SEU Tolerant Device, Circuit and Processor Design

William Heidergott
General Dynamics C4 Systems
8201 E. McDowell Road
Scottsdale, Arizona, USA
1.480.441.4598
bill.heidergott@gdc4s.com

ABSTRACT

Development of highly reliable and available systems requires consideration of the occurrence of single event upsets, the effects they have on system performance, and strategies for their prevention and mitigation. Methods of systems engineering process and the application and validation of techniques for fault tolerance are discussed as elements in the elimination and mitigation of single event upsets.

Categories and Subject Descriptors
B.8.1 [Performance and Reliability]: Reliability, Testing, and Fault-Tolerance

General Terms
Reliability

Keywords
Radiation effects, single event upset, soft error rate, fault tolerant systems, error detection and correction coding, fault avoidance, fault masking, modular redundancy, temporal redundancy

1. INTRODUCTION

The development of systems requiring high reliability and availably requires knowledge of the occurrence of Single Event Upsets (SEU), understanding of their effects, and the use of techniques to eliminate or to otherwise mitigate their impact on system operation. The existence of effects due to single energetic particle interaction with microelectronic device structures has been known and addressed since the mid-1970's. In 1975 Intel described α-particle induced Soft Error Rate (SER), space programs began reporting on-orbit occurrence of single event upsets; IBM reports the use of Error Correction Coding (ECC) in computers dating back to the mid-1960s. Focus in the 1980's was on the use of ECC to mask the occurrence of SER in SDRAM devices; work on fabrication technology modifications to reduce SER in memories and on effects in microprocessor device memory was performed in the 1990's. Developing an understanding of the occurrence, detection, and recovery techniques for logic and other elements of processor systems has been performed recently.[1] Due to the significantly higher level of energetic particle severity in the space environment, most of the work on the general class of single event effects has occurred in support of space applications. Single event upset is only one element of a broader set of single event effects that result from energetic particle interaction with the device structure. The term Soft Error Rate was established by terrestrial systems developers, whereas the space systems community uses Single Event Upset to describe the same effects.

2. ENERGETIC PARTICLE ENVIRONMENTS

Single event upset environments may be separated into the space environment (earth orbiting and interplanetary traverse), and terrestrial applications, including high altitude avionics systems. Most severe amongst these applications is that of space systems; single event upsets in space result from the traverse of cosmic rays of galactic and solar origin, trapped protons, and solar event particles. Severity of the space environment varies significantly over various orbit conditions (altitude and inclination) and exhibits profound spatial variation for various segments of orbit traverse and temporal variation due to both solar and the earth geomagnetic field conditions. Clementine, a short duration moon mapping mission, experienced 71 errors per day in a 2.1 Gbit image memory, the use of error detection and correction fault masking provisions resulted in zero errors in 1.5 million stored images due to complete mitigation of these upsets. Cassini, a long duration mission to explore Saturn, experienced 280 errors per day in 2.5 Gbit memory.[2] The European Space Agency Freja satellite experienced >200 SEUs/ day, and upset rates as high as 32 SEUs/ minute have been observed in a 2- Gbit DRAM solid-state data recorder on SOHO during intense solar particle events.[3]

Terrestrial environments include high energy and thermal neutrons and α-particles emitted by the decay of radioactive impurities (^{232}Th, ^{238}U, ^{210}Po, etc.) found in materials used in device manufacture and packaging. Terrestrial neutrons are produced by the cascade of particles that result from cosmic ray interaction with oxygen and nitrogen atoms in the earth's upper atmosphere. Due to absorption in the atmosphere, the environment peaks at an altitude of approximately 60,000 feet, decreasing by over two orders of magnitude down to sea level.[4] The population of cosmic rays reaching the upper atmosphere varies with latitude due to the shielding provided by the earth's geomagnetic field; increasing by a factor of six between equatorial latitude and the high latitude polar regions. Short-term changes (daily extremes) in the neutron environment also result from earth geomagnetic storms associated with solar particle (flare) events. Additionally, the cosmic ray flux exhibits long-term dependence on the 11-year solar cycle, in which the resultant flux of terrestrial neutrons reduces by approximately 30% during the solar maximum phase.[5] Although the terrestrial environment changes are not as severe as those of space applications, there exists considerable spatial and some temporal variation in the environment severity.

