chapters 4 to 6, we address transient faults, in particular SEUs. The problem of SEUs in memory is usually solved by adding EDAC in hardware. Chapter 4 discusses how to implement EDAC in software.

An integrated circuit may be composed of different units each with a different sensitivity to SEUs. For example, a processor consists of register file, ALU, floating point units and decoder, to name a few. When adding error detection and protection to the circuit, it is desirable to know which units are more sensitive to SEUs and need protection. Chapter 5 presents a technique for estimating the SEU sensitivity of each unit in a digital circuit.

Results of an actual space experiment are the topic of Chap. 6. The experimental data is from a pair of processor boards on the ARGOS satellite which has a Sun-synchronous orbit. We demonstrate the feasibility of software-implemented EDAC as well as many other SIHFT techniques.

[6] Permanent faults can also be caused by electromigration and hot carriers. However, these failure mechanisms are not the subject of this dissertation.

Chapter 3
PADded Cache

In this chapter, we present a new technique for tolerating permanent faults in cache memories [Shirvani 99]. Current fault-tolerance techniques for caches are limited either by the number of faults that can be tolerated or by the rapid degradation of performance as the number of faults increases. We present a new technique that overcomes these two problems. This technique uses a special *Programmable Address Decoder (PAD)* to disable faulty blocks and to re-map their references to healthy blocks.

Permanent faults occur due to reliability failures such as electromigration and hot carrier effects. They can also be caused by radiation. As mentioned in Chap. 2, microdose, single-event burnout (SEB) and single-event gate rupture (SEGR) are local permanent faults caused by radiation. PADded cache is a technique for handling these local permanent faults. It is not meant for handling global permanent faults such as TID or destructive SELs.

All high performance microprocessors use a hierarchy of cache memories to hide the slow access to the main memory [Hennessy 96]. With each new generation of integrated circuit (IC) technology, feature sizes shrink, creating room for more devices on one chip. We see a fast growth in the amount of cache that designers integrate into a microprocessor to gain higher performance. Hence, caches occupy a substantial area of a microprocessor chip and contain a high percentage of the total number of transistors in the chip. Consequently, the reliability of the cache has a big impact on the overall chip reliability. The new fault-tolerance technique presented in Appendix A addresses this issue by exploiting the architectural features of caches. We propose a reconfiguration technique that can keep the cache operational at a high level of performance (low miss rate) even in the presence of a large number of faults. Our technique provides a reconfiguration method to isolate the faulty part; it assumes the system is equipped with appropriate error detection and error recovery mechanisms such as *error detection and correction* (EDAC) codes.

In Sec. 3.1, we review the previous work on fault-tolerance techniques for memories and caches. In Sec. 3.2, we present the *PADded* caches — caches with *Programmable Address Decoders*. To evaluate the performance of PADded caches, we carried out simulations. The results are explained in Sec. 3.3. We discuss some implementation issues and the advantages of our technique in Sec. 3.4 and conclude in Sec. 3.5. This chapter is a summary of Appendix A.

3.1 Previous Work

Many processor architectures are designed so that the processor can operate without a cache under certain circumstances. Therefore, the obvious solution to a faulty cache is to totally disable it. In set-associative caches, a less extreme solution is to disable one unit (way) of the cache [Ooi 92].

Most of the transistors in a cache are in memory cells. Hence, the probability that a given defect is in a memory cell (a bit in the data or tag field) is higher than the probability of it being in the logic circuitry. If the fault is in a data or tag bit, only the block containing that bit needs to be disabled. For marking a cache block as faulty, an extra bit can be added to the set of flag bits associated with each block [Patterson 83]. If this bit is zero, the block is non-faulty and will do its normal function. If the bit is one, it will indicate that the block is faulty. In case of a direct-mapped cache, if the faulty block is accessed, it will always cause a cache miss. In case of a set-associative cache, the associativity of the set that includes that block is reduced by one. We refer to this extra bit as the *FT-bit* (fault-tolerance bit). Other names used in literature are: *availability bit* [Sohi 89], *purge bit* [Luo 94], and the *second valid bit* [Pour 93].

Disabling faulty blocks is done even when the cache is protected by *single-error correcting, double-error detecting* (SEC-DED) codes [Turgeon 91][1]. For example, *Cache Line Delete* (CLD) is a self-diagnostic feature in high-end IBM™ processor caches that automatically detects and deletes the cache blocks with permanent faults [O'Leary 89].

Replacement techniques, such as extra rows and columns, are also used in caches for yield enhancement [Youngs 96] and for tolerating lifetime failures [Turgeon 91] [Houtt 97]. With replacement techniques, there is no performance loss in caches with faults. However, the number of extra resources limits the number of faults that can be tolerated using these techniques. A replacement scheme called the *Memory Reliability Enhancement Peripheral* (MREP) is presented in [Lucente 90]. The idea is to have a set of spare words each of which can replace any faulty word in the memory. A technique similar to MREP is presented for caches in [Vergos 95]. A very small fully associative spare cache is added to a direct-mapped cache that serves as a spare for the disabled faulty blocks. The difference between this technique and MREP is that there can be more faulty blocks than there are spare blocks. The simulation results in [Vergos 95] show that one or two spare blocks are sufficient to avoid most of the extra misses caused by a few (less than 5%) faulty blocks. However, as the number of faults increases, a few spare

[1] A single error (a transient error) is correctable in a fault-free block, but if it occurs in a faulty block it may create a multiple error case that is not correctable by the SEC-DED code.

blocks is not very effective. Cache memories have an inherent redundancy that can be exploited to increase this limit.

3.2 PADded Caches

In this section, we present a new technique that has much less performance loss in caches as the number of faulty blocks increases. Figure 3.1 shows a simple block diagram of a direct-mapped cache. The mapping of block addresses to the blocks in the cache is done by a decoder which is shown as a gray rectangle in the diagram. We modify this decoder to make it programmable so that it can implement different mapping functions suitable to our technique. We call a cache with such a decoder a PADded cache. No spare blocks are added for tolerating faults. Instead, we exploit the existing non-faulty blocks as substitutes for the faulty ones.

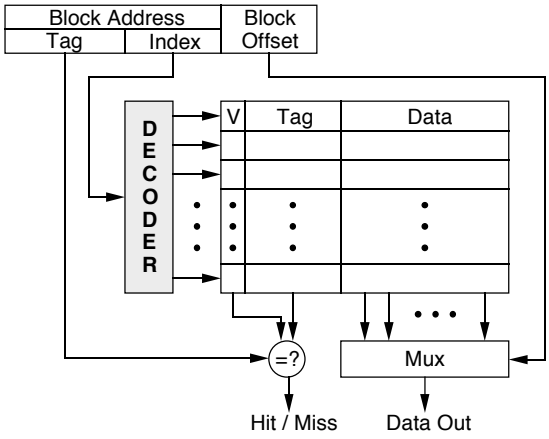

Figure 3.1 Block diagram of a direct-mapped cache.

Figure 3.2 shows part of the last three stages of a simple decoder. This type of decoder may not be used in speed-critical circuits. However, it is easier to illustrate our re-mapping procedure using this decoder. The same procedure applies to decoders that use gates instead of pass transistors as shown in Appendix A, Sec. 5.

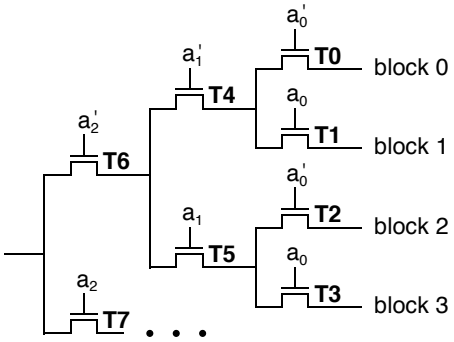

Figure 3.2 Circuit diagram of a simple decoder.

In Fig. 3.2, each *block i* output is a word line that selects a block in the cache. For example, when $a_2a_1a_0$=000, *block 0* is selected, when $a_2a_1a_0$=001, *block 1* is selected, and so forth. Now assume that *block 0* is faulty. We modify the control inputs of transistors *T0* and *T1* such that, when *block 0* is marked faulty, *T0* is always off and *T1* is always on. That is, all the references to *block 0* are re-mapped to *block 1*. Therefore, the addresses that map to *block 0* are still cacheable and will only suffer from conflict misses with *block 1*. Similarly, if *block 1* is faulty, *T0* will be always on, *T1* will be always off, and all the references to *block 1* are re-mapped to *block 0*. Figure 3.3 shows the modified decoder. Since one address bit is lost due to this re-mapping, one bit is augmented to the tag bits to distinguish the addresses that may be mapped to the same block when faults are present.

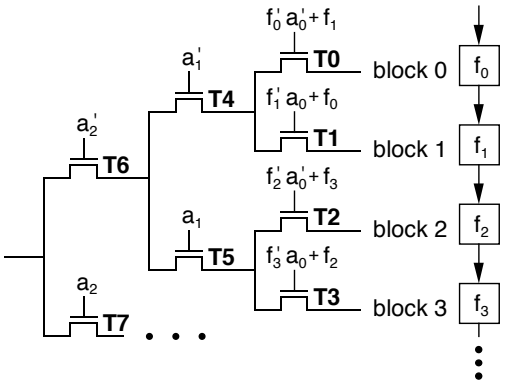

Figure 3.3 A simple Programmable Address Decoder (PAD).

Similar to the FT-bit, an extra bit is added to each block to mark it as faulty or non-faulty (the f_i's in Fig. 3.3). These flip-flops can be connected as a shift register and loaded using special instructions. The control inputs of the pass transistors are modified as shown. For example, the control for *T0* is changed from a_0' to $f_0' \cdot a_0' + f_1$. Therefore, *T0* is always on if *block 1* is faulty (f_0=0, f_1=1) and is always off if *block 0* is faulty (f_0=1, f_1=0).

The scheme can be applied to multiple levels to tolerate multiple faulty blocks and to meet the desired level of fault tolerance. For example, the control of transistor *T4* can be changed to $g_0' \cdot a_1' + g_1$, where $g_0 = f_0 \cdot f_1$ and $g_1 = f_2 \cdot f_3$. In a case where both *block 2* and *block 3* are faulty (g_0=0 and g_1=1), *T4* and *T5* will always be on and off, respectively. Therefore, the references to blocks *2* and *3* will be re-mapped to blocks *0* and *1*, respectively. In this case, the tag portion has two extra bits. The same procedure can be applied to all levels.

This technique can be similarly applied to a set-associative cache. Typically, in a physical design, there is a separate memory array for each way of a set-associative cache.

The decoder can be shared between the arrays, or it may be duplicated due to floor-planning or circuit issues. For example, consider a 4-way set-associative cache with one decoder for each way. Our technique can be independently applied to each decoder, i.e., re-mapping in one array will not affect mapping of the other arrays. Figure 3.4(a) illustrates a case where a set has a faulty block — we call this set the *faulty set*. With a PAD, the faulty block is mapped to a non-faulty block. We call the set that contains this non-faulty block the *congruent set* of the faulty set (in case of a direct-mapped cache, each set has only one block). Because the re-mapping does not affect the other blocks in the set, one healthy block of the congruent set will be shared between the faulty set and the congruent set and the rest of the blocks are used as before. If there are two decoders, one for each pair of arrays, the healthy block that shares a decoder with the faulty block will also be marked faulty and two blocks will be shared between the two sets. This is illustrated in Fig. 3.4(b).

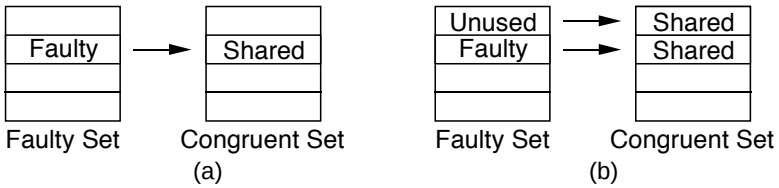

Figure 3.4 Block sharing between a faulty set and its congruent set in a 4-way set-associative cache: (a) with 4 decoders, (b) with 2 decoders.

The conflict misses that occur between a set and its congruent set can be reduced by special ordering of the address bits in the decoder. Caches exploit the spatial locality of the memory accesses. If we make a set and its congruent set far apart in the address space, there will be less conflict due to this spatial locality. Since the order of the inputs to the decoder is not important in the function of the cache, we use the index bits in the reverse order. That is, the most significant bit (msb) of the array index is connected to the least significant input bit (lsb) of the decoder. The simulation results show that this design has better performance than a direct connection (normal order).

3.3 Simulation Results

We evaluated the performance of PADded caches through simulation. We compared the miss rates of caches with two fault-tolerance techniques, simple block deletion using the FT-bit, and our new PAD technique. We refer to the first technique as FTB. The simulated caches are write-through, allocate-on-write-miss, with no prefetching and no sub-blocking, and use the LRU replacement protocol. We assume that the decoders are replicated for each way of the cache, and they are programmable for all levels. The faults are injected at random locations and miss rates are calculated for

different number of faulty blocks. Since the location of the faults affects the miss rate, each simulation was run several times and the minimum, maximum and average miss rates were recorded. We used many sets of traces for our simulations. One of them is the ATUM traces [Agarwal 86]. The miss rate for one set of traces is calculated by taking the weighted average of the miss rates of each trace in the set — weighted by the number of references in each trace. We present some of the results in this section. Details of the simulation setup and further results can be found in Appendix A.

The average miss rate of the ATUM traces is shown in Fig. 3.5 for both the FTB and PAD techniques. This graph shows the results for an 8KB cache with 16-byte blocks and with associativities: direct-mapped (DM), 2-way set-associative (SA2), and 4-way set-associative (SA4). The miss rate of PADded caches stays relatively flat and increases slowly towards the end. Figure 3.6 shows the lower left corner of Fig. 3.5. Notice that for a small percentage of faulty blocks, both techniques perform equally well for the set-associative caches.

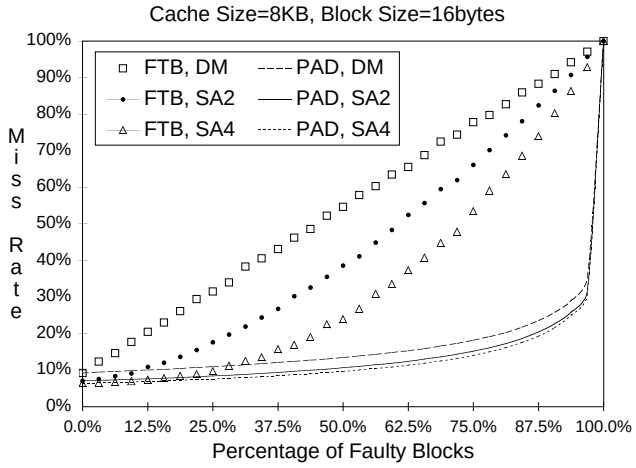

Figure 3.5 Average miss rates of the ATUM traces for different associativities.

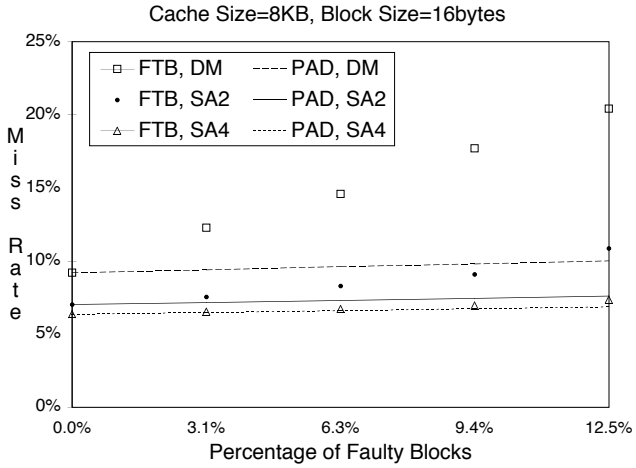

Figure 3.6 Average miss rates of the ATUM traces (lower left corner of Fig. 3.5).

Simulations of caches with different sizes show that when half of the blocks are faulty, the miss rate of a PADded cache is almost the same as a healthy cache of half the size (Appendix A, Sec. 4). This indicates that in PADded caches, the full capacity of healthy blocks is utilized and the performance decreases with almost the minimum possible degradation.

