RADIATION-INDUCED ENERGY DEPOSITION AND SINGLE EVENT UPSET ERROR RATES IN SCALED MICROELECTRONIC STRUCTURES

By

Christina L. Howe

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

December, 2005

Nashville, Tennessee

Approved:

Professor Robert A. Weller

Professor Robert A. Reed

ACKNOWLEDGEMENTS

Many people have played a significant role in my success as a graduate student at Vanderbilt University. I would like to first thank my advisor, Dr. Robert Weller, for his support and advice both professionally and personally while completing my masters work. Also, thank you to Dr. Robert Reed, Dr. Ronald Schrimpf, Dr. Marcus Mendenhall, Kevin Warren, and Scooter Ball for their direct support and contributions to this work. Thank you to Dr. Dan Fleetwood and Dr. Lloyd Massengill for technical discussions and support during this work. Thank you to all my professors at Vanderbilt University for their guidance in my academic career. I would also like to thank my friends in the Radiation Effects and Reliability Group for their help and support throughout my graduate school career. I would also like to thank my undergraduate advisor, Dr. Dick Blandford at the University of Evansville, Indiana, for giving me the encouragement and drive to attend graduate school.

Finally, thank you to my husband, Josh Howe, for his never ending support, patience, and encouragement through all of my schooling. I also want to thank my parents Wayne and Sue Tincher, Tim, Cheryl, Lance, Angie, Madeline, Grandma Long, Grandpa Wayne, Grandma Jackie, Jeff, Sandy, Jake, and Daniel for their love and support in whatever I do. None of this work would have been possible without each one of them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Advances in microelectronic technologies and economic pressure to use commercial electronic parts for space flight applications have created a new situation in which time-tested methodologies for radiation-hard electronic design, validation, and verification can no longer assure the safety of electronic parts and systems used for space exploration. In particular, there is a specific need to revisit the test methods and models used in predicting on-orbit radiation response of modern electronics. Recent single event upset (SEU) radiation effects experiments on modern technologies show trends inconsistent with current models (e.g., advanced CMOS [1] and SOI/SOS CMOS [2]).

In general, the existing on-orbit SEU models have the following shortcomings:

- They exclude combined effects from direct and indirect ionization by incident particles,
- They do not account for the angular dependence of the reaction products resulting from interactions other than electronic stopping,
- They exclude charge collection by diffusion,
- They have limited capability to analyze detailed geometrical effects, i.e., edge effects, isolation trenches, and buried oxides,
- They have no method for modeling effects associated with the complex spatial variation of charge generated by individual ion strikes.

The existing techniques, developed circa 1980, fail to provide accurate survivability estimates for some modern technologies [3]. Mature technologies have been scaled to dimensions where new phenomena challenge some of the basic simplifying assumptions of radiation effects models, which were developed for technologies fabricated in the late 70s to early 80s.

The underlying mechanisms for SEU response are 1) ionizing radiation-induced energy deposition within the device, 2) initial electron-hole pair generation and recombination and 3) the response of the device and circuit to the electron-hole pair distribution. Each occurs on its own timescale and they are often assumed to be sequential, i.e., energy deposition determines initial electron-hole pair generation, which in turn impacts device and circuit response [4], [5]. While not a topic of this thesis, this assumption may fail for modern technologies [6]; these events are assumed to happen sequentially.

A better understanding of how radiation-induced energy deposition (and therefore charge generation) is distributed in and around the sensitive collection volumes of scaled microelectronic devices will lead to a more accurate understanding of complex ground-measurement responses and help to develop more accurate methods of using ground test results to predict the on-orbit response. One method of studying energy deposition is the use of simulation software to determine the expected radiation transport through materials and devices.

Monte Carlo techniques to predict radiation transport have been implemented for several decades (see [7], [8], and [9] for examples). Monte Carlo methods are computational algorithms based on the use of random numbers and probablity statistics

Table 1: Monte Carlo codes used for radiation transport through materials.

Monte Carlo Code	Incident Particles Available
BRIC [12]	protons
CUPID [13]	protons
EGS4/EGSnrc [14]	electrons, photons
MCBEND [15]	neutrons, gamma rays, electrons
MCNP [16]	neutrons, photons, electrons
MORSE [17]	neutrons, gamma rays
SEUSIM [18]	protons

for solutions to real systems. In general, Monte Carlo methods consist of probability density functions (pdfs), random number generators, sampling rules, tallying, error estimation, variance reduction techniques, and parallelization and vectorization [10]. Monte Carlo methods are used in a variety of fields and are useful for solving equations with large numbers of possibilities [11]. It is often compared to a game of chance, such as gambling, which results in the solution to a problem. Because of the method's ability to solve large problems, it is ideal for charge transport [11]. Many incident particles and secondary particles can be accurately accounted for easily.

Over the last couple of decades, Monte Carlo methods have been applied to predict energy deposition from complex nuclear reaction events. In general, however, these tools have been limited to a small number of incident particle types, typically ions with $Z \leq 1$. Table 1 presents several codes and the particles of interest within them.

Some other Monte Carlo codes have the capability to predict radiation transport from heavy ions ($Z > 1$), but are limited in other ways. For example, NASA Langley Research Center's High Charge Energy Transport (HZETRN) [19] code is designed as a black box for engineers looking for a quick answer to the expected amount of radiation that will pass through spacecraft shielding. For those wanting more specific answers to problems regarding physics and the mechanisms of transport, this code lacks the output of information [19]. A more recently developed code, the multi-purpose Particle and Heavy Ion Transport code System (PHITS) [20], has a restricted energy range for heavy ions of up to 3 GeV/nucleon max. A third code, Monte Carlo N-Particle eXtended (MCNPX) [16], is based off MCNP and will simulate 34 different particles types. The limitation of MCNPX is usability and flexability. Complex structures can be created, but the input files are difficult to master and use effectively.