3. SINGLE PARTICLE INTERACTION WITH DEVICE STRUCTURE

The interaction of energetic particles with microelectronic device structure results in direct ionization or secondary particles that traverse the device leaving a plasma region of electron-hole pairs in its wake. Dimension of the track may be very small to greater than several microns depending on the energy of the incident particle. The density of electro-hole pairs in the plasma region varies depending on the Linear Energy Transfer (LET) of the incident particle. LET is related to energy loss per unit path length, is a function of the ion species and particle energy; and, using a conversion factor equates to charge produced per unit traveled or fC/μ . Cosmic ions, α-particles, and protons result in direct ionization; neutrons and protons interact with nuclei to produce secondary particles that produce an ionization track.

Charge collected by drift and diffusion results in several possible circuit level effects in devices. Deposited charge can activate parasitic elements in the device structure; semiconductor latchup, dielectric rupture, and other potentially destructive effects are known to occur. Severe effects such as these occur mostly due to higher charge producing particles found in the space environment. Of interest to this work is that of collected charge producing a Single Event Transient (SET) pulse, the circuit response to which may be a Single Event Upset (SEU). Some SEU occurrence results in a more pervasive and complex effect on devices known as Single Event Functional Interrupt (SEFI).

4. FAULT TOLERANT SYSTEMS

4.1 System Reliability and Availability

The two most common expressions of a system's ability to tolerate failure are reliability and availability. Reliability, *R(t)*, is the conditional probability that a system will remain operational (or can be restored to an operational state) over the anticipated life of the system. Availability, *A*, is defined as the probability that a system is accessible at any particular time, and expresses the fraction of time a system is operational.

$R(t) = e\text{-}\lambda t \quad | \quad \lambda = 1 / MTBF$

$A = \mu / (\lambda + \mu) \;|\; \mu = 1 / MTTR$

MTBF = Mean Time Between Failure
MTTR = Mean Time To Repair/Recover

A system with high reliability may exhibit unacceptable availability if the mean-time-to-repair/recover is unacceptably long. A system with high availability may in fact fail; if the mean-time-to-repair/recover is suitably low, acceptable availability may be achieved. Although availability is expressed as the percentage of the time the system is available to users, specifying the requirement is typically based on outage frequency, outage duration, and ultimately on the implications of users being denied access to the system. The presence of degraded modes or conditions that infer partial or limited availability further complicate the specification and design activity. Departures from expected system operation that contribute to unavailability are allocated to numerous sources, single event upsets receive only a portion of the allowable duration of unavailability. Availability is often expressed as a steady-state value, either as the probability that the system is operational at any random time (P(s)=0.99999, 5-nines), or as a given amount of downtime over a specified interval. Statistical mean values of system failure rate and recovery times are often used in system analysis; however, the spatial and temporal variances in the energetic particle environments that produce single event upsets need to be understood in addressing availability.

4.2 Introduction to Fault Tolerant Systems Techniques

The risk of a single event upset causing operational impact to a system establishes the need for an adequate description of component SEU characteristics and setting design requirements for single event effects management and mitigation. A technique such as Single Event Effects Criticality Analysis may be used to assess system level SEU criticality based on functional impact to the system.[6] The evaluation of alternative single event effects mitigation and fault tolerance provisions should address cost, performance, reliability, and single event effects mitigation effectiveness. System requirements and subsystem specifications must address allocated availability requirements, operational modes, and external support capability that relate to the fault tolerance characteristics of the system. Support provisions from external agents such as periodic or on-demand operational intervention in support of diagnosis and recovery actions must be defined, and the acceptability of different modes of operation including reduced capabilities and degraded levels of service, safe shutdown, and emergency modes of operation must be considered. System architecture and partitioning defines error containment boundaries, establishes the system-wide strategy for coordinated error detection and recovery, and the redundancy provisions that support fault tolerance are determined. Establishing a fault tolerance hierarchy defines detection and recovery functions for individual local subsystems, shared functions over a group of subsystems, and system-wide or global functions.