Another advantage of our technique is shown in Fig. 3.7. This figure shows the minimum and maximum miss rates of the ATUM traces for different fault locations, in a direct-mapped cache. The miss rate of FTB has a noticeably big range (as was shown in [Pour 93]) while the miss rate of PAD remains almost unchanged. This shows that PADded caches are very insensitive to the location of faults. Note that we did not do an exhaustive search for the minimum and maximum miss rates; these are the ranges observed in our simulations.

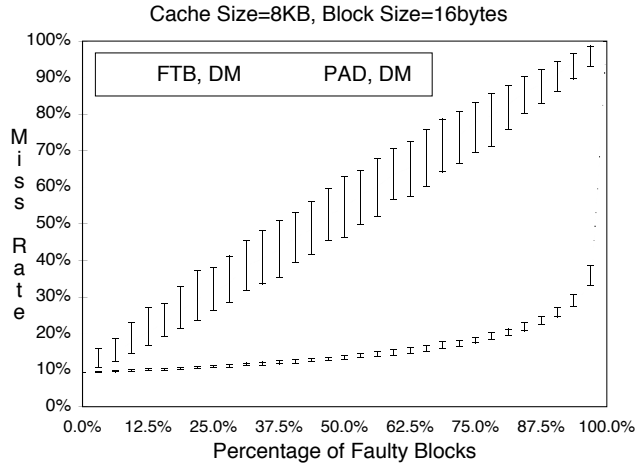

Figure 3.7 Minimum and maximum miss rates of the ATUM traces for a DM cache.

3.4 Discussion

In this section, we discuss some implementation issues and other advantages of our technique over FTB. The area overhead of the PADded cache technique was estimated at about 10% of the overall cache area for a hypothetical design and is expected to be less for actual designs (Appendix A, Sec. 5). The access time overhead is negligible. The advantage of FTB over PAD is low hardware overhead (we don't have overhead numbers for FTB and other techniques to do a better comparison). Adding an extra bit to each cache block will impose a relatively small area overhead. In some caches, this extra physical bit can be avoided by using an unused combination of the available flag bits in cache. For example, in a write-back cache, the combination *dirty*=1 and *valid*=0 can be used to mark a block as faulty.

The FTB technique relies on the availability of a path to bypass the cache when a faulty set is being accessed. Some cache organizations require blocks to be loaded into the cache before they are read by the CPU [Patterson 83]. With FTB, data that resides in an address that is mapped to a set with all faulty blocks does not have a place to go in the cache and has to bypass the cache. The same problem exists for write-back caches where store instructions write to the data cache and the new data is written into the memory when the block is replaced. For a faulty set, the store has to be executed in a write-through fashion. Similar problem exists for caches with allocate-on-write-miss policy. Therefore, hardware should be added to make the references to the faulty sets behave like references to non-cacheable addresses. However, with PAD, the data is always cacheable as long as there is a non-faulty block in the cache, so there is no need for extra hardware.

Our simulation results show that PAD is most beneficial for direct-mapped caches. For set-associative caches, both FTB and PAD perform equally well for small percentages of faulty blocks. Faults were injected at random in our simulations, modeling independence between the blocks. However, the failure mechanisms may be such that the probability of adjacent faults is higher than that depicted by this model. In that case, the performance of set-associative caches will degrade faster and PAD may be chosen over FTB for a relatively lower number of faults.

3.5 Conclusion

PADded cache is a novel fault-tolerance technique that uses graceful degradation to tolerate permanent faults in cache blocks. It can be used for on-line repair in systems for high reliability and availability. The main advantage of our technique is its slow degradation of performance as the number of faults increases. This increases the lifetime of a chip and subsequently, the availability of a system, because there will be less downtime for component replacement.

The PAD technique can also be used for yield enhancement. However, the analysis in [Nikolos 96] shows that maximum yield can be achieved by accepting a few faulty blocks. The FTB technique can provide acceptable performance for a few faulty blocks. Therefore, if yield enhancement is the goal, FTB is probably the better choice because it has lower overhead than PAD. The PAD technique provides a better solution when: the total number of faulty blocks that have to be tolerated (i.e., the manufacturing faults and the faults that occur during the lifetime of the cache) is more than a small percentage of total number of blocks, sustaining the peak performance is very critical, or the performance should be predictable. The last case is justified by the fact that FTB has a big range for the miss rate depending on the location of the faulty block, but PAD has a

very small range. In real-time applications, the system has to execute a specified program within a certain amount of time, i.e., the performance has to be predictable to a specified level. A fault-tolerant cache that can maintain its estimated miss rate for that specified program under different conditions will be the better choice.

Our technique provides a single solution for all types of caches with different policies. No extra hardware has to be added for write-back or allocate-on-write-miss caches. Application of PAD for caches that are used in multi-level cache systems and in multi-processor systems is an area for future research.

Chapter 4
Software-Implemented EDAC

Transient errors and permanent faults in memory chips are well-known reliability issues in computer systems. *Error detection and correction* (EDAC) codes — also called *error-correcting codes* (ECCs) — are the prevailing solution to this problem [Chen 84]. Typically, the memory bus architecture is extended to accommodate extra bits, and encoding and checking circuitry is added to detect and correct memory errors. This additional hardware is sometimes omitted due to its cost. If a computer is to be designed using *commercial-off-the-shelf* (COTS) components that do not have EDAC hardware for memory, the reliability problem has to be addressed with another form of redundancy.

This chapter discusses the implementation of EDAC in software and presents a technique for a system that does not have hardware EDAC but requires protection for code and data that reside in the main memory [Shirvani 00b]. The motivation for this work came from the Stanford ARGOS project (Appendix D). There is a processor board on the ARGOS satellite that does not have memory EDAC. We observed that SEUs corrupt the operating system or the main control program of this board, forcing a system reset. In order to carry out our experiments effectively, these critical programs have to be protected against SEUs.

The goal was to provide protection against transient errors (soft errors) that manifest themselves as bit-flips in memory. These errors can be caused by *single event upsets* (SEUs) discussed in Chap. 2, power fluctuations, or electromagnetic interference. It has been observed that radiation-induced transient errors also occur at ground level [O'Gorman 94] [Ziegler 96a]. Therefore, the technique presented in this chapter can also be useful for terrestrial applications. Handling permanent faults (hard errors) in memory is discussed elsewhere [Chen 84] [Rao 89] and is not the focus of this chapter.

In Sec. 4.1, we review the previous work related to software-implemented EDAC. In Sec. 4.2, we look at the problem in more detail and discuss the requirements of a software-implemented EDAC scheme. We implemented four different example EDAC coding schemes in software. These schemes are compared in Sec. 4.3. Issues that have to be considered for handling multiple errors and solutions to them are discussed in Sec. 4.4. The reliability improvement of an application in a space environment is estimated in Sec. 4.5. We conclude the chapter with a discussion in Sec. 4.6 and a summary in Sec. 4.7. This chapter is a summary of Appendix B.

4.1 Previous Work

Error control coding is a well-developed field [Rao 89] [Wicker 95]. EDAC codes are used to protect digital data against errors that can occur in storage media or transmission channels. The encoding and decoding of data can be done in hardware, software or a combination of both. Since special hardware for a coding system can be expensive, researchers have studied the feasibility of using general-purpose microprocessors for software implementation of EDAC codes [Paschburg 74] [Whelan 77]. Efficient software methods have been devised to do *Cyclic Redundancy Checking* (CRC) using table look-up [Whiting 75] [Sarwate 88]. A comparison of fast implementation of different CRC codes is given in [Feldmeier 95]. CRC codes are used for detecting multiple-bit errors in communication systems where correction can be done by retransmission. In storage systems, a coding scheme with correction capability is used. There are many different codes used in hard disks and tape backup systems. Some of these codes can be used for protecting data residing in memory chips. For example, a software implementation of a (255, 252) Reed-Solomon code that can do single-byte error correction is proposed in [Hodgart 92] for protecting RAM discs of satellite memories. However, there are differences between memory and secondary storage systems that need to be addressed in order to choose an appropriate EDAC scheme for memories.

The contributions of this work are:

- Identifying the issues in implementing EDAC in software.
- Illustrating the options and differences in coding schemes by comparing four example codes that may be considered for EDAC.
- Devising a technique that addresses all the requirements of software EDAC including multiple-bit error correction independent of system-level and chip-level structures.
- Designing a self-repairing and recovery mechanism for the software-implemented EDAC program that provides protection for the program itself and also recovers from hang-ups in this program.
- Analyzing the reliability of a system with software EDAC for main memory.
- Presenting an implementation and demonstrating its effectiveness in an actual experiment.

4.2 General Considerations

This section discusses the requirements for an EDAC scheme that is to be implemented in software as an alternative to hardware-implemented EDAC.

4.2.1 Systematic Codes

Our objective is to devise a scheme to protect the data residing in main memory. The data that are protected by software EDAC are fetched and used by the processor in the same way as unprotected data are fetched and used. The EDAC program should run as a background task and be transparent to other programs running on the processor. Moreover, the protected data bits have to remain in their original form, to make the scheme transparent to the rest of the system. This requires the use of a systematic code such that data bits are not changed and are separable from the EDAC check bits.

4.2.2 Checkpoints and Scrubbing

If the same protection that is provided by hardware is to be provided by software, each read and write operation done by the processor has to be intercepted. However, this interception is infeasible because it imposes a large overhead in program execution time. Therefore, for software-implemented EDAC, we chose to do only *periodic scrubbing* where the contents of memory are read periodically and all the correctable errors are corrected. If memory bit-flip errors are not corrected by the periodic scrubbing before a program is executed, we rely on other software-implemented error detection techniques (e.g., assertions, *Error-Detection by Duplicated Instructions* [Oh 01a], or *Control-Flow Checking by Software Signatures* [Oh 01b]) to detect the errors. When an error is detected, a scrub operation is enforced before the program is restarted.

There are two main types of information stored in a memory: code and data. Code segments contain instructions, and data segments contain the data that is used or produced in computations. Since the write operations are not intercepted to update the check bits, EDAC protection can be provided only for memory blocks that have fixed contents. This includes code segments[1] and read-only data segments. Protection for writeable data segments can be provided by modifying the application programs. *Application Program Interfaces* (APIs) can be defined so that the programmer can make function calls to the EDAC program to request protection for a specific data segment and to modify the contents of the data through the APIs (an example API is given in Appendix B, Sec. 6).

[1] After a program has been loaded and linked by the operating system, the contents of the code segment are not changed (with the exception of self-modifying codes that are not considered here). Therefore, a fixed set of check bits can be calculated for code segments.

4.2.3 Overhead

The space used for check bits reduces the amount of memory available for programs and data. Therefore, the *check-bit overhead* (# check bits / # data bits) must be as low as possible. Codes that have more capability (correction and multiple detection), have higher check-bit overhead and tend to have more complex encoding and decoding algorithms, increasing both performance overhead and program size overhead. A code should be selected that can be implemented by a fast and small program and provides correction for multiple errors. If the program is fast, it imposes low overhead on system performance. More importantly, a fast program is less vulnerable to transient errors that can occur in the processor during execution of the program. Similarly, small program size is important not just because it takes less memory space that could be used for other programs, but more importantly because, it makes the EDAC program less vulnerable to SEUs that may corrupt its own program.

The check-bit overhead of hardware EDAC is the extra memory chips that are added to the memory system to contain the check bits. There is no program size overhead for hardware EDAC but there can be some performance overhead if the latency of EDAC circuitry increases the access time of the memory. With hardware EDAC, the check bits are fetched from memory at the same time the corresponding data bits are accessed. However, with software EDAC, extra memory accesses are needed to fetch the check bits. In addition, there will be some memory accesses for fetching the EDAC program into the processor cache. Therefore, the total memory bandwidth used by software EDAC is more than that of hardware EDAC.

The EDAC program resides in memory and therefore it is vulnerable to errors itself. To correct these errors, a second copy of the EDAC program is executed. Each copy does checking and correction on the other one (cross-checking). An implementation of this self-repairing mechanism is described in Appendix B, Sec. 7.

4.3 Code Selection

4.3.1 Vertical vs. Horizontal Codes

In memory systems with hardware EDAC, the memory width is extended to accommodate the check bits. Figure 4.1(a) shows a diagram for a 32-bit memory word that is augmented with seven check bits. Each set of check bits is calculated based on the bits of one word corresponding to one address. We refer to this type of coding as a *horizontal code*. When a horizontal code is implemented in software, each word is encoded separately and the check bits are concatenated to form a word. This check word is saved in a separate address (Fig. 4.1(b)).

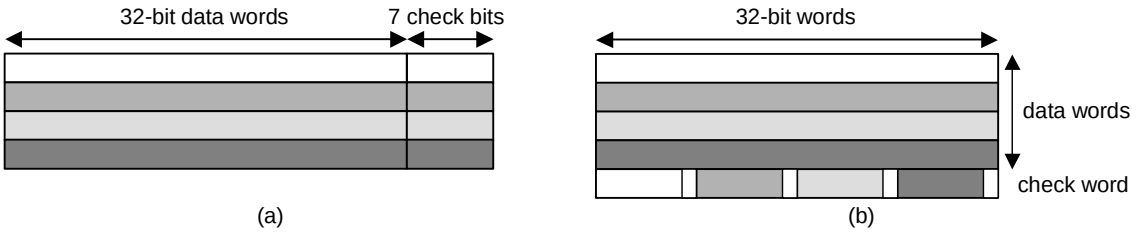

Figure 4.1 A horizontal code over bits of a word: (a) hardware implementation; (b) organization of bits when the code is implemented in software.

Another type of coding is shown in Fig. 4.2. Each set of check bits is calculated over the bits corresponding to one bit-slice of a block of words in consecutive addresses. This type of coding is used in some tape back-up systems [Patel 74] and we refer to it as a *vertical code*. This type of code matches well with the bitwise logical operations that are present in all common instruction set architectures (ISAs). The logical '`xor`' operation is used in the implementation of most of the error detecting codes. Many shifts and logical operations are required for encoding each word in a horizontal code. In contrast, vertical codes lend themselves into very efficient algorithms that can encode all the bit-slices in parallel. Therefore, a vertical code is preferred for a software-implemented EDAC scheme.

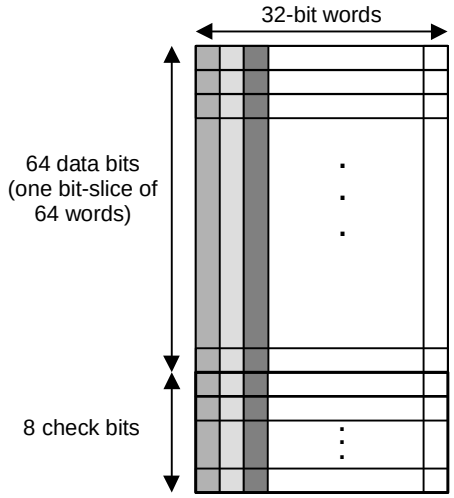

Figure 4.2 A vertical code over bit-slices of words.

Some coding schemes are not quite horizontal or vertical. An advantage of implementing EDAC in software is that it is very flexible and the designer can mix various techniques and codes that would be expensive or infeasible in hardware.

4.3.2 Overhead Comparison of Coding Schemes

We chose four different codes and compared them. These codes were chosen to illustrate the options, the differences, and the facts that need to be considered in choosing

a coding scheme. The designer of a software-implemented EDAC scheme may choose a different code depending on the application.

The four schemes are as follow: (1) a (72, 64) Hamming code implemented as a vertical code over a block of 64 data words with eight check-bit words, (2) a vertical code with the same size as scheme 1, but using a cyclic code instead of a Hamming code, (3) a (1088, 1024) 2-dimentional (vertical and diagonal) parity code similar to a rectangular code (which is vertical and horizontal), and (4) a (66, 64) Reed-Solomon (RS) code in $GF(2^{32})$ with code distance 3. Details of these schemes can be found in Appendix B, Sec. 4.2.

We implemented the four schemes in software and measured their performance on a 200MHz UltraSPARC-I microprocessor. Table 4.1 shows the results. Column 2 shows the size of the code segment of the program that does the encoding and the error detection and correction. Column 3 shows the overhead of the check bits. For Scheme 4, the block size can be larger and the overhead can be reduced as long as the probability of multiple errors in the block remains below the specifications. The decoding (error detection) speed mainly determines the performance overhead of each scheme because decoding is done more often than encoding or correction. The decoding speed of each scheme in terms of megabytes per second is shown in column 4. Column 5 summarizes the error detection and correction capability of each scheme.