The simulation tool used in this study is based on Geant4 [21] with extensions by Vanderbilt researchers [22]. Geant4 is a library of c++ routines assembled by an international collaboration for describing radiation interaction with matter [21]. It is used in many applications including high energy physics, medical, nuclear experiments, and space physics. It allows developers to easily create tools that correspond to their specific needs and applications. There are various phyics models that handle interactions of particles with matter across a broad energy range, up to 200 MeV/nucleon [23].

Chapter II describes the Monte Carlo tool used for this work and how it was used on a high performance computing cluster. This simulation tool has also been used in other recent work, see [1], [24], [25].

Chapter III of this thesis is an overview of the natural space environment and contains background necessary for a better understanding of the motivation for this thesis. A structure representative of a modern technology is shown in chapter IV and the data description is presented. In chapter V results of single-event upset (SEU) error rate calculations for two structures are presented showing that the classical method of SEU rate calculations underestimates the rate for certain structures. Detailed Monte Carlo simulations of energy depositions (charge generation) in a small volume are shown in chapter VI. The depositions are a result of interactions between the projectile ion (protons and heavy ions) and the structure. Simulations showing that it is necessary to include ionization, elastic and inelastic nuclear reactions, and screened Coulomb scattering when analyzing the impact that the heavy ion ($Z > 1$) space environment has on modern technologies is shown, a major departure from the classical view that typically only considers a simplified version of the ionization process. In chapter VII, charge generation in representative SOI volumes is shown, and finally chapter VIII concludes this thesis by a summary of the key points and results of this work.

CHAPTER II

SIMULATION TOOL

MRED - A Geant4 Application

The Monte Carlo code used to produce the results presented in this thesis is a Geant4 [21] application called MRED (Monte Carlo Radiative Energy Deposition) [1], [24], [25], [26]. Geant4 is a library of c++ routines assembled by an international collaboration for describing radiation interaction with matter [21]. It is used in many applications including high energy physics, medical, nuclear experiments, and space physics. Version 7.0.p01 of Geant4 was used to build the version of MRED used in this study. MRED is built on the Geant4 libraries with Vanderbilt additions, including a model for screened Coulomb scattering of ions [22], tetrahedral geometric objects [24], a biasing technique for cross section enhancement, and a number of additional features relevant to semiconductor device applications. The Geant4 libraries frequently contain alternative models for the same physical processes and these may differ in level of detail and accuracy. Generally, MRED is structured so that all physics relevant for radiation effects applications is available and selectable at run time. This includes electromagnetic and hadronic processes for all relevant particles, including elementary particles that live long enough to be tracked.

There are two models available for the description of the intra-nuclear cascade of nucleons produced by neutron and proton irradiation: the Bertini model, and a binary cascade alternative [23]. Generally, the Bertini model has been used in this work except as noted in chapter VII.

Heavy ion nuclear reactions are generally of less interest to the dominance of the Geant4 development and applications community and therefore ion-ion physics is less complete than that available for neutrons and protons. Nevertheless, a binary cascade model for light-ion reactions is available and recommended by its authors for projectiles up through ^{12}C. Recent comparisons to experimental results by T. Koi [27] have shown that the binary cascade model actually works fairly well for substantially heavier projectiles, and so it has been used here for projectiles up through ^{56}Fe with caution. Additional models for ion-ion collisions developed at Qinetiq [28] from original work by Wilson, et al. at NASA [29] are also available, but were not used in this work.

The model of electromagnetic interactions used for this work is the so-called standard model with the addition of screened Coulomb collisions. More detailed models are available [24], but they require substantially greater computing time and have not been observed to change these qualitative conclusions.

Using the MRED Application

The version of MRED used in this work, version 6, is structured such that an input file is created by the user stating all necessary commands and passed into MRED. This file contains information regarding the physics, structure, ion type and energy, and the desired output. An example input file can be viewed in Appendix A. Each line is documented so the reader may understand the commands. A typical input file points to other files for additional information. The files necessary for this are also documented in Appendix A. Note that future versions of MRED may not have the same type of input files as used in this work. The files presented here are for reference only.

Simulations for this work were conducted through Vanderbilt University's Advanced Computing Center for Research and Education (ACCRE). Through ACCRE's computing cluster, simulations using the MRED application were done in parallel, thereby allowing larger simulations to be performed in shorter amounts of time. A python script, developed by Dr. Marcus Mendenhall at The Vanderbilt University Free Electron Laser Lab, allows the MRED application to be duplicated to a specified number of copies and then each individual copy runs to completion independently. To run this script several files are necessary and can be found in Appendix B with a short description of what function each file performs.

The output of the version of MRED used in this work is a *Mathematica* [30] file containing the simulation data requested in the input file. In this work, *Mathematica* was used extensively for file compilation and data analysis.

CHAPTER III

BACKGROUND

Natural Space Environment

The natural space environment consists of particles that can contribute to upset or failure of electronic devices on satellites and other spacecraft. The number of particles and their typical energies vary greatly with altitude and solar activity. Most satellites operate in altitudes surrounding geosynchronous orbit (GEO) which is approximately 35,800 km above earth [31]. Particles trapped in the earths radiation belts include primarily electrons and protons, and some heavy ions ($Z > 1$). Cosmic rays, both solar and galactic, consist of protons and heavy ions. The galactic cosmic ray (GCR) environment typically consists of 85% protons, 14% alpha particles (helium), and 1% heavy ions [31]. While only 1% of the galactic cosmic rays are high energy heavy ions, they are very important when considering radiation effects in electronics because one hit by a heavy particle can do a great deal of damage within a device [31]. Spacecraft shielding can help block out lower energy particles, but can also create secondary particles which have the potential to be equally as damaging to electronics [32] or cause higher energy particles to slow down and become more ionizing [33]. Figure 1 shows the ion flux as a function of atomic mass up to nickel for the GCR environment. There are peaks at hydrogen (protons), helium (alphas), carbon, oxygen, nitrogen (the last three are known as the CNO environment), neon, silicon, and iron.