Historically, the general subject of fault tolerant systems considers numerous causes of faults, including incorrect specifications, design errors, undetected manufacturing defects, human operator actions, component damage or failure, and interaction with the operating environment. The language of fault tolerant systems identifies a *fault* as the initiating event (in this case an energetic particle induced single event transient or resultant upset); *errors* are identified as the undesired system states caused by the fault. If error detection and recovery do not take place in a timely manner, a *failure* can occur that will be manifested by the inability of the system to provide specified service. *Fault tolerance* is the capability of a system to recover from a fault or error without exhibiting failure. A fault in a system does not necessarily result in an error; a fault may be latent in that it exists but has not resulted in an error; the fault must be sensitized by a particular system state and input conditions to produce an error. The techniques of fault tolerant systems include *fault avoidance*, *fault masking*, *detection* of erroneous or compromised system operation, *containment* of error propagation, and *recovery* to normal system operations. The ability to detect at the lowest level possible, with error containment within the minimal number of components and subsystems, enables simpler and more effective recovery provisions.[7,8]

Fault tolerant subsystem design includes error detection and recovery provisions for faults within the subsystem as well as support for interfaces with other subsystems and global detection and recovery provisions. Detection and recovery may be either concurrent with normal system operation or preemptive in which case normal system operation is suspended until completion of the recovery activities. Correction provisions may be real-time; recovery provisions may be either backward (rollback to a previous error-free state) or forward, constructing a valid error free new state from existing protected information. The recovery sequence includes the identification and removal of errors, restoration of valid subsystem states, and validation of the completion of successful recovery operation. It must be demonstrated that adequate testability is provided to detection and correction/recovery provisions, ensuring that manufacturing or component defects do not exist in systems which compromise the operation or performance of fault tolerance provisions. Furthermore, circuitry added to enable such testability must be examined for the propensity to introduce new fault conditions or to exacerbate the rate of fault occurrence.[9]

Validation and verification activities must include the ability to establish that the design supports the requirements established for fault tolerance. Methods and provisions utilized in subsystem evaluation and for various levels of system integration verify the performance of detection, correction, and recovery provisions. Qualitative evaluation of fault tolerant design provisions includes the use of formal and heuristic methods, test, and experimentation to demonstrate operability of detection and recovery provisions.

4.3 Fault Avoidance

Fault avoidance addresses techniques to reduce the occurrence of faults through reduction in the severity of the energetic particle environment or the mitigation of circuit response to the traverse of energetic particles. Reduction in the severity of the environment includes consideration of alternative applications conditions, such as orbit altitude and inclination in space systems and latitude and altitude of atmospheric neutrons, and the use of shielding provisions to reduce the severity of the environment when translated to sensitive component locations. Reduction in the severity of the alpha particle environment can be achieved through screening and selection of materials used in device manufacture; including low alpha emitting solder and plastic encapsulant materials. Due to the limited energy and range of alpha particles emitted by plastic encapsulatnts, the use of shielding provisions such as polyimide material deposited onto the die surface can result in significant reduction in the population of alpha particles reaching sensitive device structures. Shielding materials having large thermal neutron capture cross-section are effective in attenuating the population of thermal neutrons. Due to their energy and resultant range, shielding is much less effective in attenuating the high-energy atmospheric neutrons, trapped and solar protons, or cosmic ray heavy ion environments.

Fault avoidance provisions also address the mitigation of single event upset through reduction in charge generation and collection, and mitigation of circuit response to collected charge. Techniques directed at reducing collected charge typically rely on the ability to truncate the funnel dimensions, to reduce the carrier lifetime in the region of the sensitive volume. Manufacturers have demonstrated success in the reduction of alpha particle induced soft error rate through the use of retrograde well profile; the increasing carrier concentration with increased distance into the well region results in reduced carrier lifetime, thus reducing the charge collected from the region below the sensitive volume.[10]

Mitigation of circuit response includes provisions to increase critical charge, suppress response by limiting circuit bandwidth such that no response is generated by the short collected charge transient pulse, and to provide circuit design provisions blocking the propagation of pulses in memory cell feedback paths. Memory manufacturers have utilized stacked capacitor structures to increase cell capacitance, single event upset hardened devices have used high valued polysilicon resistors in the feedback path of the SRAM cell, working with the cell capacitance prevents the cell from responding to the short duration of charge collection. Numerous unique and novel circuit designs have been developed for memory cells, latches, and registers; based on techniques to reduce the bandwidth of the cell to achieve immunity to the transient caused by collected charge or to provide redundant storage or blocking provision to prevent upset.[11-14] Although effective in improving cell single event upset characteristics, the increased cell size and the reduced bandwidth are contrary to the achievement of high-speed operation.

4.4 Fault Masking and Redundancy

Implementation of fault masking typically utilizes some form of redundancy; variations include *informational redundancy* (redundant data structures), *spatial redundancy* (redundant hardware), and *temporal redundancy* (redundant sequential operations). The fault detection and correction capabilities of redundant structures are good; recovery complexity depends upon the nature of the function performed by each of the structures.