Table 4.1 Comparison of program size, check-bit overhead and decoding (error detection) speed of the four coding schemes.

Scheme	Program Size (bytes)	Check-bit Overhead = check-bit/data (words)	Decoding Speed (MB/s)	Detection/Correction Capability
Hamming	14,307	8/64=12.5%	187.80	bit-slice SEC-DED per block
Cyclic	6,731	8/64=12.5%	29.24	bit-slice SEC-DED per block
Parity	6,747	2/32=6.25%	34.68	SEC-DED per block
RS (d=3)	6,723	2/64=3.125%	24.41	SbEC[2] per block

Notice that column 2 shows only the size of the core part of the EDAC program that implements the encoding and decoding of the codewords (including correction). There are other parts of the program that maintain the list of memory segments that are scrubbed, implement the interleaving technique (discussed in Sec. 4.5), communicate with other programs, etc. The size of these parts, which is not included in column 2, depends on the features of the EDAC program and is the same for all the coding schemes. In our implementation, these parts were about 15,000 bytes in size. The differences in

[2] SbEC = Single-byte Error Correction; which means any error (single or multiple bit) within a single word can be corrected.

the core size are small compared to the size of the whole EDAC program. Therefore, when comparing the coding schemes, the core program size is a minor factor.

Scheme 1 has the highest decoding speed but also has the largest program size. Large program size is a minor disadvantage as discussed in the previous paragraph. Scheme 2 has the same check-bit overhead and detection/correction capability as Scheme 1, but has a much lower decoding speed (this speed may be acceptable depending on the application). Schemes 3 and 4 have lower check-bit overhead at the expense of less detection/correction capability.

There are many other EDAC codes and the proper code is chosen depending on application specifications. A scheme that has smaller program size, lower check-bit overhead and higher decoding speed is preferred. The last decision factor is the capability of the codes in handling multiple errors.

4.4 Multiple Error Correction

Multiple errors occur in two ways: (1) multiple SEUs can occur before the memory is scrubbed for errors, or (2) a single SEU causes a *multiple-bit upset* (MBU). In the former case, the scrubbing frequency needs to be adjusted according to the SEU rate to avoid exceeding the correction capability of the utilized EDAC code with a high level of confidence. The latter case has to be approached differently.

It has been observed that a single particle can affect multiple adjacent memory cells and cause multiple bit-flips [O'Gorman 94][Ziegler 96b][Liu 97][Reed 97] [Hosken 97]. MBUs occurred in 1-10% of SEUs in a set of satellite experiments [Underwood 97][Oldfield 98][Shirvani 00a]. The fact that these multiple errors correspond to memory cells that are physically adjacent should be considered when designing an EDAC scheme. If the design is such that the physically adjacent bits belong to separate codewords, these errors can be corrected. To achieve this, the designer of the EDAC scheme needs to know the mapping of physical bits of the memory structure, to the logical bits in memory address space (location of the bits in a programmer's view of the memory). This mapping is determined by the system-level structure and the chip-level structure (detailed analysis can be found in Appendix B, Sec. 5).

A 512K×8 memory chip (Cypress CYM1465) is taken as an example [Cypress 99]. To completely derive the physical to logical mapping of the bits inside this memory chip, we looked at the actual physical implementation of the chip. A small portion of this chip is shown in Fig. 4.3(b). Bit 2 and bit 6 are physically adjacent. The number in each cell corresponds to the logical address of the word that contains that bit. This correspondence is illustrated in Fig. 4.3(a) where the same bits are numbered in a logical

view of the memory. Let us consider bit 2 of address 18. This bit is physically adjacent to bit 2 of addresses 01, 02, 03, 17, 19, 33, 34 and 35 (we refer to this as *type 1 adjacency*)— if the geometries are small enough, we may have to consider adjacency with a larger radius [Hosken 97]. For a more interesting example, consider bit 6 of address 16. This bit is physically adjacent with bit 6 of addresses 00, 01, 17, 32 and 33 (type 1), and with bit 2 of addresses 15, 31 and 47 (we refer to this as *type 2 adjacency*); which is something not quite expected. Adjacencies of type 2 are in different bit-slices and vertical codes can correct MBUs of this type. However, type 1 adjacencies are in the same bit-slice and vertical codes may fail to correct the corresponding MBUs[3].

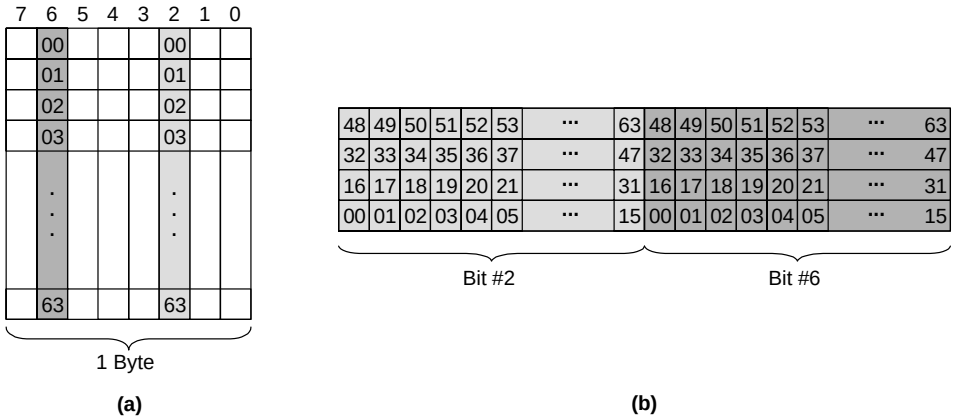

Figure 4.3 Bit positions for a small portion of the memory array in CYM1465: (a) logical positions, (b) physical positions.

To handle type 1 adjacencies with a vertical code, a code with higher correction capability can be used. However, codes with higher correction capability have higher check-bit, performance and program size overhead. Another solution is to logically separate the physically adjacent bits so that each error is in a different codeword. This can be done by *interleaving* the words that belong to the protected blocks. Figure 4.4 shows a 4-way interleaved EDAC scheme that has 64 data words and 8 check-bit words. Starting from address 0, the words of a protected block belong to memory addresses 0, 4, 8, 12, 16,…, 252. Looking at Fig. 4.3(b), we see that having address 0 and 16 in the same block is not desirable. Any *i-way* interleaving scheme, where *i* is of the form $i = 2^k$ (a power of 2), $i = 2^k - 1$ or $i = 2^k + 1$, has the same issue. Therefore, when

[3] A horizontal code can correct the MBUs corresponding to both types of adjacencies. In other words, the internal structure of some memories is such that hardware EDAC works well for all MBUs. However, this is not always true. For example, the internal structure of a ×8 memory chip from Texas Instruments is such that MBUs can occur within individual words [Underwood 92]. Such single-word multiple-bit upsets (SMUs) [Koga 93][Johansson 99] will defeat a SEC-DED horizontal code. Therefore, in this case, a well-designed software EDAC can be more effective than a hardware EDAC.

choosing an interleaving factor, it is best to avoid these numbers. By doing so, the scheme will be independent of the internal structure of the memory chips because for any internal structure, the adjacencies will have a relation that has these three forms (with different *k*'s).

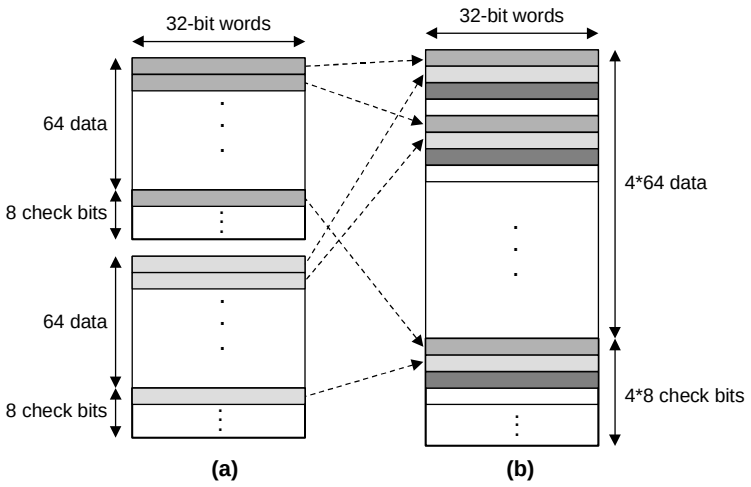

Figure 4.4 Logical mapping of words in a 4-way interleaving technique: (a) blocks of EDAC protected data and the corresponding check-bit words; (b) the location of these words in memory address space.

4.5 Reliability Improvement

Software EDAC was implemented and executed in the ARGOS experiments. It has proved to be very effective in enhancing the availability of the system (details can be found in Appendix B, Sec. 8). Without software EDAC, the system works an average of 2 days before it crashes due to SEU corruptions in programs and needs a reset. With software EDAC this average was increased to about 20 days which is still short but is an order of magnitude improvement.

We also quantified the reliability obtained by software EDAC for programs running in SEU prone environments using analytical methods. The environment assumed for this analysis closely matches the environment for the ARGOS experiment. Reliability of a program with no EDAC, with hardware EDAC, and with software EDAC is shown in Fig. 4.5 for a period of 48 hours. The graph shows that software EDAC improves the reliability of a program that has no EDAC protection by several orders of magnitude[4]. We also quantified the sensitivity of program reliability to scrubbing interval. This analysis shows that for low-radiation environments, the scrubbing interval can be increased (thereby reducing the performance overhead) without appreciably affecting reliability. Details of our analysis can be found in Appendix B, Sec. 9.

[4] The reliability improvement in the ARGOS experiment was lower because we could not provide protection for the data segments of the operating system.

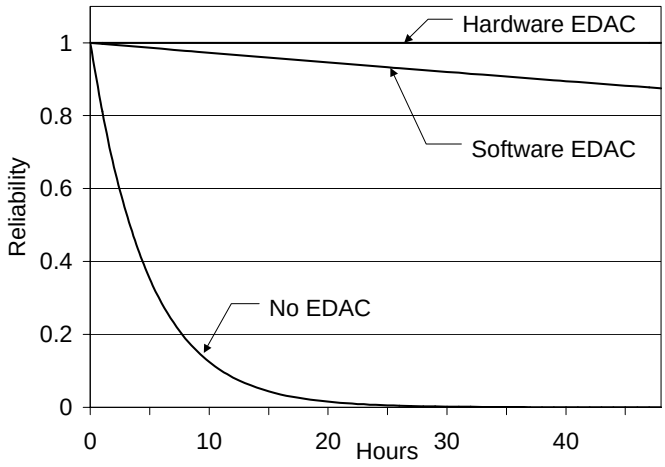

Figure 4.5 Reliability comparison of software EDAC, hardware EDAC and a system with no EDAC.

4.6 Discussion

Solid-state memories, such as RAMs, are used for the main memory, secondary storage, and processor caches in a computer system. Let us consider EDAC protection for main memory. With software EDAC, the data that is read from main memory may be erroneous, if the error occurs after the last scrub operation and before the time of reading. In other words, single-bit errors may cause failures. In contrast, hardware EDAC checks all the data that is read from memory, and corrects single-bit errors. Therefore, hardware EDAC provides better reliability and, when possible, should be the first choice for protecting the main memory. When hardware EDAC is not available or affordable, software EDAC can be used as a low-cost solution for enhancing the system reliability.

For cases where data is read and written in blocks of words rather than individual words, software EDAC may be a better choice than hardware EDAC. For example, when solid-state memories are used for secondary storage (such as in satellites), the processor can access the data through a buffer in main memory rather than directly from the secondary storage. For this secondary storage memory, EDAC protection can be provided in software by periodic scrubbing and APIs. All read operations are checked for errors, and single-bit errors in this memory will not cause failures. Therefore, assuming that the execution of the EDAC program is error-free, software EDAC can provide the same reliability as hardware EDAC if the same coding scheme is used. Considering the flexibility of software EDAC, it is possible to implement more capable coding schemes that are infeasible in hardware, and thereby provide better reliability through software. Moreover, as mentioned in Sec. 4.4, there are cases of MBUs where

hardware EDAC fails but software EDAC can correct the errors. Therefore, software EDAC could be a better choice for such applications.

4.7 Summary

In many computer systems, the contents of memory are protected by an EDAC code. Bit-flips caused by SEUs are a well-known problem in memory chips and EDAC codes have been an effective solution to this problem. These codes are usually implemented in hardware using extra memory bits and encoding-decoding circuitry. In systems where EDAC hardware is not available, the reliability of the system can be improved by providing protection through software. Codes and techniques that can be used for software implementation of EDAC are discussed and compared.

We looked at the implementation requirements and issues, and presented some solutions. We discussed how system-level and chip-level structures relate to multiple error correction. A solution was presented to make the EDAC scheme independent of these structures.

The technique presented in this section was implemented and used effectively in an actual space experiment. We have observed that SEUs corrupt the operating system or programs of a computer system that does not have any EDAC for memory, forcing us to frequently reset the system. Protecting the entire memory (code and data) may not be practical in software. However, we have demonstrated that software-implemented EDAC is a low-cost solution that can provide protection for code segments and can significantly enhance the availability of a system in a low-radiation space environment. This reliability improvement is demonstrated through both a satellite experiment and analytical estimates which are based on parameter values that closely match the environment of the satellite experiment.

For applications where read and write operations are done in blocks of words, such as secondary storage systems made of solid-state memories (RAM discs), software-implemented EDAC could be a better choice than hardware EDAC, because it can be used with a simple memory system and it provides the flexibility of implementing more complex coding schemes.

Chapter 5
SEU Characterization of Digital Circuits

This chapter describes techniques for characterizing the effects of SEUs during a radiation test that is conducted on ground or in space. We propose a technique that can be used for any digital circuit but describe it in the context of processor chips.

The error rate in an entire processor chip can be determined by building a system with two processors and irradiating one of them while both processors execute a program that extensively exercises the internal logic of the processors and generates outputs for comparison. A much more difficult task is to determine the relative rate at which various areas of logic within the chip are upset and the specific logical effects that occur. For a microprocessor, different areas of logic are: register file, ALU, multiply and divide unit and floating-point unit, on-chip caches, etc.

Estimating the SEU rate in a digital circuit is important in designing a fault-tolerant system and systems that are exposed to radiation. If the error rate in each functional unit is known, efforts can be focused on the units that are more vulnerable to SEUs. Once fault tolerance has been added to reduce the effects of SEUs in those units to an acceptable level, we can focus on less sensitive and critical units and continue this procedure until the overall system reliability requirements are met.

There is a second motive for estimating the error rate of individual functional units. The SEU sensitivity of a processor depends on the application program that is being executed on the processor [Elder 88][Kimbrough 94][Koga 85][Shaeffer 92] [Thomlinson 87][Velazco 92]. This is due to the different utilization of the processor functional units by different programs. If the error rate in each functional unit is known, one can predict the error rate of different application programs without having to execute them during a radiation test.

We continue this chapter by reviewing the previous work and listing our contributions in Sec. 5.1. Section 5.2 presents our method and how the test programs are written. We use the MIPS R3000 architecture for illustration. The technique can be similarly applied to other processors and digital circuits. In Sec. 5.3, we show how errors in bistables can be distinguished from errors in combinational logic by operating a sequential circuit at different clock frequencies. Sources of measurement errors, solutions for reducing these errors, and limitations of our technique are discussed in Sec. 5.4. To test the proposed technique, we conducted fault injection simulations. The simulation setup and results are presented in Sec. 5.5. Section 5.6 concludes the chapter. This chapter is a summary of Appendix C.

5.1 Background and Previous Work

As explained in Chap. 2, the error rate of a circuit, λ, depends on two things: (1) particle hit rate and, (2) SEU cross section (σ). Particle hit rate, or *flux*, is measured in (particles/cm^2)/sec. *Cross section* is a measure for the sensitivity of the circuit to ionizing radiation and determines what fraction of particles that hit the circuit actually cause an SEU, that is, σ = # errors / (particle fluence), where particle fluence is particles/cm^2. If we know the cross section of the chip, then the error rate will be $\lambda = flux \times \sigma$.