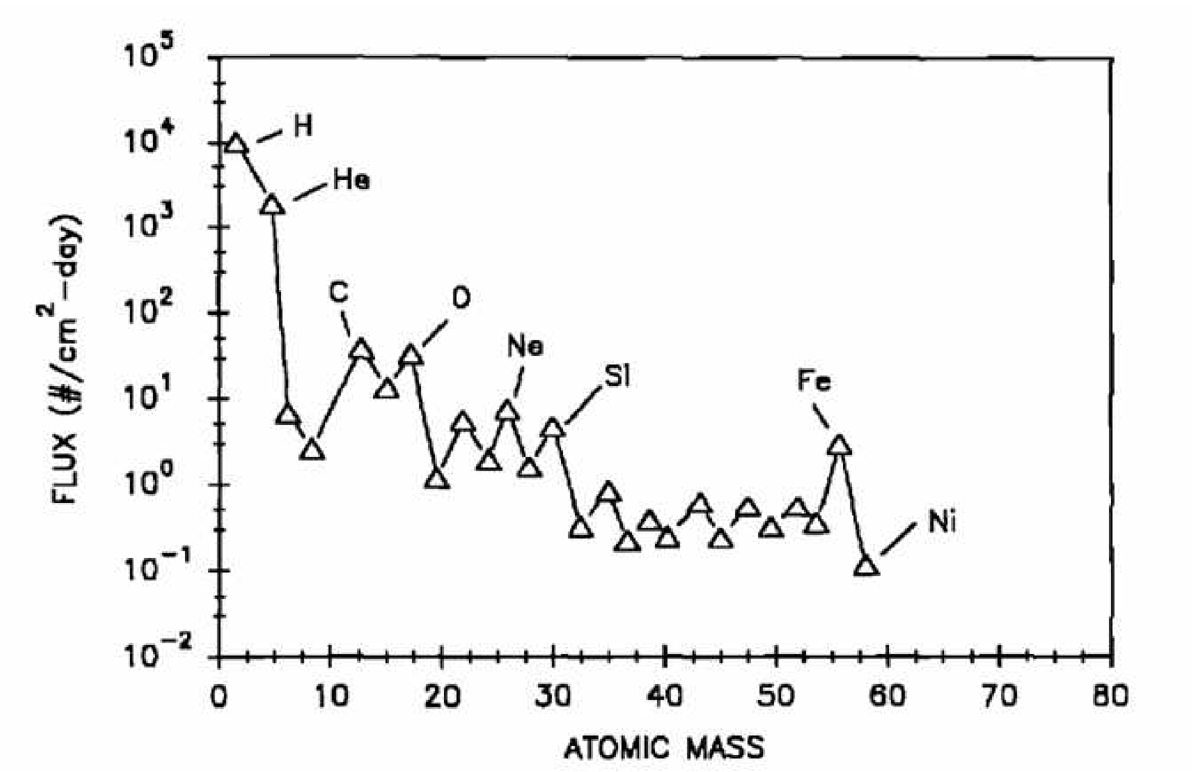


Figure 1: Galactic cosmic ray particle flux as a function of atomic mass for ions up to 60 atomic mass units [34].

Particles and Energies of Interest in this Work

In this thesis, simulated particles are consistent with the GCR environment with energies ranging from 0.1 to 10^5 MeV/nucleon, see figure 2. The particles chosen for this thesis were based on their frequency in the GCR spectrum as seen in figure 1.

Figure 2 was created using CREME96 [35] and assumes 100 mils of aluminum shielding. It plots particle flux $(cm^2\text{-s-MeV})^{-1}$ as a function of particle kinetic energy (MeV) for protons, alphas, oxygen, neon, and iron ions. The peak in the flux occurs near 500 MeV/u for all species. Protons are the most abundant particles at geosynchronous orbit (GEO), followed by alphas, oxygen, neon, and iron (for most energies). The heavier ions are less abundant, but can still have a large, dominant