The increase in hardware content in informational redundancy is typically less than spatial modular redundancy. *Information redundancy* and encoding provisions are designed such that in the presence of errors, the information is internally inconsistent, enabling Error Detection And Correction (EDAC) implementations. EDAC codes can vary widely in detection and correction capabilities, code efficiency, and complexity of encoding and decoding circuitry. Cyclic redundancy checks, other cyclic codes, and convolutional coding schemes are used to detect errors in serial data transfer interfaces and storage media; such codes are generally classified as either block codes or convolutional codes. Hamming codes are the first class of linear block codes that were devised for error correction; these codes and their variations have been widely used in digital communications and data storage systems. The most common block code used in memory systems is the single-error-correcting, double-error-detecting (SEC-DED) Hamming code; for a code to be useful for high-speed memory applications, its structure must permit rapid parallel encoding and decoding operations, Hsiao developed SEC-DEC codes by examining the properties of shortened Hamming codes to identify specific codes that enable fast encoding and decoding processes. The Bose, Chaudhuri, and Hocquenghem (BCH) codes form a large class of powerful error correcting cyclic codes the most commonly recognized being the non-binary Reed-Solomon (RS) codes.[15-17]

The nature of coding precludes its use in most computational operations. Simple arithmetic functions permit the use of parity through multistage adder circuits, and arithmetic codes such as AN and residue codes exist. AN is a non-separate code in which the added information is not distinct from the initial information, permitting fault detection in addition operations. Other residue codes are separate in nature; inverse residue codes are capable of detecting multiple errors that are unidirectional in nature, and bi-residue codes with two separate encoding levels provide error correction capability. The use of residue coding schemes for fault tolerance has received very limited application due to the high overhead requirement for pre-multiplying input variables, limited correction capabilities, and lack of capability for general computational operations.[18]

In the most common form of *spatial or modular redundancy* is Triple Modular Redundancy (TMR), a fault in an individual module is corrected by the action of a voter through the majority consensus, or two-out-of-three, voting rules. The individual modules may consist of basic functions such as memory elements, D-flip-flops, or latches, and more complex logic blocks such as state machines or highly complex logic functions; including complete processing functional blocks, or entire subsystems.[19]

The notion of *modular redundancy* can be extended to the general sense of n-modular redundancy (nMR) with associated enhancements to the fault tolerance capabilities and expense associated with their implementation. The capability to change the nature of a modular voting scheme is termed dynamic redundancy. In general, dynamic redundancy consists of three actions, detection of error, location of the error, and reconfiguration of the structure to perform in an optimum manner. If there exists a highly probable condition in which failures in both modules would produce identical module output conditions, the redundant module could potentially contain a complementary function instead of an identical copy of the primary module. In this case the dual of the primary module is utilized in the design of the redundant module. An element of difficulty in complex circuits using spatial redundancy is the ability to recover the faulty processor to a known-good condition so that its outputs may again be utilized in detection and correction operations. Often the recovery operation in the faulty processor requires recovery of data and control states and resynchronization processes that cannot be performed concurrent with normal operations.

An alternative to spatial redundancy is *temporal redundancy* in which the same hardware or software elements are used in consecutive operations, using diversity in time to provide results which may be compared using similar comparator/detection, and correction techniques discussed above for spatial redundancy. The technique is dependent upon the requirement for known error-free conditions on the input signals or variables used in the operations; either independent detection and correction must be provided to inputs, or the use of alternating logic techniques or complementing functions must be utilized to provide tolerance to input signal or variable errors. Alternating logic utilizes a class of boolean functions which are self dual, i.e. they satisfy the property f(x1_, x2_, . . ., xn_) = f_(x1, x2, . . ., xn). Recomputing with shifted operands (RESO) is applicable to certain problems in which the shifting of inputs forms a complementing function that produces known relationship in outputs that may be utilized in detection and correction.