Since the observability of general-purpose registers is higher than other parts, register file testing is the most popular test in SEU characterization. Some studies used the cross section of latches to infer the cross section of the whole chip [Asenek 98][Brown 97][Sexton 90]. However, the experiment in [Velazco 92] shows that a cross section derived from testing only the registers is not representative of cross section for an application program and is insufficient to characterize the circuit behavior.

Error classification is another issue in SEU characterization of digital circuits. Errors collected during a fault injection experiment are classified based on their behaviors or properties. For example, errors are classified as bit errors, word errors and complex errors in [Elder 88] and as data errors, sequencing errors, address errors and other errors in [Velazco 92]. Then each type may be attributed to one or more functional units. For example, catastrophic errors such as unexpected jumps are attributed to the program counter and instruction decoder. In [Cusick 85], where the data, address and control pins of a Z80 microprocessor are checked for errors, single-bit errors in data and address pins are associated with bit-flips in general purpose registers, and single-bit errors in control pins are associated with internal latches. These associations are clearly crude and not accurate.

Typically, in order to characterize a processor chip, a combination of test programs are used in which each program targets a specific unit and the combined set exercises the whole chip. For example, [Kouba 97] uses a set of ALU, I/O and FPU intensive programs in order to exercise the whole chip. This method could yield an average and a range for the cross section for the whole chip. However, since many of the units of a processor are always used, it cannot give a separate cross section for each unit. Another approach is to use the application that will eventually execute on the system. However, in general, the final application may not be available for radiation tests. Moreover, it is desirable to be able to predict the cross section of a processor for different applications without having to run them in radiation experiments.

In this chapter, we propose a method that attempts to go beyond these limits and enhance the SEU characterization process of digital circuits. The contributions of this work are:

- A new software method for SEU characterization of digital circuits using weighted test programs, system of linear equations and multiple regression, which yields more accurate cross sections of individual units than previous methods and does not depend on ambiguous error classifications.
- Analysis of weighted test programs and the issues and caveats in writing them.
- A new technique for separating the cross section of bistables and combinational logic in SEU characterization of sequential circuits using clock frequency dependency of SEU cross section.

5.2 The Weighted Test Programs Method

The fact that the cross section of a processor depends on the program executed on the system is shown in many studies [Elder 88][Kimbrough 94][Koga 85][Shaeffer 92][Thomlinson 87][Velazco 92]. SEU vulnerability depends not only on processor technology and physical element cross section, but also on the duty factors (utilizations) imposed by the instruction stream [Elder 88][Koga 85][Thomlinson 87]. The *duty factor* of each unit is the percentage of time the unit is active. For a storage element, this is the percentage of time that the element holds a live value (a value that is going to be used sometime during program execution). Multiplying the average cross section per bit by the total number of bits in the device gives an overestimate of the device cross section because all bits are not used at all times. A better approach is to estimate the number of "live and relevant" registers (registers whose SEU will cause observable errors during the relevant program execution) and use that as the number of bits [Koga 85]. A software tool is presented in [Asenek 98] for estimating the duty factors of the registers in a program. In [Elder 88], it is suggested that by knowing the cross section of each unit, σ_i, and their associated duty factors, f_i, the total SEU cross section, σ_T, of the processor executing that specific program can be predicted by:

$$\sigma_T = \sum_i \sigma_i f_i$$

This predication can be done if the σ_i's and f_i's are known. The tool presented in [Asenek 98] can be extended to estimate the f_i's of different units. In this section, we propose a method that uses this basic formula for estimating the σ_i's (cross section of each unit).

Consider a test program that calculates a sum over the contents of 10 general-purpose registers (Fig. 5.1(a)) and assume we want to estimate the cross sections of the register file and the ALU. We can break the cross section of the processor executing this program into three parts:

$$\sigma_T = \sigma_{reg} f_{reg} + \sigma_{ALU} f_{ALU} + \sigma_{remainder} f_{remainder} \quad (1),$$

where σ_{reg} and σ_{ALU} are the cross sections of the register file and the ALU, respectively, and $\sigma_{remainder}$ lumps the cross section of the remainder of the active units of the processor (fetch unit, decode unit, etc.). f_{reg}, f_{ALU} and $f_{remainder}$ are the duty factors of the corresponding units. 10 registers are live during almost the entire execution. If the processor has 32 general purpose registers, then f_{reg} is 10/32 for this program.

```
init r1, .., r10;
sum = 0;
outer loop {
  loop1 n times {
    add  sum,sum,r1
    add  sum,sum,r2
    ...
    add  sum,sum,r10
    add  sum,sum,r1
    add  sum,sum,r2
    ...
    add  sum,sum,r10
  }
  loop2 m times {
    add  r0,sum,r1
    add  r0,sum,r2
    ...
    add  r0,sum,r10
    add  r0,sum,r1
    add  r0,sum,r2
    ...
    add  r0,sum,r10
  }
}
check sum;
```

(a)

```
init r1, .., r20;
sum = 0;
outer loop {
  loop1 n times {
    add  sum,sum,r1
    add  sum,sum,r2
    ...
    add  sum,sum,r10
    add  sum,sum,r11
    add  sum,sum,r12
    ...
    add  sum,sum,r20
  }
  loop2 m times {
    add  r0,sum,r1
    add  r0,sum,r2
    ...
    add  r0,sum,r20
    add  r0,sum,r11
    add  r0,sum,r12
    ...
    add  r0,sum,r20
  }
}
check sum;
```

(b)

Figure 5.1 Pseudocode of a test program for ALU and register file: (a) using 10 registers, (b) using 20 registers.

During the execution of *loop2* in Fig. 5.1(a), the destination register for the `add` instructions is *r0* which is a constant 0 in MIPS architecture. In other words, we are ignoring the results that are produced by the ALU in this loop. The ALU output is used in every cycle in *loop1* but is ignored in *loop2*. Therefore, f_{ALU} is adjustable and is equal to $n/(n+m)$, where n and m are the loop counters in Fig. 5.1. Note that the ALU is also used for incrementing and checking the loop counter variable, and some registers are allocated for the loop counter and sum value. One can count these activities in f_{ALU} and

f_{reg} or in $f_{remainder}$. For this example test, the cross section of the remainder of the processor (other than ALU and register file), $\sigma_{remainder}$, is not the target and $f_{remainder}$ is unknown in these programs. However, we write the test programs such that $f_{remainder}$ remains the same among the test programs and can be replaced by a constant number in equation (1). We assume, without any loss of generality, that $f_{remainder}$ is 1, knowing that the derived $\sigma_{remainder}$ will be a fraction of the actual $\sigma_{remainder}$.

Next, consider the test program in Fig. 5.1(b) that uses 20 registers instead of 10. For this program, f_{reg} is 20/32. We can modify the *m* and *n* parameters in one of these test programs such that it has a different f_{ALU}. This will give us a third program for the third equation. Once we run these three programs while the processor is irradiated, and measure the error rate for each of them, we get three equations with three unknowns, namely σ_{reg}, σ_{ALU} and $\sigma_{remainder}$. Solving these equations will give us the three unknown cross sections.

The basis of our proposed method is as follows. We write a set of programs that exercise a subset of the functional units of the chip, each with known and different duty factors. We call these *weighted test programs*. These programs are run while the system is irradiated and the total cross sections are measured by counting the total number of errors that occur in the execution and using the formula $\lambda = flux \times \sigma$. Finally, the resulting system of linear equations is solved to get the individual cross sections. We call this the weighted test program (WTP) method.

For example, if we break the cross sections into 3 parts (*x*, *y* and *z*) and write 3 programs that exercise these parts with different duty factors (a_i's, b_i's and c_i's), we get:

$$\sigma_1 = a_1 x + b_1 y + c_1 z$$
$$\sigma_2 = a_2 x + b_2 y + c_2 z$$
$$\sigma_3 = a_3 x + b_3 y + c_3 z$$

This can be solved for the unknowns assuming that there are no linear dependency between the equations. In Appendix C, we discuss how the estimates can be enhanced by having more equations than unknowns.

The structure used in the test programs in Fig. 5.1 can be expanded to a generic structure for testing multiple functional units. This structure is shown in Fig. 5.2. Each inner loop targets one functional unit (FU) and maximizes the duty factor of that unit. However, this does not necessarily mean that a unit that is exercised in one loop should not be active in any other loop (examples are given in Appendix C). The cross section equation for this generic format is:

$$\sigma_T = \sigma_{reg} f_{reg} + \sigma_{FU1} f_{FU1} + \sigma_{FU2} f_{FU2} + ... + \sigma_{FUk} f_{FUk} + \sigma_{remainder} f_{remainder}.$$

```
initialize x registers;
R1 = R2 = .. = Rk = 0;
outer loop {
  loop1 n1 times {  // test FU1
    ...
    ...
    ...
  }  // accumulate result in R1
  loop2 n2 times {  // test FU2
    ...
    ...
    ...
  }  // accumulate result in R2

  ...

  loopk nk times {  // test FUk
    ...
    ...
    ...
  }  // accumulate result in Rk
}
check R1, R2, .., Rk;
```

Figure 5.2 Pseudocode of a generic test program for SEU characterization using the WTP method.

Duty factors take real values between 0, for no activity, and 1, for active all the time ($0 \leq f_i \leq 1$). Controlling the duty factors in a test program can only be done with careful programming and, in many cases, is not trivial. For example, the two test programs in Fig. 5.1 are designed to have different register file and ALU duty factors (f_{reg} and f_{ALU}) but the same $f_{remainder}$. Two things have been considered in writing these programs to keep $f_{remainder}$ the same across the test programs. First, the bodies of the inner loops in Fig. 5.1(a) are repeated twice so that it matches with that in Fig. 5.1(b) in terms of number of instructions and execution clock cycles. This will make the branch frequency and program sizes the same.

Second, the structure of *loop1* is used for *loop2* with minimum changes so that f_{reg} and $f_{remainder}$ are almost the same between these loops. In *loop2*, a `nop` operation could be used instead of the `add` instructions. However, with `nop`, a different instruction is decoded and the register file is not accessed. Hence, an `add` instruction that reads the register file is a better choice.

We can create test programs with desired f_{ALU} by changing the *n* and *m* parameters. f_{reg} can also be changed by using the same program structure but using a different number of registers. The same structure can be used for testing other computational units such as the multiplier or the divider unit. Instruction execution latencies and stall cycles should be considered in estimating the duty factors when testing these units (more detailed discussion can be found in Appendix C, Sec. 3.4).

One of the issues with software methods for SEU characterization is their limited cross section granularity. Perhaps $\sigma_{remainder}$ is the best example because it lumps many different smaller cross sections into one. Some cross sections that are lumped into one cross section in one test program may be separated in another test program. However, many of these smaller cross sections correspond to units that are involved in the execution of any program. Therefore, it may be hard or even impossible to separately estimate these cross sections using software.

Consider the ALU tests in Fig. 5.1. Let us assume a pipelined processor architecture. In such architecture, there are latches between pipeline stages (Fig. 5.3). In particular, there are two latches at the input of the ALU to hold the operand values. An error in one of these latches can have the same effect as an error in the combinational circuit of the ALU (the output latch can also be considered if the output is not bypassed). In fact, the cross section σ_{ALU}, used in the equations of the ALU test in Fig. 5.1, lumps the ALU cross section and the cross section of the relevant latches. A more precise equation that separates the two cross sections is:

$$\sigma_T = \sigma_{reg} f_{reg} + \sigma_{ALU} f_{ALU} + \sigma_{latches} f_{latches} + \sigma_{remainder} f_{remainder}.$$

We would like to separate these two cross sections to see how much each contributes to the overall cross section. The result can determine if there is a need to add error detection for each part, e.g., parity for the registers and residue code ([Avizienis 71]) for the ALU.

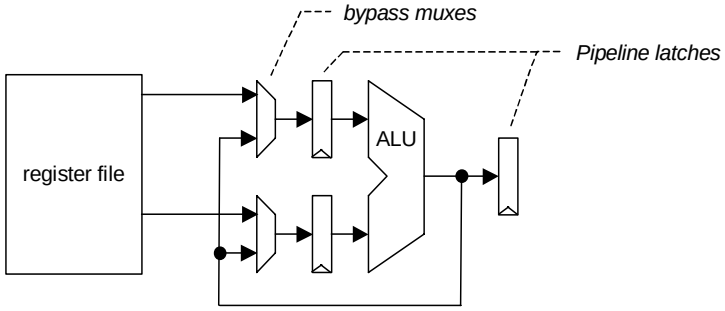

Figure 5.3 Block diagram of a simple pipelined architecture with operand bypassing.

To estimate the ALU cross section, we have to go through the latches to get values in and out of the ALU (in some architectures it may be possible to test the latches separately, e.g., see [Koga 85]). Consequently, the duty factors of ALU and latches (f_{ALU} and $f_{latches}$) are linearly dependent, that is, they increase or decrease by the same amount. Due to this linear dependency, the system of linear equations will not yield individual values for σ_{ALU} and $\sigma_{latches}$, just a total of the two. In general, other techniques have to be used when this type of linear dependency exists between functional units in a circuit. In the next section, we propose one such technique that may be used for the above example.

5.3 SEUs in Sequential and Combinational Circuits

A sequential circuit is composed of memory elements and combinational logic. In this chapter, we use the term "*sequential logic*" to refer to only the memory elements in the circuit, such as flip-flops, latches, registers and embedded RAM, and not the combinational logic parts. SEUs in sequential logic manifest themselves as bit-flips. A *bit-flip* is an undesired change in the state of a storage element – from 0 to 1 or from 1 to 0. Some previous studies show that for transients (glitches in voltage caused by an SEU) that are much shorter than clock cycle, the cross section of sequential logic is independent of clock frequency and remains constant [Buchner 97][Shoga 94]. Other studies have found different behaviors, from nonlinear ([Reed 96]) to linear ([Gardic 96][Reed 96]) dependencies.

SEUs in combinational circuits cause transients at their outputs. The energy deposited in a node by an ionizing particle can be modeled as a voltage (or current) pulse. If this pulse appears at the input of a following flip-flop (or latch) close to the latching edge of the clock, the transient may cause an error in system behavior. This window of vulnerability (Fig. 5.4) is referred to as the *latching window*[1] ([Cha 93]) and is the sum of the flip-flop (or latch) setup and hold times. If the transient pulse arrives at the flip-flop outside this window, the pulse will have no effect and will not cause any error. If the clock frequency is increased, the outputs of combinational logic are sampled at a higher rate. Therefore, the probability of latching the transient pulse increases. In fact, many studies have shown a linear dependency between cross section of combinational circuits and clock frequency [Buchner 96][Buchner 97][Liden 94][Reed 96][Schneiderwind 92].

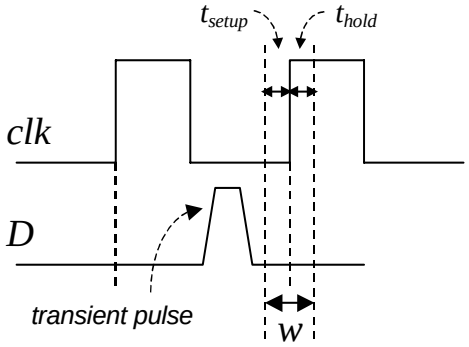

Figure 5.4 Latching window (*w*) of a transient pulse on the data input of a *D* flip-flop.

Let us address the problem of separating the cross section of an ALU (combinational logic) and pipeline latches (sequential logic) described at the end of Sec. 5.2. As explained in the previous paragraph, the cross section of combinational circuits

[1] Also known as the *metastability window*, or the *decision window* [Wakerly 00].