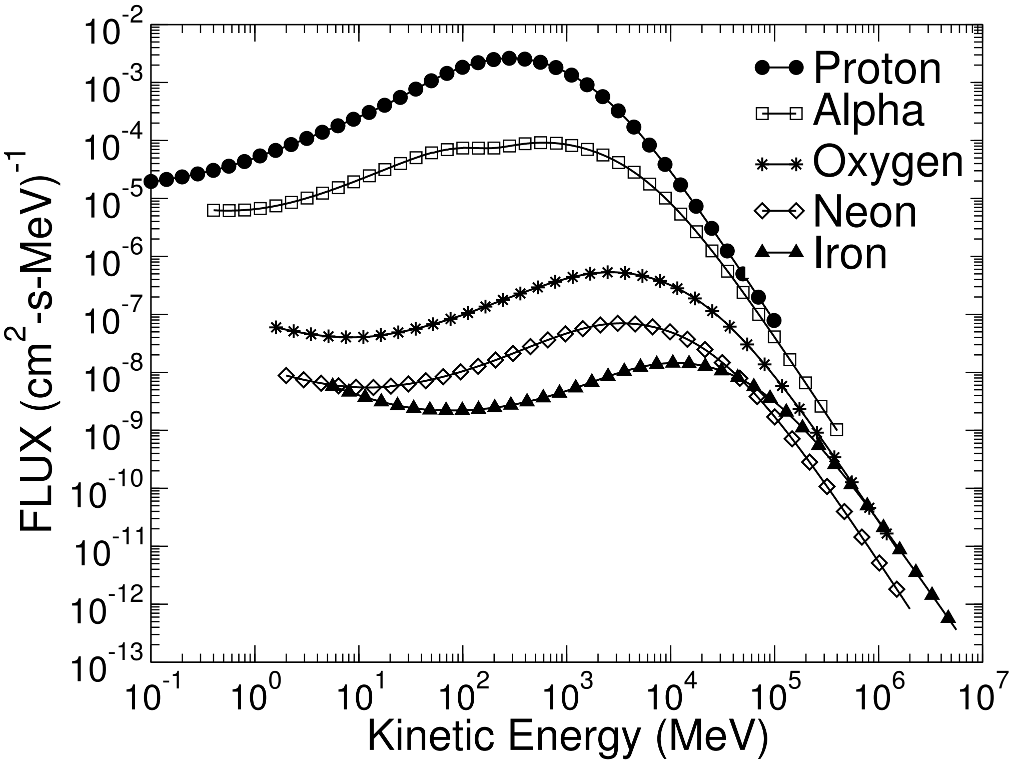


Figure 2: Particle flux as a function of kinetic energy at GEO for protons, alphas, oxygen, neon, and iron [35].

effect on the behavior of devices during space flight.

Results in this thesis are compared over ranges of energies found in space and consistent with ground test facilities. Particle energies found in space are not obtainable by test facilities so a lower range is used to predict the on-orbit response of a microelectronic device. Energies consistent with ground test facilities cover the lower range of figure 2, typically from 15-50 MeV/u, with a max of around 200 MeV/u obtainable at the National Superconducting Cyclotron Laboratory (NSCL) at Michigan State University [3], for $Z > 1$ and 3-200 MeV for protons.

Radiation Effects

Ionizing radiation from particles found in the natural space environment can contribute to total-dose damage, single-event effects, and displacement damage. Protons are accountable for all types of damage while electrons cause total-dose and displacement damage and heavy ions cause single-event effects (SEE). Total-dose damage occurs over the lifetime of the device and is due to changes in the materials from radiation [36]. Displacement damage also occurs over the lifetime of the device and is a result of atoms being knocked out of their lattice sites, creating defects in the materials that make up the device [31].

An SEE is a localized interaction of a single ionizing particle which occurs at random and can have an immediate impact on the device response [37]. SEEs are the result of energy deposition within the materials of a device which contribute to a change in the device's charge. One type of SEE is a single-event upset (SEU). An SEU can be a transient, permanent, or static error [37]. The likelihood an SEU will occur in a space environment is commonly given in a rate of errors/bit/day. The focus of this thesis is on SEUs and energy deposition (charge generation) within a stack of materials resulting from heavy ions and protons.

Energy Deposition

Energy deposition via ionization occurs by two manners: direct and indirect ionization. Direct ionization occurs from reactions between the primary particle and the material being penetrated until the particle comes to rest. When the particle hits a material, some of its energy is transferred directly to the target material thereby creating charged ions. In most devices, this type of ionization is the main mechanism responsible for upsets from ions with Z $>$ 1 [3].

Indirect ionization occurs by secondary particles and includes nuclear elastic and inelastic reactions and screened Coulombic scattering. Secondary particles created through indirect ionization can be much heavier than the original particle and therefore have the capability to deposit larger amounts of energy (or generate charge) [5]. In this thesis generated charge via ionization is tracked from all physical processes from both direct and indirect ionization.

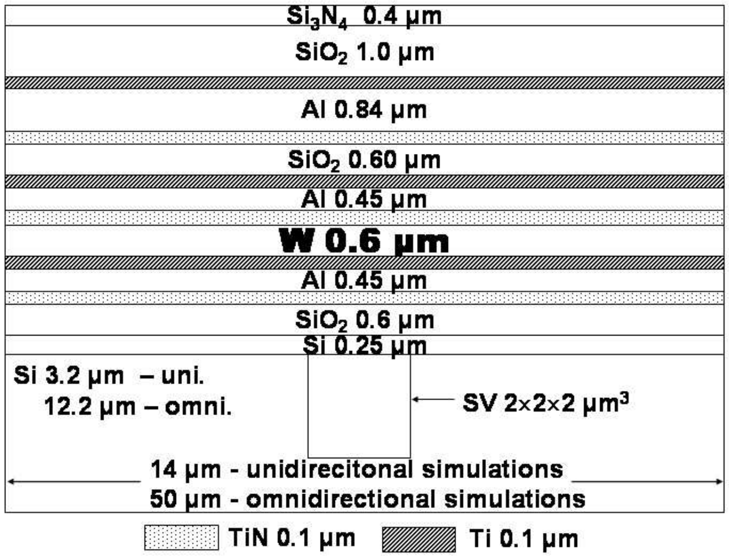
In the past, direct ionization from particles having a Z $\leq$ 1 has been the focus of simulation tools and models for on-orbit response. Previous tools (see table 1 for examples) did not have the capability to track nuclear reactions in solids from heavy ions (Z $>$ 1), and the nuclear component (indirect ionization) was considered insignificant. However, for newer technologies that are scaled to dimensions not previously considered and contain materials that were not used in past semiconductor processing, the indirect ionization plays a role that can not be ignored. This will be shown in the subsequent chapters of this thesis.