In older technologies single event transient pulses would not propagate along a path of combinatorial logic gates and the limited clocking frequency reduced the probability of sampling an incorrect logic state during the presence of a transient. The bandwidth of advanced technologies supports propagation of such pulses, and high clock frequencies provide ample opportunity for clocking of errant results into memory elements. The use of static spatial redundancy as a fault tolerance technique for single event upsets in memory elements (latches, flip-flops, and RAM), along with temporal redundancy as a mitigation provision against single event transients in combinatorial logic has been proposed for advanced technology devices.[20] The combined scheme uses temporal triple mode redundancy by sampling the output of combinatorial logic at three different times, storing the individual result in three different latch or flip-flop elements, and majority voting the latched result. By ensuring that the separation between clock edges is greater than the duration of a transient pulse emerging from the combinatorial logic, the transient can corrupt only one of the three values. Majority voting of static triple spatial redundancy ensures that upset of individual latches will also be corrected through action of the majority vote logic.

4.5 Error Detection, Containment and Recovery

Whereas the fault detection capabilities of hardware based schemes such as modular redundancy are good, fault detection is the most difficult aspect of fault tolerance to achieve in applications oriented fault tolerance using algorithms, or software based fault detection, due to the elusive nature of the signature of most faults. A fault that results in a check-stop result enables obvious opportunity for detection; conditions of compromised system state produce signatures that are difficult to discern. Once detection is accomplished, recovery may be as complete and efficient as necessary to restore normal system operation.[21]

The ability to apply software fault tolerance techniques relies on specific properties of the problems being addressed by processor operations and are embodied in a *constraint predicate*. Three defined predicate subclasses formulate the constraint predicate: *progress*, *feasibility*, and *consistency*. The *progress predicate* utilizes the notion that there exist steps in the sequence of process operations and that the decomposition of the process into a finite number of operations blocks provides opportunity for testability at these intermediate points in the process. The *feasibility predicate* implies constraints that are apparent from the nature of the problem on which the processor is operating. Contained within this notion is the property of boundary conditions as constraints from which to generate a good predicate; testable results must be within the solution space of the problem as defined by the boundary conditions. *Consistency* conditions imply the ability to infer validity in intermediate or final results from input variables and previous intermediate or final process results. Consistency tests are a powerful technique; entire constraint predicates have been developed using only consistency conditions.

Application-oriented fault tolerance is a software approach to the problem of error detection and recovery; the software components are called assertions. The application of a set of metrics to a problem specification results in software assertions that may be embedded in the program code; assertions having the form "if not ASSERTION then ERROR". The extent of error detection is determined by the perceptiveness of the assertions to discern a compromised state of program execution. The recovery capability is determined by the response that is embodied in the error branch path. Applications-oriented fault tolerance works on the principle that testing of a program's intermediate results for conformance to specification ensures that the end result will be within specification, and if an error does not manifest itself in a failure, then the fault is of no consequence or interest.[22]

Several acceptance test techniques employed in detecting the presence of faults are *N-version programming*, *self-checking software*, *recovery blocks*, and the *watchdog coprocessor*. *N-version programming* is intended as a provision to detect defects in software design, coding, and integration. Parallel or sequential execution of programs and comparing the results provides opportunity for fault detection, much in the same way as modular redundancy does for hardware fault detection provisions.[23]

The *self-checking software* technique has aspects of functionality, control flow, and data in order to provide error detection. The functional and data aspects examine the reasonableness of the results, and may include an assessment of the input variables in performing acceptance testing of algorithm results. The capability provided by seemingly ad-hoc techniques in assessing the reasonableness of algorithm results is a powerful technique in the detection of computation errors. The control aspect includes checks on the execution flow from entry point to exit points in algorithm blocks, and only valid paths between the algorithm blocks are permitted. One approach to establishing the correctness of high-level control flow uses structure labeling that is embedded in the syntax of the program text. The introduction of path tags to check the validity of the sequencing of blocks and block-tags to verify that execution of blocks proceeded properly from entry to exit points are examples of such structure labels. Each block contains a unique signature, and upon entry the block-tag is set to the value of the signature. The block-tag is verified upon exit from the routine to confirm that the block was not entered in any manner except the valid entry point. The path-tag is then set to the value of the next block signature, which is checked on entry into all blocks. Similar control checking of program iterative execution loops for illegal entry, completion, and branch related loop termination is utilized.