(σ_{comb}) increases linearly with clock frequency. Cross sections of sequential circuits (σ_{seq}) have shown different behaviors with clock frequency. If we know how σ_{seq} changes for the sequential elements used in the circuit under test, we can use this property to break the linear dependencies in our equations by running the test programs at different clock frequencies (assuming that the circuit can work properly at those clock frequencies). For example, let us assume that σ_{seq} does not change with clock frequency. We can operate the processor at its nominal frequency (ω_1) and half its nominal frequency ($\omega_2 = ½\,\omega_1$). If we know that $\sigma_{ALU@\omega 2} = ½\ \sigma_{ALU@\omega 1}$ and $\sigma_{latches@\omega 2} = \sigma_{latches@\omega 1}$, the resulting equations will be:

$$\sigma_{T@\omega 1} = \sigma_{reg@\omega 1} f_{reg} + \sigma_{ALU@\omega 1} f_{ALU} + \sigma_{latches@\omega 1} f_{latches} + \sigma_{remainder@\omega 1} f_{remainder}$$
$$\sigma_{T@\omega 2} = \sigma_{reg@\omega 1} f_{reg} + 1/2\,\sigma_{ALU@\omega 1} f_{ALU} + \sigma_{latches@\omega 1} f_{latches} + \sigma_{remainder@\omega 2} f_{remainder}$$

which can be solved to get separate σ_{ALU} and $\sigma_{latches}$. Note that $\sigma_{remainder}$ may include some combinational logic and change with frequency. Therefore, we use $\sigma_{remainder@\omega 1}$ and $\sigma_{remainder@\omega 2}$ in the equations and estimate them separately.

In summary, clock frequency dependency can be used just like duty factors for creating linearly independent equations. Note that this method will not work if σ_{comb} and σ_{seq} have exactly the same frequency dependency.

5.4 Measurement Errors

As explained in Sec. 5.2, we write a system of linear equations based on the measured total cross sections and the duty factors designed in the weighted test programs. There are several sources of error in these equations that lead to inaccuracy in the solutions (the estimated σ_i's). One source of error in the equations is the uncertainty in the duty factors associated with each functional unit. We saw this in the example of Sec. 5.2 where we tried to match the duty factors in two test programs. The second source of error in the equations is the fact that error detection coverage in test programs is not 100 percent, which means, some errors may not be detected during test and the measured σ_T may have some inaccuracy (a detailed discussion can be found in Appendix C, Sec. 5).

The key points in using the weighted test programs method effectively are the following. First, the duty factors need to be controlled independently and as accurately as possible. Second, the error detection coverage should be as high as possible. Third, all the external effects should be considered and either isolated or counted in the experiments. Fourth, in the weighted test programs method there is no need to classify the errors and associate them with different functional units when there is ambiguity. The separation will be done by solving the system of linear equations. However, if there

is only one possible source for an error, we can clearly use this fact for estimating the cross section of the corresponding functional unit.

The enumerated measurement errors cause inaccuracy in the estimated cross sections. We cannot eliminate all the measurement errors. However, the estimation errors can be reduced by having more equations than the number of unknowns in the linear equations. The statistical analysis of the derived linear equations is discussed in Appendix C. This method was used in the fault injection simulations that are explained in the next section.

5.5 Fault Injection Simulations

To test the proposed technique, we conducted a series of fault injection simulations using a Verilog model of a simple pipelined MIPS processor (the MIPS-Lite used in the Stanford computer architecture course). Three sites were chosen for fault injection: (1) the output of the ALU (before the bypass path forks off as shown in Fig. 5.5), (2) the register file, and (3) the output of the decoder that designates the destination register of the instructions. In each fault injection simulation, a single bit error is injected.

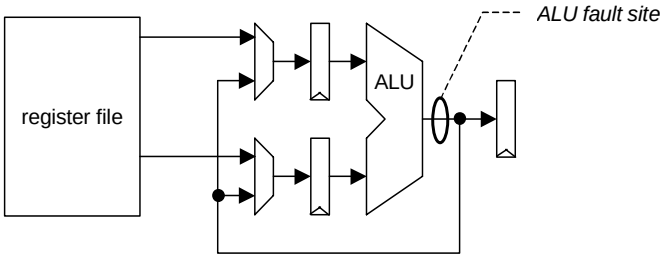

Figure 5.5 Block diagram showing the fault injection site for the ALU.

The programs in Fig. 5.1 were used as the test programs. We used 2 values for f_{reg} = 14/32 and 20/32 and 3 values for f_{ALU} = 1, 2/3 and 1/3. That means 6 different programs (assuming $f_{remainder}$ is 1 and constant), which gives us 6 equations with 3 variables: σ_{reg}, σ_{ALU} and $\sigma_{remainder}$. Notice that these tests are written for estimating σ_{reg} and σ_{ALU}, not $\sigma_{remainder}$.

Each test program takes about 6000 cycles to execute. In each fault injection simulation, a random cycle was selected as the time that the transient fault occurs (single fault assumption) and one of the three fault injection sites was selected randomly based on a chosen distribution. This fault distribution simulates possible numbers for σ_{reg}, σ_{ALU} and $\sigma_{remainder}$ that could happen in a real chip. Each test program was simulated 10,000 times (a total of 10,000 faults for each program). Further details of the simulations can be found in Appendix C.

Table 5.1 Fault injection simulation results.

Fault Distribution (reg. file, ALU, remainder)	Total Num of Faults	Errors counted for test program with f_{reg}, f_{ALU}					
		14/32, 1	14/32, 2/3	14/32, 1/3	20/32, 1	20/32, 2/3	20/32, 1/3
(30% , 50% , 20%)	10,000	**7,788**	**6,260**	**4,623**	**8,463**	**6,850**	**5,236**
(90% , 5% , 5%)	10,000	**4,751**	**4,494**	**4,302**	**6,508**	**6,155**	**5,918**

Table 5.1 shows the simulation results for two different fault distributions. The numbers in bold type are the total number of errors counted for each test program. Using these numbers, the linear equations from the first fault distribution are:

$$
\begin{aligned}
7788 &= 14/23\,\sigma_{reg} + \sigma_{ALU} + \sigma_{remainder} \\
6260 &= 14/23\,\sigma_{reg} + 2/3\,\sigma_{ALU} + \sigma_{remainder} \\
4623 &= 14/23\,\sigma_{reg} + 1/3\,\sigma_{ALU} + \sigma_{remainder} \\
8463 &= 20/23\,\sigma_{reg} + \sigma_{ALU} + \sigma_{remainder} \\
6850 &= 20/23\,\sigma_{reg} + 2/3\,\sigma_{ALU} + \sigma_{remainder} \\
5236 &= 20/23\,\sigma_{reg} + 1/3\,\sigma_{ALU} + \sigma_{remainder}
\end{aligned}
$$

Solving these equations using the multiple linear regression method explained in Appendix C will yield: σ_{reg} = 3,339, σ_{ALU} = 4,794 and $\sigma_{remainder}$ = 1,567. We can also calculate the confidence intervals for each case. Table 5.2 summaries the results of multiple linear regression calculations. Column 2 and 3 are the simulated cross sections (by fault injection) and estimated cross sections (solution of the equations), respectively. The last column is the 95% confidence interval for the estimations.

Table 5.2 Simulated cross sections and the corresponding estimations from multiple linear regression analysis.

Fault Distribution	Functional Unit	Cross Sections (σ)		95% Confidence Interval
		Simulated	Estimated	
(30%, 50%, 20%)	Reg. File	3,113	3,339	[2,905 3,773]
	ALU	4,846	4,794	[4,645 4,944]
	Remainder	2,041	1,567	[1,313 1,821]
(90%, 5%, 5%)	Reg. File	8,980	8,949	[8,238 9,661]
	ALU	509	779	[534 1,024]
	Remainder	511	81	[-337 498]

The numbers show that, despite all the sources of error in estimations that were enumerated in Sec. 5.4, the targeted cross sections, σ_{reg} and σ_{ALU}, can be estimated by the proposed method within ±5% of the actual value. However, the estimate for σ_{ALU} in the second fault distribution shows that the method may not be very accurate when a cross section is a very small portion of the overall cross section. The accuracy of

estimations are enhanced as the remainder portion of the system is broken into smaller units and more test programs are written to estimate their cross sections. Our method may not yield accurate estimates for small contributors to the overall cross section but it can determine the major contributors, which is the primary objective of the method.

5.6 Summary and Conclusions

In this chapter, we present a new method for SEU characterization of digital circuits and estimating the cross section of its constituent units during a radiation test. We use the fact that cross section of a chip changes under different conditions. One such condition is the utilization (duty factor) of the units. The other condition is the clock frequency. These two parameters are used to set up a system of linear equations based on a set of weighted test programs. Then, linear regression is used to solve the equations and obtain estimates for the cross section of each unit. The confidence interval of the estimated cross sections is shrunk by having more equations than unknowns.

The proposed method alleviates some of the limits of previous methods. Cross section granularity may be the primary limit of our technique. We used two parameters that affect cross sections: duty factor and clock frequency. If other parameters are found to affect cross sections, they can be used in a similar way to create linearly independent equations and achieve smaller granularity for cross sections.

Simulation results show that our method can determine the major contributors to the overall cross section and can give good estimation for cross sections of the corresponding units. Due to measurement errors that are enumerated in Sec. 5.4, our method may not yield accurate estimates for small contributors of the overall cross section, which is not a major drawback because the goal is to find the major contributors of the overall cross section.

One advantage of our method is that, unlike previous methods, it does not need any error location or error classification. This is because, for each test program, all the errors are collected and considered for the total cross section in the corresponding equation. The errors are separated indirectly (as opposed to direct tracing of the error to its source) when the individual cross sections are derived by solving the equations.

Chapter 6
The ARGOS Project: Experimental Results

The Stanford ARGOS project [Shirvani 98] is an experiment carried out on the computing test-bed of the NRL-801: *Unconventional Stellar Aspect* (USA) experiment on the *Advanced Research and Global Observations Satellite* (ARGOS) that was launched in February 1999. The ARGOS satellite [Wood 94] has a Sun-synchronous, 834-kilometer altitude orbit with a mission life of three years. The USA Experiment, one of the eight experiments that are flying on the ARGOS satellite, is primarily a low-cost X-ray astronomy experiment. The opportunity to perform an experiment in fault-tolerant computing evolved out of the need for a processor to analyze the X-ray data on-board, and autonomous navigation. The objective of the computing test-bed in the USA experiment on ARGOS is the comparative evaluation of approaches to reliable computing in a space environment, including radiation hardening of processors, architectural redundancy and fault tolerance software technology. This goal is met by flying processors and comparing performance in orbit during the ARGOS mission.

The experiment utilizes two 32-bit MIPS R3000 compatible processors. Each processor board is integrated as one double-sided 6U VME board containing the processor chip set, EEPROM, and 2M bytes of RAM:

- The *Hard board* uses the Harris RH3000 radiation hardened chip set, features a self-checking processor pair configuration running at 10MHz, has no cache memory[1], and has hardware EDAC for its 2MB SOI (Silicon On Insulator) SRAM memory. This board runs on VxWorks OS version 5.0.1.
- The *COTS board* uses a 25MHz 3081 microcontroller from IDT, uses only commercial off-the-shelf (COTS) components and has no hardware error detection mechanism except for parity for cache memories. The caches are configured as 4KB of instruction cache and 16KB of data cache. This board runs on VxWorks OS version 5.3.1.

Due to the difference in clock frequencies and the presence of cache memory on the COTS board, the COTS board is about 25 times faster than the Hard board.

The two boards are operated simultaneously in orbit. This is the first example of a so-called "McCluskey test", i.e., the simultaneous operation of commercial and hardened processors of the same class in the same orbital environment. Earlier experiments to gather fault-tolerance data have been limited in their scope. They either

[1] The chip set has caches but they were disabled due to a system bug.

implemented only one fault-tolerance technique and collected very limited data, or they artificially injected faults, which may not fully represent the condition in an actual space environment. In this research, we gathered data in an actual space environment, thereby avoiding the necessity of relying on questionable fault injection. Since the ARGOS satellite has a LEO polar orbit, a variety of radiation environments are encountered during this mission, providing a rigorous test. The boards execute test programs and collect data on the occurrence and detection of errors. It is possible to update the software on the boards based on the results received during the mission and test different SIHFT techniques.

This chapter presents the results of the experiments up to the end of January 2001. Section 6.1 summarizes the results of the Hard board. In Sec. 6.2, we briefly describe the software that was run on the COTS board, the error detection, correction and recovery technique used in the software and present the results of the experiments on this board. Section 6.3 concludes the chapter. Details of the processor board can be found in Appendix D.

6.1 Hard Board

The Hard board has hardware EDAC for memory and a self-checking processor pair. Moreover, the data and address buses have parity bits. Upon a mismatch between the master and shadow processors, an exception is generated leading to a system halt and reset. Uncorrectable memory errors or parity errors also lead to system halt.

Several errors have been observed in the Hard board. These errors occurred during the execution of two tests: a memory test and a sine table generation test. In the memory test, first a fixed pattern is written in a block of memory and then the contents of the block are read back and checked for correct pattern repeatedly (in a program loop). In the sine table generation test, first the sine values are loaded in a table and then the sine table entries are calculated and compared with the entries in the table repeatedly. The two tests were run in a multi-tasking fashion. Eight errors occurred in the first program and nine in the second for the data sizes and periods shown in Table 6.1.

Table 6.1 Errors in the Hard board.

Program	Data Size	Running Period	Number of Errors
Memory Test	256KB	140 days	4
	512KB	349 days	4
Sine Table Generation	128KB	191 days	3
	512KB	250 days	9

There was also one exception that led to a system halt. For all other errors, the checks in the test programs (comparison against correct value) detected the error and

reported it, and the program continued its execution. That means the master and shadow processors were in agreement on the errors. Therefore, the errors were not upsets in one of the processors. According to the EDAC status registers, there was no case of double errors that was not correctable by the EDAC hardware either. The number of errors in each program does not correlate with the data size, which could be an indication that the source of errors is not SEUs in main memory. With the limited information available, we could not pinpoint the source of these errors but the evidence suggests that they occur in a place that is a common source for both processors. One such place is the data buffers between memory and processors.

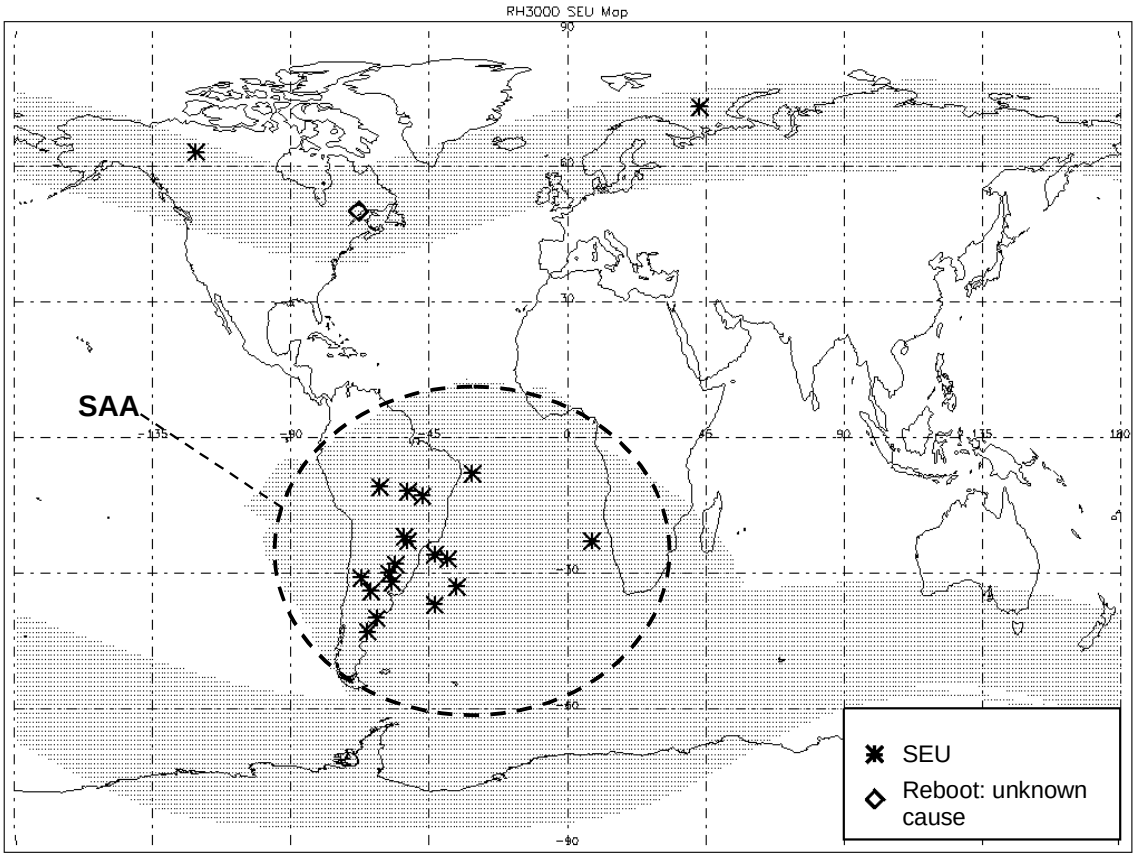

Figure 6.1 Geographical map of the upsets in the Hard board.