CHAPTER IV

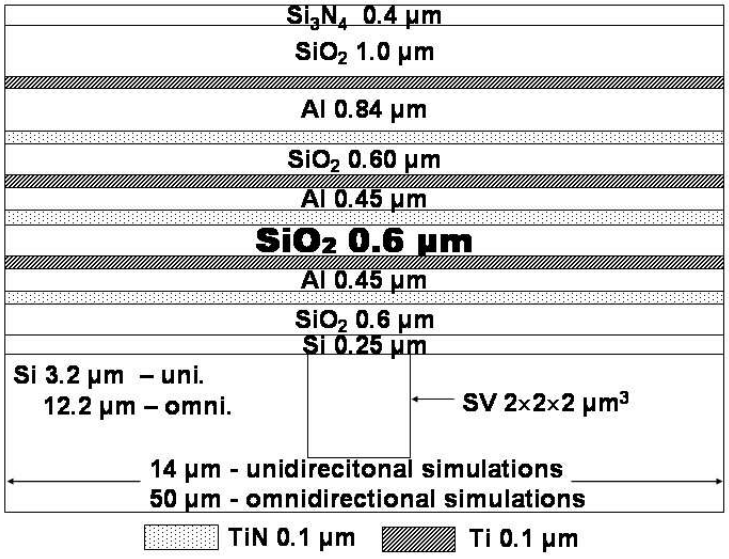
MODELING SCALED CMOS

Two stylized targets are used to investigate the device structure of a modern complementary metal oxide semiconductor (CMOS) technology with a multilayer metallization system, as seen in figures 3(a) and 3(b). They are multilayer stacks of 16 alternating oxides and metal layers of various thicknesses with lateral dimensions for normally incident unidirectional simulations of 14×14 μm^2 and 50×50 μm^2 for omnidirectional simulations. The sensitive volume (SV) for these structures is a $2 \times 2 \times 2$ μm^3 silicon volume located beneath the metallization stack. Sensitive volume refers to the region in which energy deposition (charge generation) must occur to produce an upset. Use of the sensitive-volume concept allows rapid and convenient estimation of the SEU sensitivity of circuits fabricated in a technology with multiple overlayers; more accurate simulations that include detailed descriptions of device and circuit response are possible [24].

The only difference between the two targets in figure 3 is a 600 nm thick layer located 1.5 μm above the sensitive volume. In one structure, this layer is composed of silicon dioxide, while in the other it is tungsten, which is commonly used in integrated circuits to provide electrical connections between layers of metallization or in contacts to the underlying silicon. Specific tungsten structures are commonly referred to by function such as vias, plugs, studs, or interconnects. Tungsten is used because it polishes nicely, deposits easily into high aspect ratio holes, and has low diffusivity. Copper is also commonly used but has been shown to poison silicon if placed too



(a) with a tungsten layer



(b) without a tungsten layer

Figure 3: Cross section representative of scaled CMOS structures (a) with a tungsten layer and (b) without a tungsten layer. Lateral dimensions for normally incident unidirectional simulations are $14 \times 14\ \mu m^2$ and $50 \times 50\ \mu m^2$ for omnidirectional simulations.

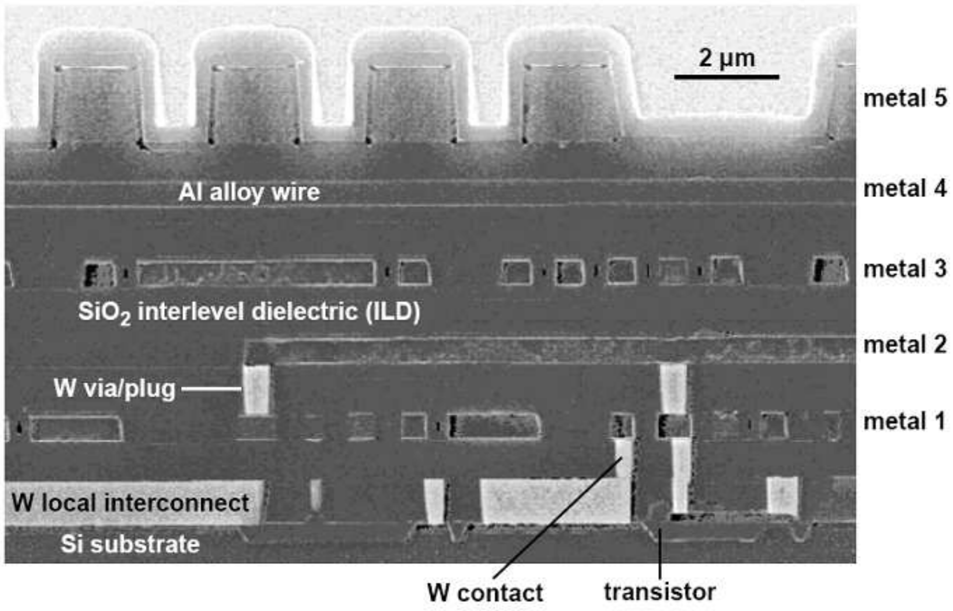


Figure 4: High resolution image of CMOS device cross section with multilevel interconnects and tungsten vias [39].

near the sensitive region [38]. This becomes a larger issue as devices scale to smaller dimensions and these high Z materials are pushed closer to sensitive regions. Figure 4 shows a high resolution image of a CMOS structure containing a tungsten plug. The effects of the tungsten layer on the cross section for various energy depositions will be shown throughout the next few chapters.

Using MRED, a series of simulations were performed using one of the two structures for a fixed ion with a defined energy, tracking the energy deposition via ionization inside the sensitive volume from all physical processes. These energy deposition events are histogrammed into logarithmically spaced bins. The output from MRED is in energy deposited, which is then converted to charge generated by using 22.5 MeV for each 1.0 pC of charge (3.6 eV needed to generate an electron hole pair in silicon [4] divided by the electron charge of 1.6×10^{-19} C equals 22.5 MeV/pC). The integral cross section (σ) for generating a charge (Q) or greater can be determined using the following summation:

$$\sigma(Q, Z, E_Z) = \frac{\sum\limits_{i=i_Q}^{i_{max}} N_i}{\Phi} \tag{1}$$

where N_i is the number of events in the i-th bin, i_Q is the bin corresponding to charge Q, i_{max} is the maximum bin, and Φ is the fluence. It is important to note that σ depends on the ion (Z), ion energy (E_Z), target geometry, and stoichiometry. The fluence is computed by:

$$\Phi = \frac{N_P}{A} \tag{2}$$

where N_P is the total number of ions simulated and A is the irradiated area. The method described above is analogous to the formulation developed for Monte Carlo evaluation of proton-induced effects in [40] and references therein.