The *recovery block* approach uses acceptance tests in the form of checksums and other bounds checking on computation results to detect the presence of errors. The recovery block notion in this provision is that subsequent executions of the process using the same or different algorithms provide a recovery path from which acceptable results may be generated. The recovery block is a language construct supporting the incorporation of program redundancy into a fault tolerant program. The syntax of recovery block incorporation may take the form: ensure T by B1 else by B2, . . . else by Bn else error. T denotes the acceptance test, B1 denotes the primary try block, and Bn denotes the alternate try blocks. The T acceptance test is a logical expression representing the criterion for determining the acceptability of the execution results of the try blocks.[24]

Another attempt at monitoring the behavior of a system is the use of a *watchdog coprocessor*. Most embedded processor systems use a hardware watchdog timer to detect halts to processor execution or errors in program control flow that are detected through failure of the software execution to reset the timer within a prescribed period of time. The watchdog coprocessor extends the notion from the simple hardware timer to the use of an additional processor to check the results of primary on-line processor elements. An active watchdog may implement interaction with the on-line processor as simple as an "I'm OK" message or heartbeat monitor that the watchdog expects to receive within a prescribed window. If not received within the valid window, the watchdog interprets this result that the on-line processor control flow is disrupted and asserts interrupt or reset of the on-line processor. The watchdog may execute concurrently with the on-line processor, or it may operate off-line and pre-compute results for subsequent acceptance testing. It invokes decisions on the integrity of the system based on assertions about the main process, assuming that faults either disrupt program control flow, corrupt database contents, or produce incorrect numerical results.

One specialized application of information redundancy to computational problems is Algorithm Based Fault Tolerance (ABFT). In this approach, some attribute of the function being performed is exploited with the use of information or time redundancy to achieve error detection, correction, or recovery. Most ABFT techniques developed to date address computational problems that exhibit structure and regularity which can be exploited to develop informational redundancy, such as matrix computations, sorting, Fast Fourier Transforms, QR factorization, singular value decomposition, least squares minimization, and other signal processing applications.[25]

To minimize the impact to system operations and the extent of recovery operations, and to enhance the probability of successful system recovery, errors must be confined to the module or subsystem in which the fault occurred. Typically, error containment boundaries are hierarchically defined, with errors confined at the lowest level possible. Containment boundaries can be established by subsystems checking their own outputs, or by validating all input information. If error detection is activated but error recovery is not supported, the subsystem process is typically halted to prevent error propagation.

If an unmasked fault has propagated in a system, a recovery period is needed to correct the system. Most recovery schemes restore system operation to a previous correct state or recovery point. A processor is rolled back to a recovery point by restoring the processor state and key variables to a known good condition, invalidating cache memory (which is likely to have been corrupted by error propagation) and forcing cache data to be restored from protected main memory. A checkpoint is a copy of an application's state that is stored in a protected region of the system. When a failure is detected, the application's state is rolled back to the saved previous checkpoint and execution resumed at that point. Backward error recovery can be defined as the capability of a system to return to a consistent state that existed before it failed; a checkpoint is then defined as a consistent state from which the execution can be restarted.[18]

4.6 Validation of Fault Tolerance

Several approaches have been utilized to assess the effects of faults in processor systems and for validating fault handling mechanisms, including analytical modeling, experimental techniques (hardware pin faults, memory corruption, and ion irradiation), simulation modeling (register transfer level, gate level, and op-code level simulation) and fault emulation (memory, bus, and register transfer level). Use of analytical modeling is problematic due to the very large nature of the problem, and the simplifying assumptions that make the analysis tractable are regarded as compromising the usefulness of the technique and the validity of the results. Experimental techniques involve monitoring the behavior of a system while faults are introduced and recording the error detection and correction/recovery performance of the system. This approach requires the use of an accelerating agent to produce sufficient faults to gather statistically valid measurements on the fault tolerant performance of the system. Simulation based methods provide the capability to establish the time of occurrence, location and type of fault, and the transient or permanent characteristics of the fault. Physical injection of faults utilizes charged particles, such as heavy-ion fission fragments from a Californium-252 source, or the use of a heavy ion or proton accelerator to produce single event faults. In addition to the ability to quantify fault tolerance, experimental techniques provide the capability to study the consequences of faults that escape detection or are not afforded complete correction or recovery. Complexity of most systems results in the need to perform many experiments to achieve statistical confidence in the validation result These results are utilized in analysis of system performance, often using techniques such as Markov process modeling, to make assertions regarding the fault tolerant performance and availability of the system.

5. REFERENCES

[1] Bossen, D., *CMOS Soft Errors and Server Design*, Radiation Induced Soft Errors in Silicon Components and Computer Systems Tutorial, *IRPS 2002*.

[2] Johnston, A., *Mitigation Methods for Soft Errors and Related Radiation Effects in Spacecraft*, Radiation Induced Soft Errors in Silicon Components and Computer Systems Tutorial, *IRPS 2002*.