Figure 6.1 shows the geographical map of the upsets. The dark regions in this map are the regions with high density of radiation and are mainly two parts: (1) the *South Atlantic Anomaly* (SAA), marked with the dashed circle, and (2) the polar cusps. All the errors occurred in either the SAA region or near the polar cusps, which is a good indication that the upsets were caused by radiation.

The average error rate on the Hard board was 4×10^{-2} upsets per day (20 errors in 489 days) which is two orders of magnitude lower than average error rate on the COTS

board. This may be due to the different SRAM components on the two boards and not the different processors.

6.2 COTS Board

The major source of errors in the COTS board was SEUs in main memory. We did not get any clear indication of an SEU in the processor. Processor caches have parity bits and in case of parity error, the data is fetched from main memory. Therefore, test programs cannot detect SEUs in cache memories. However, when a cache parity error occurs, one of the flag bits in the status register of the 3081 processor is set. We periodically monitored this flag during our tests and did not observe any error indication.

The results of our test programs and fault tolerance techniques on this board are presented in the following subsections.

6.2.1 Memory Tests

Hundreds of memory SEUs (bit-flips) were observed by running two memory tests on the COTS board. The first test initializes a block of memory with a fixed pattern (55 hex) then continuously reads the contents and checks for the correct pattern[2]. This test was executed with different block sizes and with cached and non-cached addresses. The results are shown in Table 6.2. It can be observed that number of errors has a strong correlation with the data size and running period, indicating that most of the errors are due to SEUs in main memory. The difference observed in error rates could indicate that the source of some of the errors is not the main memory. To distinguish the SEUs in main memory from SEUs in other parts, the tests were written such that if an error is detected, the corresponding address is read again and checked for error. If the error persists in the second read, then we count it as a memory SEU. The experimental results indicate that all errors are in main memory (all second reads returned the same error). Therefore, the small differences are most probably due to the variation in radiation density and randomness of SEU occurrence.

Table 6.2 Errors in the COTS board from the memory test with fixed pattern.

Data Size	Running Period	Number of Errors	SEUs/MByte per day
512KB cached	18 days	41	4.56
256KB cached	36 days	48	5.33
128KB cached	84 days	54	5.14
128KB non-cached	84 days	59	5.62

[2] Execution of each iteration of this memory test is interleaved with the execution of other test programs.

The second memory test uses 7 different patterns with equal test time for each pattern shown in Table 6.3. Some test patterns use two values for alternative addresses. The memory chips on the COTS board are byte-wide (×8) and, for example, pattern 00,FF will write 00 to even addresses and FF to odd addresses in each chip. Except for the first two patterns that are all 0's and all 1's, respectively, all other patterns create equal number of 0's and 1's in the memory. This test collected 301 errors in 256KB of memory in 212 days, which yields an error rate of 5.68 SEUs/MB per day. The detailed results are shown in Table 6.3. Column 3 shows what percentage of total errors occurred during test with each pattern. Columns 4 and 6 break the number in column 2 into the number of bit-flips from 0 to 1 and from 1 to 0, respectively. Columns 5 and 7 translate the numbers in columns 4 and 6 into percentages for each row.

As Table 6.3 shows, the errors show some pattern dependency. Error rate is the lowest for pattern (00, FF) and highest for pattern (AA, 55). The 0 to 1, 1 to 0 ratio is not even and in fact is different for each pattern. This ratio is skewed most for patterns (00, FF) and (55, 55). This data can be used for a detailed analysis of the sensitivity of the memory cells if the circuit layout is available[3]. The results of this analysis could lead to circuit designs that are less sensitive to SEUs.

Table 6.3 Errors in the COTS board from the memory test with different patterns.

Pattern	Errors		0 to 1 bit-flip		1 to 0 bit-flip	
	Num	%	Num	%	Num	%
00,00	45	15.0	45	100.0	0	0.0
FF,FF	45	15.0	0	0.0	45	100.0
00,FF	34	11.3	12	35.3	22	64.7
FF,00	36	12.0	15	41.7	21	58.3
AA,AA	43	14.3	23	53.5	20	46.5
AA,55	52	17.3	23	44.2	29	55.8
55,55	46	15.3	17	37.0	29	63.0
Total	301	100	135	44.9	166	55.1

The average memory error rate calculated based on the two memory tests mentioned above and the software EDAC discussed in the next section is 5.5 SEUs/MByte per day. There were 7 cases of MBU in the errors collected by the two memory tests. Combining these cases with the MBU cases from software EDAC, yields that, in our experiment, MBUs occurred in 1.44% of SEUs in memory.

Figure 6.2 shows the geographical map of the SEUs in the COTS board. We see the concentration of the upsets in the SAA region as was the case for the Hard board (Fig. 6.1). There are also cases of upsets in regions with low radiation density (white parts in

[3] We did not have access to this information.

the map). This indicates that the SRAMs on the COTS board are sensitive to particle energy levels in these regions, too. Therefore, error detection and correction is necessary even for satellite orbits that do not go through high radiation regions.

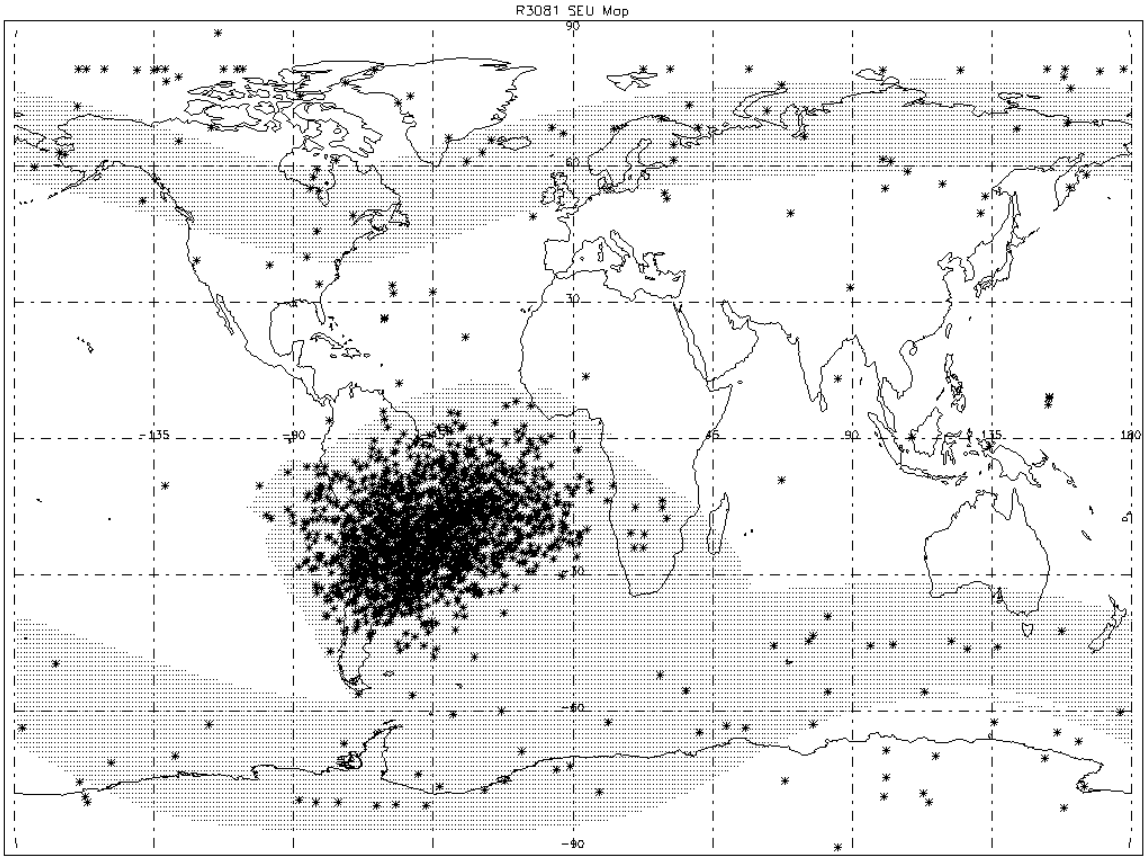

Figure 6.2 Geographical map of the upsets in the COTS board.

6.2.2 Software-Implemented EDAC

There is no hardware EDAC to protect the main memory of the COTS board. In the early stages of the experiment, we observed that SEUs corrupted the memory, forcing frequent system resets. We implemented EDAC in software and used periodic scrubbing to protect the code segments of OS and application programs (details of our technique are described in Chap. 4 and Appendix B). The scrub interval we used was 30 seconds. Software EDAC took about 3% of the CPU time which is a low performance overhead. Without software EDAC, the system worked an average of 2 days before it crashed and needed a reset. With software EDAC this average was increased to about 20 days[4]. Therefore, software EDAC improved the availability of the COTS board by an order of magnitude.

[4] The longest period we had at the time of this writing was 41 days.

On average, 450KB of memory was protected by the software EDAC for 329 days. During this period, 831 errors were detected and corrected. There was no case of uncorrectable error. Among these errors, there were 12 cases of MBU. All cases were double-bit errors. The SEU rate calculated from these errors is 5.66 SEUs/MByte per day with 1.5% MBU (calculated as 12/(831-12)).

6.2.3 Software-Implemented Error Detection and Recovery

We implemented several software-implemented error detection techniques including: duplication, software TMR, algorithm-based fault tolerance (ABFT). However, due to limited experiment time, we concentrated on testing two SIHFT techniques that were developed at Stanford CRC[5]. These techniques are briefly explained below.

Transient errors that occur in a processor can be detected by executing a program multiple times, and comparing the outputs produced by each execution. Duplication can be done at the task level by the programmer or by the OS. It can also be done at the instruction level during program compilation. *Error Detection by Duplicated Instructions* (EDDI) technique uses the latter approach. In this technique, the original instructions (master instructions) are interleaved with the duplicated instructions (shadow instructions). Data structures are also duplicated. Computation results from master and shadow instructions are compared before each store operation. Upon mismatch, the program jumps to an error handler that will cause the program to restart. Since code and data are duplicated, memory SEUs can also be detected. Details of this technique can be found in [Oh 01a].

EDDI can only detect some of the control-flow errors. To enhance the detection coverage for this type of error, the *Control-Flow Checking by Software Signatures* (CFCSS) technique is used. CFCSS is an *assigned* signature method where unique signatures are associated with each block during compilation. These signatures are embedded into the program using the immediate field of instructions that use constant operands. A run-time signature is generated and compared with the embedded signatures when instructions are executed. Details of this technique can be found in [Oh 01b].

Some errors may cause a task to hang. A watchdog timer was implemented to detect these errors. There are also errors that are detected by hardware (e.g., reserved

[5] Another limiting factor in our experiment was the low error rate. We had to run each test for a long time to collect statistically significant number of errors. This limited the number of test programs and consequently, the variety of fault tolerance techniques that we could execute and test.

instruction, or illegal address). For these errors, an exception is generated and the OS suspends the faulty task. Since the task does not respond to the main control program in time, the watchdog timer expires and the task is restarted.

To facilitate error recovery, we broke our test software into modules and ran each module as a separate task. A main module controls the execution of all the other modules. When one of the error detection mechanisms detects an error, the erroneous module is aborted and restarted without corrupting the context of the other modules.

EDDI and CFCSS are pure software techniques for detecting hardware errors. These techniques do not require any changes in hardware or any support from the OS. We have applied the combination of EDDI and CFCSS to two sort algorithms (Insert sort and Quick sort) and an FFT algorithm. The input to all three programs is an array with 1024 entries. The insert sort algorithm was tested with both integer and floating-point numbers. We did an assertion check after each execution to see if there was an undetected error. For the sort algorithms, we checked that the data is sorted correctly. For the FFT algorithm, we calculated a checksum of the results and compared it against the expected checksum stored in the program.

Each of the computation test programs has two parts: the core part of the code implements the algorithm, and the wrapper part does the data initialization and communicates with the main control program. EDDI and CFCSS are only implemented for the core part and not the wrapper due to the limitations of these techniques with function calls to the OS. EDDI and CFCSS can detect bit-flips in the code and data segments of the programs. To test this capability, code segment of the core part of the computation test programs are not periodically scrubbed for errors by the software EDAC. Once an error is detected by either EDDI or CFCSS, the corresponding task is terminated and restarted. If the error was caused by an SEU in processor or in data segment in memory, the first restart is successful. If the task stops on first restart, it is most probably due to an SEU in code segment in memory. In this case, a request is sent to software EDAC to check for errors and a second restart is attempted. In case the second restart fails, the task is considered unrecoverable and is dropped from the list of test programs that are being executed. These failures occur, for example, when the OS data structures (which are not protected by software EDAC) become corrupt due to SEUs.

To increase the number of collected errors, we increased the probability of getting an error by expanding the size of code or data in each test program. For data expansion, we replicated the input data array and in each iteration of the program, used one of the inputs in a round-robin fashion. For code expansion, we replicated the code and in each

iteration of the program, executed one of the copies in a round-robin fashion. Columns 2 and 3 of Table 6.4 show the code and data sizes for each of the test programs. Columns 4 and 5 show the running time and the number of detected errors in each test program, respectively. Column 6 is the normalized error rate based on the numbers in columns 2 to 5. The Insert Sort test with floating-point numbers showed an unusually high error rate but its running time was not long enough to support any conclusion. For the Quick Sort and FFT programs, we used two different code or data sizes[6]. The results from the FFT programs with two large data sizes show that the number of errors scales with the amount of memory used by the program and the error rates are close to the rate derived from the memory tests. The error rate of the Quick Sort programs with two large codes sizes is lower than other tests because not all the SEUs in the code segment cause errors in the program. This is due to the fact that the SEU may occur in the initialization part of the program (that is executed once at the program start and never used again until a program restart), or in bits of an instruction that are not used in the instruction opcode, or in unused words of the code segment. As the numbers show, this effect is enhanced for larger code size (row 5). Overall, the results indicate that the majority of the errors were caused by SEUs in memory. We did not get any indication of an error in the processor.

Table 6.4 Errors in the COTS board from the computation tests.

Program	Code Size	Data Size	Running Period	Number of Errors	SEUs/MB per day
Insert Sort – Integer	4 KB	160 KB	178 days	156	5.47
Insert Sort – Floating Point	4 KB	80 KB	26 days	21	9.85
Quick Sort – Integer	50 KB	8 KB	54 days	13	4.25
	75 KB	8 KB	132 days	30	2.80
FFT	6 KB	160 KB	41 days	42	6.32
	6 KB	80 KB	132 days	60	5.41

Table 6.5 shows the total number of errors in each test program and the detection mechanism that detected these errors. Most of the errors were detected by EDDI. However, there are errors detected by CFCSS and watchdog timer, especially for tests with larger code sizes. This shows the significance of these two detection mechanisms in complementing the detection coverage of EDDI. Overall, the combination of EDDI, CFCSS and watchdog timer detected a total of 321 errors. There was one case of undetected error (detected by assertion check explained above). These numbers yield a

[6] This is an example implementation of the weighted test programs method presented in Chapter 5 with $\sigma_T = \sigma_{memory} f_{memory} + \sigma_{processor} f_{processor}$.

99.7% error detection coverage. Out of the 321 detected errors, 4 cases had unsuccessful recovery. This shows that the implemented recovery technique had a 98.8% success rate.

Table 6.5 Errors detected by each SIHFT technique and undetected errors in the computation tests on the COTS board.

Program	Total Num. of Errors	Errors Detected by Each Technique			Undetected Errors
		EDDI	CFCSS	Watchdog Timer	
Insert Sort – Integer	156	156	–	–	–
Insert Sort – Floating Point	21	21	–	–	–
Quick Sort – Integer	43	31	5	6	**1**
FFT	102	99	1	2	–
Total	322	307	6	8	**1**

6.3 Throughput Comparison

The performance overhead of the error detection techniques used in the COTS board is about 170%. The COTS board processor (IDT3081) ran at 25MHz and had cache memory, and the Hard board processor (RH3000) ran at 10MHz and did not have cache memory due to a system bug. Therefore, the COTS board was about 25 times faster than the Hard board[7] which offsets the performance overhead of the SIHFT techniques used on the COTS board. Overall, the throughput of the COTS board was an order of magnitude higher than the throughput of the Hard board. The throughputs would be about the same if the radiation-hardened board had cache memory.