A better understanding of cross section can be obtained through a simple example. Suppose a blind folded person throws darts at a dart board hung on a wall. The cross section is then determined by the number of hits on the board divided by the total number of darts thrown normalized by the area of the wall. The darts that hit the dart board are analgous to the ions depositing energy into the sensitive volume and the total number of darts thrown are analagous to the total number of ions simulated. The area of the wall the darts are being thrown towards is analgous to the area of the total structure on which particles are randomized over. Therefore cross section is the probability of hitting a sensitive region (dart board or volume within a structure) weighted by the area.

CHAPTER V

SINGLE-EVENT UPSET ERROR RATE CALCULATION

By calculating an SEU error rate, a prediction of a device's on-orbit response can be determined. In this chapter, an SEU error rate is calculated to determine how the rate is effected when a tungsten layer is present. An SEU error rate for a specific ion and device with critical charge Q_{crit} can be calculated using:

$$Rate_{SEU}(Z) = \sum \sigma(Q_{crit}, Z, E_Z) * \phi(Z, E_Z) * \Delta E_i \tag{3}$$

where $\phi(Z, E_Z)$ is the ion flux (from figure 2) and ΔE_i is the energy bin width. The total on-orbit rate is a summation of $Rate_{SEU}$ over all ions of interest. For the rate calculation, the integral cross section is computed using an omnidirectional ion fluence randomized over 2π steradians of the exposed structure surface as depicted in figure 5. This type of fluence best represents a realistic space environment.

Figure 6 and figure 7 show the integral cross section as a function of energy of the incident particle for four different amounts of generated charge. These simulations were done using ^{16}O ions incident on the structure with and without the tungsten layer respectively.

The simulations in figure 6 follow, for the most part, the expected trend at fixed incident particle energy (E_Z): decreasing cross section for increased charge generation. For all chosen values of generated charge (except 0.22 pC) the direct ionizing process of the primary particle does not play an important role. For the 0.22 pC curve the direct ionizing process dominates for low energy ($<$ 100 MeV) oxygen ions. Also

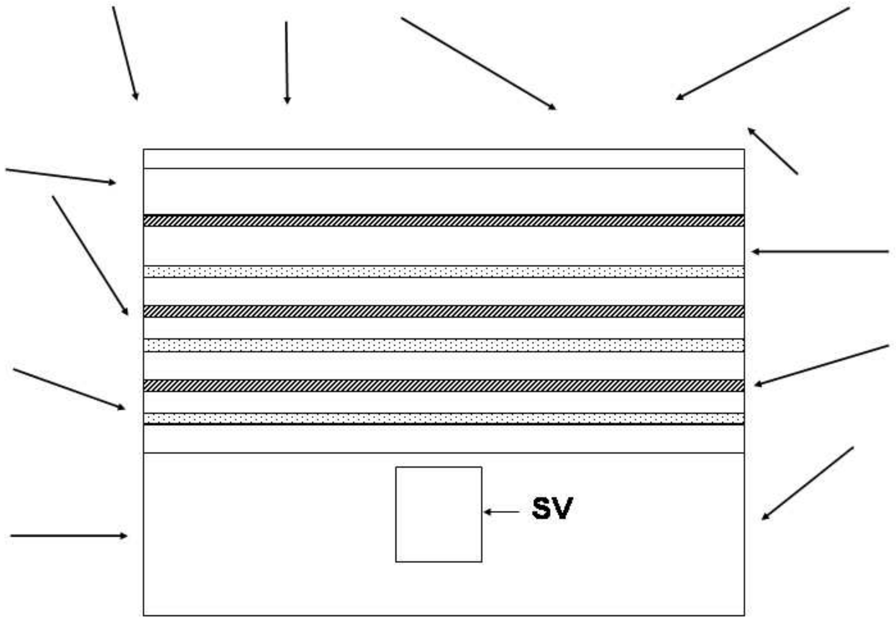


Figure 5: Omnidirectional simulations with the ion fluence randomized over 2π steradians.

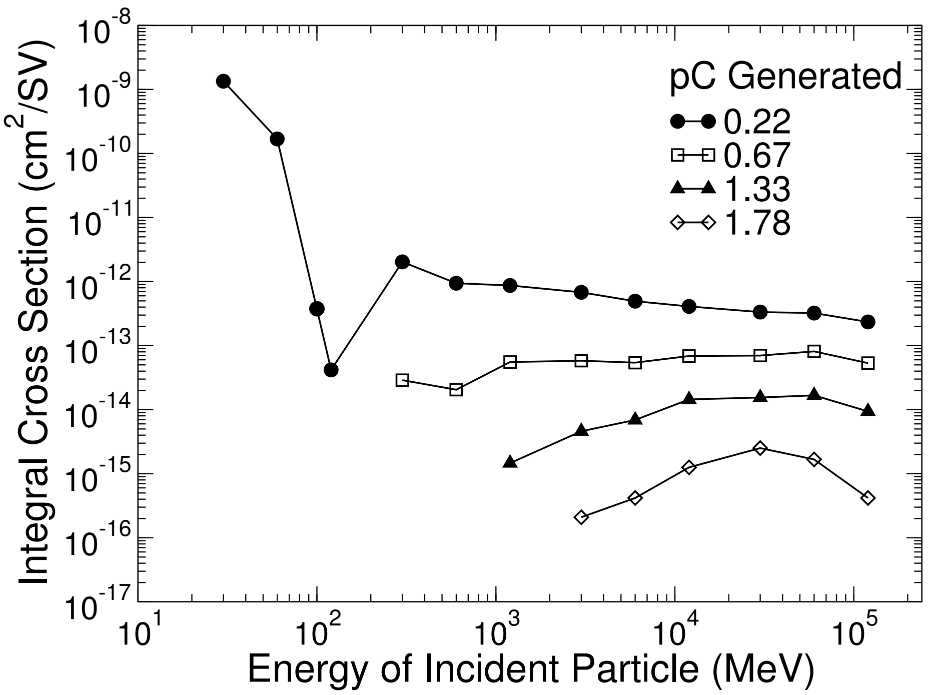


Figure 6: Integral cross section as a function of incident particle energy for oxygen ions incident on the structure with W layer.