[3] Dodd, P. and Sexton, F., *Mitigation of Single- Event Effects in Mission- Critical Systems*, Radiation Induced Soft Errors in Silicon Components and Computer Systems Tutorial, *IRPS 2002*.

[4] Normand, E., *Single-Event Effects in Avionics*, IEEE Transactions on Nuclear Science, Vol. 43, No. 2, April 1996, Page 461-474

[5] National Geophysical Data Center (NGDC), National Oceanic and Atmospheric Administration (NOAA), http://www.ngds.noaa.gov/stp/SOLAR/COSMIC_RAYS/cosmic.html

[6] LaBel, K., Michele Gates, M., Barth, J., Stassinopoulos, E.G., Johnston, A. and Marshall, P., *Single Event Criticality Analysis (SEECA)*, http://flick.gsfc.nasa.gov/radhome/papers

[7] Somani, A.K. and Viadya, N.H., *Understanding Fault Tolerance and Reliability*, Computer, pp 45-50, (Apr 1997).

[8] Avizienis, A., *Toward Systematic Design of Fault-Tolerant Systems*, Computer, pp 51-58, (April 1997)

[9] Nelson, V., *Fault-Tolerant Computing: Fundamental Concepts*, Computer, pp 19-25, (July 1990)

[10] *Mitsubishi Electric Develops High-Frequency Synchronous SRAM with Dramatically Reduced Soft Error Rate*, International Solid-State Circuits Conference, Feb 1999, http://www.mitsubishichips.com/press/releases/fsram_99.htm

[11] Vinson, J, *Circuit Reliability of Memory Cells with SEU Protection*, IEEE Transactions on Nuclear Science, Vol. 39, No. 6, pp 1671-1678, (December 1992)

[12] Rockett, L., *An SEU Hardened CMOS Data Latch Design*, IEEE Transactions on Nuclear Science, Vol. 35, No. 6, pp 1682-1687, (December 1992)

[13] Calin, T., Nicolaidis, M., Velazco, R., *Upset Hardened Memory Design for Submicron CMOS Technology*, IEEE Transactions on Nuclear Science, Vol. 43, No. 6, pp 2874-2878, (December 1996)

[14] Norely, M., Liu, and Whitaker, S., *Low Power SEU Immune CMOS Memory Circuits*, IEEE Transactions on Nuclear Science, Vol. 39, No. 6, pp 1679-1684, (December 1992)

[15] Lin, S. and Costello, D., *Error Control Coding – Fundamentals and Applications*, Prentice-Hall, (1983)

[16] Fujiwara, E. and Pradhan, D., *Error-Control Coding in Computers*, Computer, (July 1990)

[17] Chen, I. and Yen, I., *Analysis of Probablistic Error Checking Procedures on Storage Systems*, The Computer Journal, Vol. 38, No. 5, (1995)

[18] Peercy, M. and Banerjee, P., *Fault Tolerant VSLI Systems*, Proceedings of the IEEE, pp 745-758, (1993)

[19] Vaidya, N.H., *Comparison of Duplex and Triplex Memory Reliability*, IEEE Transactions on Computers, Vol. 45, No. 4, pp 503-507, (1996)

[20] Mavis, D. and Eaton, P., *Temporally Redundant Latch for Preventing Single Event Disruptions in Sequential Integrated Circuits*, Mission Research Corporation Technical Report P8111.29, October 1998

[21] Sosnowski, J., *Transient Fault Tolerance in Digital Systems*, IEEE Micro, (1994)

[22] McMillin, B., *Fault Tolerance for Multicomputers: The Application Oriented Paradigm*, Department of Computer Science, University of Missouri-Rolla, (1997)

[23] Banatre, M. and Lee, P., *Hardware and Software Architectures for Fault Tolerance, Experiences and Perspectives*

[24] Kim, K.H., and Welch, H.O., *Distributed Execution of Recovery Blocks: An Approach for Uniform Treatment of Hardware and Software Faults in Real-Time Applications*, Fault-Tolerance Systems: Techniques and Applications, edited by Hoang Phan, pp 95-105, (1992)

[25] Blaquiere, Y., Gagne, G., Savaria, Y., and Evequoz, C., *A New Efficient Algorithm-Based SEU Tolerant System Architecture*, IEEE Transactions on Nuclear Science, Vol. 42, No. 6, (Dec 1995)