6.4 Summary

The results from the Hard board show that despite all the hardware fault tolerance techniques used in the board, there are cases of undetected errors. Even if single points of failure are eliminated by better design, additional fault tolerance techniques, perhaps in software, may still be required for high reliability.

As expected, on the COTS processor board, memory SEUs were the main source of errors in the ARGOS experiments. Our memory tests collected 503 errors during 350 days. The geographical maps show concentration of SEUs in the *South Atlantic Anomaly* (SAA) region and the polar cusps, which matches with the previous published data ([Underwood 92][Hosken 97][Underwood 98]). We also identified the MBUs which consisted 1.44% of the SEU events. We used different patterns in our memory tests and found some pattern dependency in the error rate and the ratio of 0-to-1 and 1-to-0 bit-flips.

[7] We compared the performance of the two boards by measuring the execution time of 1000 iterations of a sort program which had no error detection (no SIHFT technique).

The software-implemented error detection and recovery techniques that we have used in ARGOS have been effective for the error rate observed in the COTS board. Software EDAC improved the availability of the COTS board by an order of magnitude and had only 3% performance overhead. Even though hardware EDAC would be preferable for main memory, software EDAC provided acceptable reliability for our experiment. The experimental results show 99.7% error detection coverage and 98.8% error recovery coverage. Our results show that COTS with SIHFT are viable techniques for low-radiation environments such as the LEO orbit of the ARGOS satellite.

Chapter 7
Concluding Remarks

Fault tolerance is an important aspect of computers and digital systems. In this dissertation, we focused on faults caused by radiation. The challenge in the ARGOS project was to determine to what extent commercial electronics, such as microprocessors and RAMs, can be used in space. To answer this question, we collected data in an actual space environment thereby avoiding the necessity of relying on questionable fault injection methods. We executed test programs on a radiation-hardened board and a COTS board at the same time and compared their behavior.

The results from the radiation-hardened processor board show that despite all the hardware fault tolerance techniques used in the board, there are cases of undetected errors. Even if single points of failure are eliminated by better design, additional fault tolerance techniques, perhaps in software, may still be required for higher reliability.

For the COTS board, we developed, implemented, and tested software-implemented error detection, correction and recovery techniques that do not require any hardware assistance. The experimental results show 99.7% error detection coverage and 98.8% error recovery coverage. Despite the performance overhead of the SIHFT techniques, the throughput of the COTS board was an order of magnitude higher than that of the Hard board due to the higher CPU speed and cache memory on the COTS board.

The conclusion of our project is that COTS with SIHFT are viable techniques for low-radiation environments such as the LEO orbit of the ARGOS satellite. We demonstrated this by successful operation of COTS with SIHFT in the ARGOS satellite, despite the memory upset rate of 5.55 SEUs/MByte per day that we experience in orbit.

We developed techniques for tolerating permanent and transient faults and devised a method for SEU characterization of digital circuits. Even though, our focus was faults caused by radiation, the techniques can be used for permanent or transient faults caused by other sources such as electromigration (for permanent faults), or coupling noise and interference (for transient faults). Moreover, there is a growing concern about radiation-induced faults in deep sub-micron technologies even for commercial components that are used at ground level.

As technology advances toward nanometer feature sizes and low-power devices, new reliability concerns may arise. SEGR and microdose (Chap. 2) are examples of possible new failure mechanisms in new technologies that we may have to deal with even for commercial applications. Therefore, fault tolerance could become a commodity feature in future electronic devices.

References

[Abraham 83] Abraham, J.A., E.S. Davidson and J.H. Patel, "Memory System Design for Tolerating Single Event Upsets," *IEEE Trans. Nucl. Sci.*, Vol. 30, No. 6, pp. 4339-44, Dec. 1983.

[Abramovici 90] Abramovici, M. et al., *Digital Systems Testing And Testable Design*, IEEE Press, 1990.

[Agarwal 86] Agarwal, A., R.L. Sites, M. Horowitz, "ATUM: A New Technique for Capturing Address Traces Using Microcode," *Proc. 13th Annu. Symp. Comput. Architecture*, pp. 119-127, June 1986.

[Asenek 98] Asenek, V., et al., "SEU Induced Errors Observed in Microprocessor Systems," *IEEE Trans. on Nuclear Science*, Vol. 45, No. 6, pp. 2876-83, Dec. 1998.

[Avizienis 71] Avizienis, A., "Arithmetic Error Codes: Cost and Effectiveness Studies for Application in Digital System Design," *IEEE Trans. Comp.*, C-20, pp. 1322-1331, Nov. 1971.

[Baumann 00] Baumann, R.C., and E.B. Smith, "Neutron-Induced Boron Fission as a Major Source of Soft Errors in Deep Submicron SRAM Devices," *IEEE Intr'l Reliability Physics Symp.*, pp.152-7, April 2000.

[Baumann 95] Baumann, R.C., et al., "Boron Compounds as a Dominant Source of Alpha Particles in Semiconductor Devices," *IEEE Intr'l Reliability Physics Symp.*, pp. 297-302, 1995.

[Baze 97] Baze, M.P., and S.P. Buchner, "Attenuation of Single Event Induced Pulses in CMOS Combinational Logic," *IEEE Trans. on Nuclear Science*, Vol. 44, No. 6, pp. 2217-23, Dec. 1997.

[Beahan 00] Beahan, J., et al., "Detailed Radiation Fault Modeling of the Remote Exploration and Experimentation (REE) First Generation Testbed Architecture," *Proc. IEEE Aerospace Conf.*, pp. 278-281, Mar. 2000.

[Brown 97] Brown, G.R., "Honeywell Radiation Hardened 32-bit Processor Central Processing Unit, Floating Point Processor, and Cache Memory Dose Rate and Single Event Effects Test Results," *IEEE Radiation Effects Data Workshop*, pp. 110-5, 1997.

[Buchner 91] Buchner, S., et al., "Charge Collection in GaAs MESFETs and MOSFETs," *IEEE Trans. on Nuclear Science*, Vol. 38, No. 6, pp. 1370-6, Dec. 1991.

[Buchner 96] Buchner, S., et al., "Modification of Single Event Upset Cross Section of an SRAM at High Frequencies," *Radiation Effects on Components and Systems Conf.*, pp. 326-32, 1996.

[Buchner 97] Buchner, S., et al., "Comparison of Error Rates in Combinational and Sequential Logic," *IEEE Trans. on Nuclear Science*, Vol. 44, No. 6, pp. 2209-16, Dec. 1997.

[Burnett 93] Burnett, D., et al., "Soft-Error-Rate improvement in advanced BiCMOS SRAMs," *IEEE Intr'l Reliability Physics Symp.*, pp.156-160, March 1993.

[Calin 96] Calin, T., M. Nicolaidis, and R. Velazco, "Upset Hardened Memory Design for Submicron CMOS Technology," *IEEE Trans. on Nuclear Science*, Vol. 43, No. 6, pp. 2874-2878, Dec. 1996.

[Cha 93] Cha, H., et al., "A Fast and Accurate Gate-Level Transient Fault Simulations Environment," *IEEE Intr'l Symp. on Fault-Tolerant Computing (FTCS-23)*, pp. 310-9, June 1993.

[Chen 84] Chen, C.L., and M.Y. Hsiao, "Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review," *IBM J. Res. Develop.*, Vol. 28, pp. 124-134, Mar. 1984.

[Chen 00] Chen, F., et al., "Demonstration of the Remote Exploration and Experimentation (REE) Fault-Tolerant Parallel-Processing Supercomputer for Spacecraft Onboard Scientific Data Processing," *Proc. Intr'l Conf. on Dependable Systems and Networks* (FTCS-30 and DCCA-8), pp. 367-372, New York, NY, June 25-28, 2000.

[Cusick 85] Cusick, J., et al., "SEU Vulnerability of the Zilog Z-80 and NSC-800 Microprocessors," *IEEE Trans. on Nuclear Science*, Vol. NS-32, No. 6, pp. 4206-11, Dec. 1985.

[Cypress 99] Data Sheets of Cypress Semiconductor Corp. SRAMs, http://www.cypress.com/design/datasheets/index.html, Cypress Semiconductor Corp., 1999.

[Dai 99] Dai, C., et al., "Alpha-SER Modeling & Simulation for Sub-0.25μm CMOS Technology," *Proc. Symp. On VLSI Tech.*, pp. 81-2, 1999.

[Dale 95] Dale, C.J., et al., "Fiber Optic Data Bus Space Experiment on Board the Microelectronics and Photonics Test Bed (MPTB)," *Proc. of the SPIE – The Intr'l Society for Optical Eng.*, Vol. 2482, pp. 285-293, April 1995.

[Dell 97] Dell, T.J., "A White Paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory," http://www.chips.ibm.com:80/products/memory/chipkill/chipkill.html, IBM Microelectronics Division, Rev. 11/19/97.

[DineroIV 98] "Dinero IV Trace-Driven Uniprocessor Cache Simulator", http://www.cs.wisc.edu/~markhill/DineroIV/, Rel. 7, Feb. 1998.

[Elder 88] Elder, J.H., et al., "A Method for Characterizing a Microprocessor's Vulnerability to SEU," *IEEE Trans. on Nuclear Science*, Vol. 35, No. 6, pp. 1678-81, Dec. 1988.

[Feldmeier 95] Feldmeier, D.C., "Fast Software Implementation of Error Detection Codes," *IEEE/ACM Trans. Networking*, Vol. 3, No. 6, pp. 640-651, Dec. 1995.

[Fleetwood 94] Fleetwood, D.M., et al., "Accounting for Time-Dependent Effects on CMOS Total-Dose Response in Space Environment," *Radiat. Phys. Chem.* 43, No. 1/2, pp. 129-138, 1994.

[Friedman 85] Friedman, A.L., et al., "Single Event Upsets in Combinatorial and Sequential Current Mode Logic," *IEEE Trans. on Nuclear Science*, Vol. NS-32, No. 6, pp. 4216-18, Dec. 1985.

[Gardic 96] Gardic, F., et al., "Dynamic Single Event Effects in a CMOS/Thick SOI Shift Register," *Radiation Effects on Components and Systems Conf.*, pp. 333-9, 1996.

[Goodman 91] Goodman, R.M., et al., "The Reliability of Semiconductor RAM Memories with On-Chip Error-Correction Coding," *IEEE Trans. on Information Theory*, Vol. 37, No. 3, pp. 884-96, May 1991.

[Gussenhoven 97] Gussenhoven, M.S., et al., "APEXRAD: Low Altitude Orbit Dose as a Function of Inclination, Magnetic Activity and Solar Cycle," *IEEE Trans. Nucl. Sci.*, Vol. 44, No. 6, pp. 2161-2168, Dec. 1997.

[Hareland 00] Hareland, S., et al., "Intel Perspectives on SER," *Topical Research Conf. On Reliability*, pp. 28, Oct. 30 – Nov. 1, 2000.

[Hash 97] Hash, G.L., et al., "Radiation Hardness Assurance Categories for COTS Technologies," *IEEE Radiation Effects Data Workshop*, pp. 35-40, 1997.

[Hasnain 92] Hasnain, Z., and A. Ditali, "Building-in Reliability: Soft Errors – A Case Study," *IEEE Intr'l Reliability Physics Symp*, pp. 276-280, 1992.

[Hennessy 96] Hennessy, J.L., and D.A. Patterson, *Computer Architecture, A Quantitative Approach*, 2nd Edition, Morgan Kaufmann Pub., Inc., San Mateo, CA, 1996.

[Hiemstra 99] Hiemstra, D.M., and A. Baril, "Single Event Upset Characterization of the Pentium MMX and Pentium II Microprocessors Using Proton Irradiation," *IEEE Trans. on Nuclear Science*, Vol. 46, No. 6, pp. 1453-60, Dec. 1999.

[Hines 90] Hines, W.H., and D.C. Montgomery, *Probability and Statistics in Engineering and Management Science*, 3rd ed., (Chapter 15), John Wiley & Sons, Inc., 1990.

[Hodgart 92] Hodgart, M.S., "Efficient Coding and Error Monitoring for Spacecraft Digital Memory," *Int'l J. Electronics*, Vol. 73, No. 1, pp. 1-36, 1992.

[Holmes-Siedle 93] Holmes-Siedle, A., and L. Adams, *Handbook of Radiation Effects*, Oxford University Press, 1993.

[Hosken 97] Hosken, R., et al., "Investigation of Non-Independent Single Event Upsets in the TAOS GVSC Static RAM," *IEEE Radiation Effects Data Workshop*, pp. 53-60, July 1997.

[Houtt 97] Houtt, W.V., et al., "Programmable Computer System Element with Built-In Self-Test Method and Apparatus for Repair During Power-On," *U.S. Patent 5,659,551*, Aug. 1997.

[IEEE-TNS] IEEE Trans. On Nuclear Science, Dec. issues.

[Johansson 99] Johansson, K., et al., "Neutron Induced Single-word Multiple-bit Upset in SRAM," *IEEE Trans. on Nuclear Science*, Vol. 46, No. 6, pp. 1427-1433, Dec. 1999.

[Kang 86] Kang, S.M., and D. Chu, "CMOS Circuit Design for Prevention of Single Event Upset," *IEEE Intr'l Conf. On Computer Design*, pp. 385-388, Port Chester, NY, Oct. 1986.

[Kimbrough 94] Kimbrough, J.R., et al., "Single Event Effects and Performance Prediction for Space Applications of RISC Processors," *IEEE Trans. on Nuclear Science*, Vol. 41, No. 6, pp. 2706-14, Dec. 1994.

[Koga 84] Koga, R., and W.A. Kolasinski, "Heavy Ion-Induced Single Event Upsets of Microcircuits: A Summary of the Aerospace Corporation Test Data," *IEEE Trans. on Nuclear Science*, Vol. 31, No. 6, pp. 1190-1195, Dec. 1984.

[Koga 85] Koga R., et al., "Techniques of Microprocessor Testing and SEU-Rate Prediction," *IEEE Trans. on Nuclear Science*, Vol. NS-32, No. 6, pp. 4219-24, Dec. 1985.

[Koga 93] Koga, R., et al., "Single-word Multiple-bit Upsets in Static Random Access Devices," *IEEE Trans. on Nuclear Science*, Vol. 40, No. 6, pp. 1941-1946, Dec. 1993.

[Kouba 97] Kouba, C.K., and Gwan Choi, "The Single Event Upset Characterization of the 486-DX4 Microprocessor," *IEEE Radiation Effects Data Workshop*, pp. 48-52, 1997.

[LaBel 96] LaBel, K.A., et al., "Commercial Microelectronics Technologies for Applications in the Satellite Radiation Environment," *IEEE Aerospace Applications Conf.*, Vol. 1, pp. 375-390, 1996.

[Liden 94] Liden, P., et al., "On Latching Probability of Particle Induced Transients in Combinational Networks," *IEEE Intr'l Symp. on Fault-Tolerant Computing (FTCS-24)*, pp. 340-9, June 1994.

[Liu 97] Liu, J., et al., "Heavy Ion Induced Single Event Effects in Semiconductor Device," *17th Int'l. Conf. on Atomic Collisions in Solids (in Nucl. Instrum. & Methods in Physics Res.)*, Sec. B, Vol. 135, No. 1-4, pp. 239-243, 1997.

[Lucente 90] Lucente, M.A., C.H. Harris and R.M. Muir, "Memory System Reliability Improvement Through Associative Cache Redundancy," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 19.6.1-19.6.4, May 1990.

[Lum 97] Lum, G.K., et al., "System Hardening Approaches for a LEO Satellite with Radiation Tolerant Parts," *IEEE Trans. Nuclear Science*, Vol. 44, No. 6, pp. 2026-2033, Dec. 1997.

[Luo 94] Luo, X. and J.C. Muzio, "A Fault-Tolerant Multiprocessor Cache Memory," *Proc. IEEE Workshop on Memory Technology, Design and Testing*, pp. 52-57, August 1994.