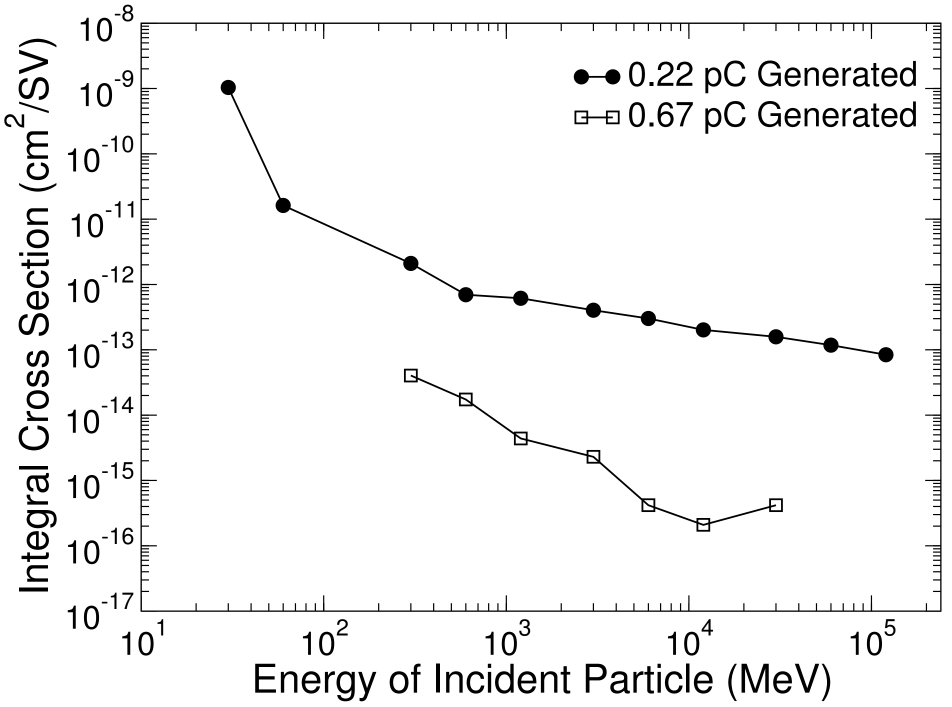


Figure 7: Integral cross section as a function of incident particle energy for oxygen ions incident on the structure without W layer.

note the trend in low E_Z cutoff for all charge generation (except 0.22 pC) is due to the Coulomb barrier introduced between the incident oxygen ion and the target materials. The Coulomb barrier is the energy barrier due to electrostatic repulsion that two nuclei must overcome in order to get close enough to undergo reactions [41]. The Coulomb barrier cutoff is not evident in the 0.22 pC curve. For this case the lowest energy events are due to screened Coulomb scattering and/or direct ionization.

Simulations in figure 7 also follow the expected trend. Notice only two charge generations are plotted because all others were zero for all incident energies. The integral cross section is also much lower for this structure for all incident particle energies.

Figures 8 and 9 show the computed on-orbit SEU error rate as a function of critical

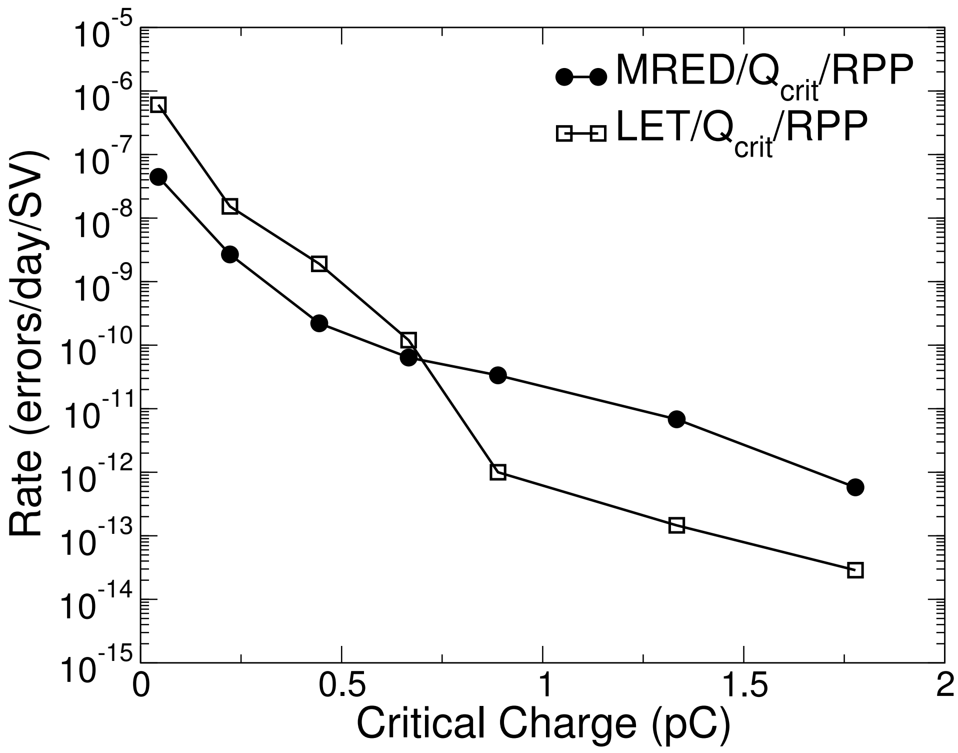


Figure 8: SEU error rate for the structure with W layer computed using MRED and a traditional RPP method. The inclusion of indirect ionization processes in MRED increases the rate by nearly two orders of magnitude for critical charge > 0.65 pC.

charge for both structures compared with the rate computed using CREME96 [35]. Again, the critical charge, Q_{crit}, is the minimum charge generation within the sensitive volume required to produce an upset. The error rates are determined by Eq. 3, which provides the rate of events that generate a charge greater than or equal to Q_{crit}.