[McCluskey 86] McCluskey, E.J., *Logic Design Principles*, Prentice-Hall, Inc., 1986.

[Melinger 94] Melinger, J.S., et al., "Critical Evaluation of the Pulsed Laser Method for Single Event Effects Testing and Fundamental Studies," *IEEE Trans. on Nuclear Science*, Vol. 41, No. 6, pp. 2574-84, Dec. 1994.

[Messenger 92] Messenger, G.C., and M.S. Ash, *The Effects of Radiation on Electronic Systems*, 2nd ed., Van Nostrand Reinhold, 1992.

[Moran 96] Moran, A., "Single Event Effect Testing of the Intel 80386 Family and the 80486 Microprocessor," *IEEE Trans. on Nuclear Science*, Vol. 43, No. 3, pt. 1, pp. 879-85, June 1996.

[MPTB] http://ssdd.nrl.navy.mil/www/mptb/index.htmlx, Feb. 2001.

[Mueller 99] Mueller, M., et al., "RAS Strategy for IBM S/390 G5 and G6," *IBM J. Res. Develop.*, Vol. 43, No. 5/6, pp. 875-888, Sep./Nov. 1999.

[Nicolaidis 99] Nicolaidis, M., "Time Redundancy Based Soft-Error Tolerance to Rescue Nanometer Technologies," *IEEE VLSI Test Symp.*, pp. 86-94, April 1999.

[Nikolos 96] Nikolos, D. and H.T. Vergos, "On the Yield of VLSI Processors with On-Chip CPU Cache," *Proc. 2nd European Dependable Computing Conference*, pp. 214-229, October 1996.

[Normand 96] Normand, E., "Single-Event Effects in Avionics," *IEEE Trans. Nuclear Science*, Vol. 43, No. 2, pp. 461-474, April 1996.

[NSREC] *Proc. IEEE Nuclear and Space Radiation Effects Conference* (NSREC) and *Radiation Effects Workshop*, http://www.nsrec.com/

[O'Gorman 94] O'Gorman, T.J., "The Effect of Cosmic Rays on the Soft Error Rate of a DRAM at Ground Level," *IEEE Trans. Electron Devices*, Vol. 41, No. 4, pp. 553-557, April 1994.

[O'Leary 89] O'Leary, B.J., A.J. Sutton, "Dynamic Cache Line Delete," *IBM Tech. Disclosure Bull.*, Vol. 32, No. 6A, pp. 439, Nov. 1989.

[Oh 01a] Oh, N., P.P. Shirvani and E.J. McCluskey, "Error Detection by Duplicated Instruction in Superscalar Microprocessors," *IEEE Trans. Reliability*, (scheduled to appear in the Sep. 2001 issue).

[Oh 01b] Oh, N., P.P. Shirvani and E.J. McCluskey, "Control-Flow Checking by Software Signatures," *IEEE Trans. Reliability*, (scheduled to appear in the Sep. 2001 issue).

[Ohring 98] Ohring, M., *Reliability and Failure of Electronic Materials and Devices*, Academic Press, Sec. 7.6, 1998.

[Oldfield 98] Oldfield, M.K., and C.I. Underwood, "Comparison Between Observed and Theoretically Determined SEU Rates in the TEXAS TMS4416 DRAMs and On-Board the UoSAT-2 Microsatellite," *IEEE Trans. on Nuclear Science*, Vol. 45, No. 3, pp. 1590-1594, June 1998.

[Oldham 93] Oldham, T.R., et al., "Total Dose Failures in Advanced Electronics from Single Ions," *IEEE Trans. Nuclear Science*, Vol. 40, No. 6, pp. 1820-1830, Dec. 1993.

[Ooi 92] Ooi, Y., M. Kashimura, H. Takeuchi, and E. Kawamura, "Fault-Tolerant Architecture in a Cache Memory Control LSI," *IEEE J. of Solid-State Circuits*, Vol. 27, No. 4, pp. 507-514, April 1992.

[Paschburg 74] Paschburg, R.H., "Software Implementation of Error-Correcting Codes," Univ. Illinois, Urbana, (AD-786 542), Aug. 1974.

[Patel 74] Patel, A.M., and S.J. Hong, "Optimal Rectangular Code for High Density Magnetic Tapes," *IBM J. Res. Develop.*, Vol. 18, pp. 579-88, November 1974.

[Patel 85] Patel, A.M., "Adaptive Cross-Parity (AXP) Code for a High-Density Magnetic Tape Subsystem," *IBM J. Res. Develop.*, Vol. 29, pp. 546-62, November 1985.

[Patterson 83] Patterson, D.A., et al., "Architecture of a VLSI Cache for a RISC," *Proc. Int'l Symp. Comp. Architecture*, Vol. 11, No. 3, pp. 108-116, June 1983.

[Pour 93] Pour, A.F. and M.D. Hill, "Performance Implications of Tolerating Cache Faults," *IEEE Trans. Comp.*, Vol. 42, No. 3, pp. 257-267, March 1993.

[RADECS] Association RADECS (RADiation and its Effects on Components and Systems), http://www.radecs.org/

[Radhome] Radiation Effects and Analysis, NASA Goddard Space Flight Center, http://radhome.gsfc.nasa.gov/top.htm

[Radnet] JPL Radiation Effects Database, http://radnet.jpl.nasa.gov/

[Rao 89] Rao, T.R.N., and E. Fujiwara, *Error-Control Coding for Computer Systems*, Prentice Hall, 1989.

[REE] http://www-ree.jpl.nasa.gov/index.html, Feb. 2001.

[Reed 96] Reed, R.A., et al., "Single Event Upset Cross Sections at Various Data Rates," *IEEE Trans. on Nuclear Science*, Vol. 43, No. 6, pp. 2862-7, Dec. 1996.

[Reed 97] Reed, R., et al., "Heavy Ion and Proton-Induced Single Event Multiple Upset," *IEEE Trans. Nucl. Sci.*, Vol. 44, No. 6, pp. 2224-9, July 1997.

[Saleh 90] Saleh, A.M., et al., "Reliability of Scrubbing Recovery-Techniques for Memory Systems," *IEEE Trans. on Reliability*, Vol. 39, No. 1, pp. 114-22, April 1990.

[Sarwate 88] Sarwate, D.V., "Computation of Cyclic Redundancy Checks via Table Look-up," *Communications of the ACM*, Vol. 31, No. 8, pp. 1008-13, Aug. 1988.

[Saxena 95] Saxena, N.R., et al., "Fault-Tolerant Features in the HaL Memory Management Unit," *IEEE Trans. Comp.*, Vol. 44, No. 2, pp.170-179, February 1995.

[Schneiderwind 92] Schneiderwind, R., et al., "Laser Confirmation of SEU Experiments in GaAs MESFET Combinational Logic," *IEEE Trans. on Nuclear Science*, Vol. 39, No. 6, pp. 1665-70, Dec. 1992.

[SEECA] Single Event Effect Criticality Analysis, http://radhome.gsfc.nasa.gov/radhome/papers/seecai.htm

[Seifert 01] Seifert, N., et al., "Historical Trend in Alpha-Particle Induced Soft Error Rates of the Alpha Microprocessor," *IEEE Intr'l Reliability Physics Symp.*, April 30 – May 3, pp. 259-265, 2001.

[Sexton 90] Sexton, F.W., et al., "SEU Characterization and Design Dependence of the SA3300 Microprocessor," *IEEE Trans. on Nuclear Science*, Vol. 37, No. 6, pp. 1861-8, Dec. 1990.

[Shaeffer 92] Shaeffer, D.L., et al., "Proton-Induced SEU, Dose Effects, and LEO Performance Predictions for R3000 Microprocessors," *IEEE Trans. on Nuclear Science*, Vol. 39, No. 6, pp. 2309-15, Dec. 1992.

[Shaneyfelt 94] Shaneyfelt, M.R., et al., "Hardness Variability in Commercial Technologies," *IEEE Trans. Nuclear Science*, Vol. 41, No. 6, pp. 2536-2543, Dec. 1994.

[Shirvani 98] Shirvani, P.P. and E.J. McCluskey, "Fault-Tolerant Systems in a Space Environment: The CRC ARGOS Project," *CRC-TR 98-2*, Stanford University, Stanford, CA, Dec. 1998.

[Shirvani 99] Shirvani, P.P. and E.J. McCluskey, "PADded Cache: A New Fault-Tolerance Technique for Cache Memories," *17th IEEE VLSI Test Symposium*, pp. 440-445, Dana Point, CA, Apr. 24-29, 1999.

[Shirvani 00a] Shirvani, P.P., N. Oh, E.J. McCluskey, D. Wood and K.S. Wood, "Software-Implemented Hardware Fault Tolerance Experiments; COTS in Space," *Proc. International Conference on Dependable Systems and Networks* (FTCS-30 and DCCA-8), Fast Abstracts, pp. B56-7, New York, NY, June 25-28, 2000.

[Shirvani 00b] Shirvani, P.P., N. Saxena and E.J. McCluskey, "Software-Implemented EDAC Protection Against SEUs," *IEEE Trans. on Reliability, Special Section on Fault-Tolerant VLSI Systems*, Vol. 49, No. 3, pp. 273-284, Sep. 2000.

[Shoga 94] Shoga, M., et al., "Single Event Upset at Gigahertz Frequencies," *IEEE Trans. on Nuclear Science*, Vol. 41, No. 6, pp. 2252-8, Dec. 1994.

[Siewiorek 92] Siewiorek, D.P., and R.S. Swarz, *Reliable Computer Systems: Design and Evaluation*, 2nd Edition, Digital Press, Burlington, MA, 1992.

[Sohi 89] Sohi, G. S., "Cache Memory Organization to Enhance the Yield of High-Performance VLSI Processors," *IEEE Trans. Comp.*, Vol. 38, No. 4, pp. 484-492, April 1989.

[Swift 94] Swift, G.M., et al., "A New Class of Single Event Hard Errors," *IEEE Trans. Nuclear Science*, Vol. 41, No. 6, pp. 2043-2048, Dec. 1994.

[Takano 96] Takano, T., et al., "In-orbit experiment on the fault-tolerant space computer aboard the satellite Hiten," *IEEE Trans. on Reliability*, Vol.45, No. 4, pp. 624-631, Dec. 1996.

[Thomlinson 87] Thomlinson, J., et al., "The SEU and Total Dose Response of the INMOS Transputer," *IEEE Trans. on Nuclear Science*, Vol. 34, No. 6, pp. 1803-7, Dec. 1987.

[Tosaka 96] Tosaka, Y., et al., "Impact of Cosmic Ray Neutron Induced Soft Errors on Advanced Submicron CMOS Circuits," *VLSI Technology Symp.*, pp. 148-9, 1996.

[Turgeon 91] Turgeon, P.R., A.R. Stell, M.R. Charlebois, "Two Approaches to Array Fault Tolerance in the IBM Enterprise System/9000 Type 9121 Processor," *IBM J. Res. Develop.*, Vol. 35, No. 3, pp. 382-389, May 1991.

[Tylka 97] Tylka, A.J., et al., "CREME96: A Revision of the Cosmic Ray Effects on Micro-Electronics Code," *IEEE Trans. Nucl. Sci.*, Vol. 44, No. 6, pp. 2150-2160, Dec. 1997.

[Uhlig 95] Uhlig, R., et al., "Instruction Fetching: Coping with Code Bloat," *Proc. 22nd Int'l Symp. Comp. Architecture*, pp. 345-356, June 1995.

[Underwood 92] Underwood, C.I., et al., "Observation of Single-Event Upsets in Non-Hardened High-Density SRAMs in Sun Synchronous Orbit," *IEEE Trans. Nucl. Sci.*, Vol. 39, No. 6, pp. 1817-1827, Dec. 1992.

[Underwood 97] Underwood, C.I., "The Single-Event-Effect Behavior of Commercial-Off-The-Shelf Memory Devices – A Decade in Low-Earth Orbit," *Proc. 4th European Conf. on Radiation and Its Effects on Components and Systems*, pp. 251-8, Palm Beach, Cannes, France, Sep. 1997.

[Underwood 98] Underwood, C.I., "The Single-Event-Effect Behavior of Commercial-Off-The-Shelf Memory Devices – A Decade in Low-Earth Orbit," *IEEE Trans. Nucl. Sci.*, Vol. 45, No. 6, pp. 1450-1457, Dec. 1998.

[Velazco 92] Velazco, R., et al., "SEU Testing of 32-Bit Microprocessors," *IEEE Radiation Effects Workshop*, pp. 16-20, 1992.

[Vergos 95] Vergos, H.T., and D. Nikolos, "Performance Recovery in Direct-Mapped Faulty Caches via the Use of a Very Small Fully Associative Spare Cache," *Proc. Int'l Comp. Performance and Dependability Symp.*, pp. 326-332, April 1995.

[Wagner 88] Wagner, et al., "Alpha-, Boron-, Silicon-, and Iron-Ion-Induced Current Transients in Low-Capacitance Silicon and GaAs Diodes," *IEEE Trans. on Nuclear Science*, Vol. 35, No. 6, pp. 1578-84, Dec. 1988.

[Wakerly 00] Wakerly, J.F., *Digital Design Principles & Practices*, 3rd ed., Prentice Hall, 2000.

[Wang 99] Wang, J.J., et al., "SRAM Based Re-programmable FPGA for Space Applications," *IEEE Trans. on Nuclear Science*, Vol. 46, No. 6, pp. 1728-1735, Dec. 1999.

[Whelan 77] Whelan, J.W., "Error Correction with a Microprocessor," *Proc. IEEE National Aerospace and Electronics Conf.*, pp. 1317-23, 1977.

[Whiting 75] Whiting, J.S., "An Efficient Software Method for Implementing Polynomial Error Detection Codes," *Computer Design*, Vol. 14, No. 3, pp. 73-7, Mar. 1975.

[Wicker 95] Wicker, S.B., *Error Control Systems for Digital Communications and Storage*, Englewood Cliffs, N.J., Prentice Hall, 1995.

[Winokur 99] Winokur, P.S., et al., "Use of COTS Microelectronics in Radiation Environments," *IEEE Trans. Nucl. Sci.*, Vol. 46, No. 6, pp. 1494-1503, Dec. 1999.

[Wood 94] Wood, K.S., et al., "The USA Experiment on the ARGOS Satellite: A Low Cost Instrument for Timing X-Ray Binaries," Published in *EUV, X-Ray, and Gamma-Ray Instrumentation for Astronomy V*, ed. O.H. Siegmund & J.V. Vellerga, SPIE Proc., Vol. 2280, pp. 19-30, 1994.

[Worley 90] Worley, E., R. Williams, A. Waskiewicz, and J. Groninger, "Experimental and Simulation Study of the Effects of Cosmic Particles on CMOS/SOS RAMs," *IEEE Trans. on Nuclear Science*, Vol. 37, No. 6, pp. 1855-1860, Dec. 1990.

[Yang 95] Yang, G.-C., et al., "Reliability of Semiconductor RAMs with Soft-Error Scrubbing Techniques," *IEE Proc. – Computers and Digital Techniques*, Vol. 142, No. 5, pp. 337-44, Sept. 1995.

[Youngs 96] Youngs, L., G. Billus, A. Jones, and S. Paramanandam, "Design of the UltraSPARC™-I Microprocessor for Manufacturing Performance," *Proc. of the SPIE*, Vol. 2874, pp. 179-186, 1996.

[Zhang 98] Zhang, K., et al., "Methods for Reducing Soft Errors in Deep Submicron Integrated Circuits," *IEEE 5th Intr'l Conf. Solid-State and Integrated Circuit Technology*, pp. 516-519, Beijing, China, Oct. 1998.

[Ziegler 96a] Ziegler, J.F., et al., "(all articles)," *IBM J. Res. Develop.*, Vol. 40, No. 1, Jan. 1996.

[Ziegler 96b] Ziegler, J.F., et al., "IBM Experiments in Soft Fails in Computer Electronics (1978-1994)," *IBM J. Res. Develop.*, Vol. 40, No. 1, pp. 4, Jan. 1996.

[Ziegler 98] Ziegler, J.F., "Terrestrial Cosmic Ray Intensities," *IBM J. Res. Develop.*, Vol. 42, No. 1, pp. 117-139, Jan. 1998.