The LET/Q_{crit}/RPP rate (open symbols, Figs. 8 and 9) includes direct ionization from all ions in space computed assuming a $2 \times 2 \times 2$ μm^3 rectangular parallelepiped (RPP) and a single critical charge. This calculation was done with the traditional method, i.e., single RPP and single critical charge, using CREME96 (the chord-length model of SEU rate prediction implemented by Pickel and Blandford in 1978 [42]). The chord-length model assumes a RPP and calculates the number of ions that generate

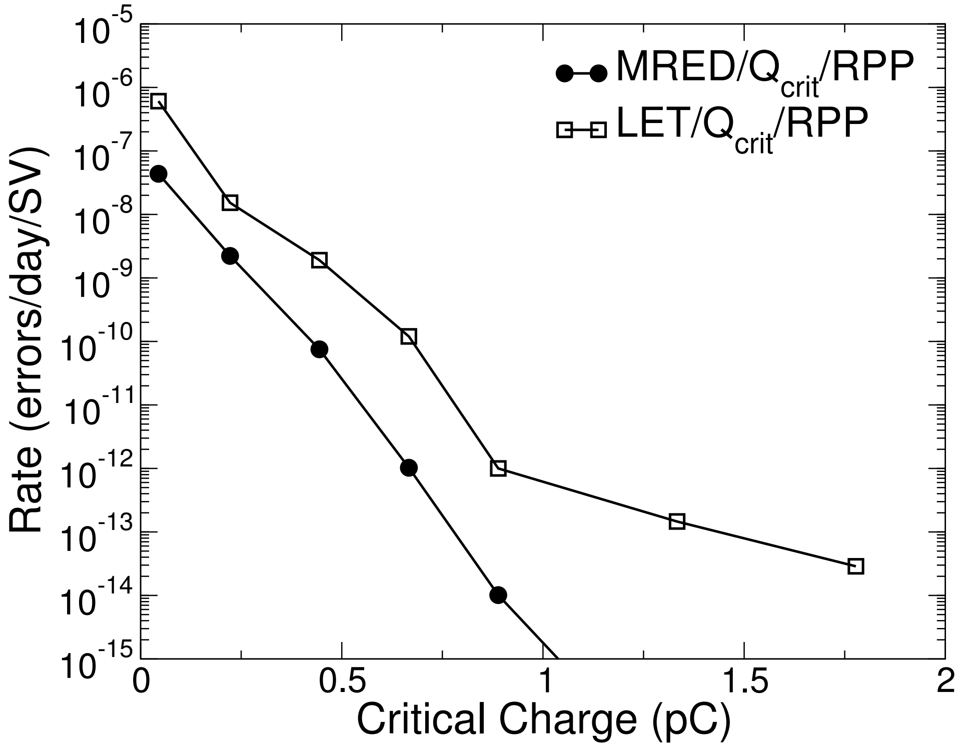


Figure 9: SEU error rate for the structure without W layer computed using MRED and a traditional RPP method. The traditional RPP rate method sufficiently predicts the expected rate and direct ionization dominates.

sufficient charge to upset a sensitive region with an analytic chord-length distribution and an integral LET distribution [6], [3].

The MRED rate (solid symbols, Figs. 8 and 9) is a sum of the individual values for $Rate_{SEU}$ in Eq. 3 for oxygen ions and alpha particles. This calculation includes direct and indirect ionization computed using all physical processes defined in MRED and is likewise computed assuming a $2 \times 2 \times 2$ μm^3 RPP and a single critical charge. Note the LET/Q_{crit}/RPP computations include all ions in the space environment while the MRED-based computations only include oxygen and alphas (the most frequently occurring ions ($Z > 1$) in the space environment).

For the structure with the tungsten layer (Fig. 8), note that even though only oxygen ions and alpha particles are considered, the SEU error rate is dominated by indirect ionization for events that generate more than 0.65 pC; this is a lower limit for the rate. Direct ionization dominates the SEU error rate below 0.65 pC. Since the LET/Q_{crit}/RPP method which includes direct ionization only is nearly two orders of magnitude lower than the MRED/Q_{crit}/RPP rate which includes both direct and indirect ionization, indirect ionization is the leading factor in the observable difference between the two methods. Considering only the direct ionization component will result in an underestimation of the SEU error rate by nearly two orders of magnitude. This demonstrates that the classical SEU rate calculation techniques may not be valid for technologies that have tungsten layers, or other high Z materials, near the sensitive regions. Here we arbitrarily define high Z as materials with an atomic number greater than silicon. This assumption to other high Z materials is valid because the intranuclear binary collision model does not depend on nuclear structure [23].

For the structure without the tungsten layer, direct ionization is the dominant component. For this type of structure, the classical methods for computing the SEU rate are valid. The difference seen between the two rate methods in Fig. 9, for charge generations < 0.65 pC, is due to the inclusion of all ions in the LET/Q_{crit}/RPP calculation while MRED included only alphas and oxygen ions.

Fig. 10 compares the total SEU rate calculated by MRED for the two structures directly. When the tungsten layer is present, the SEU rate is greater by a factor of 100 for certain critical charges. If the overlayers are not considered when calculating the SEU rate, the resulting rate will be underestimated for events that generate between 0.65 and 1.75 pC.

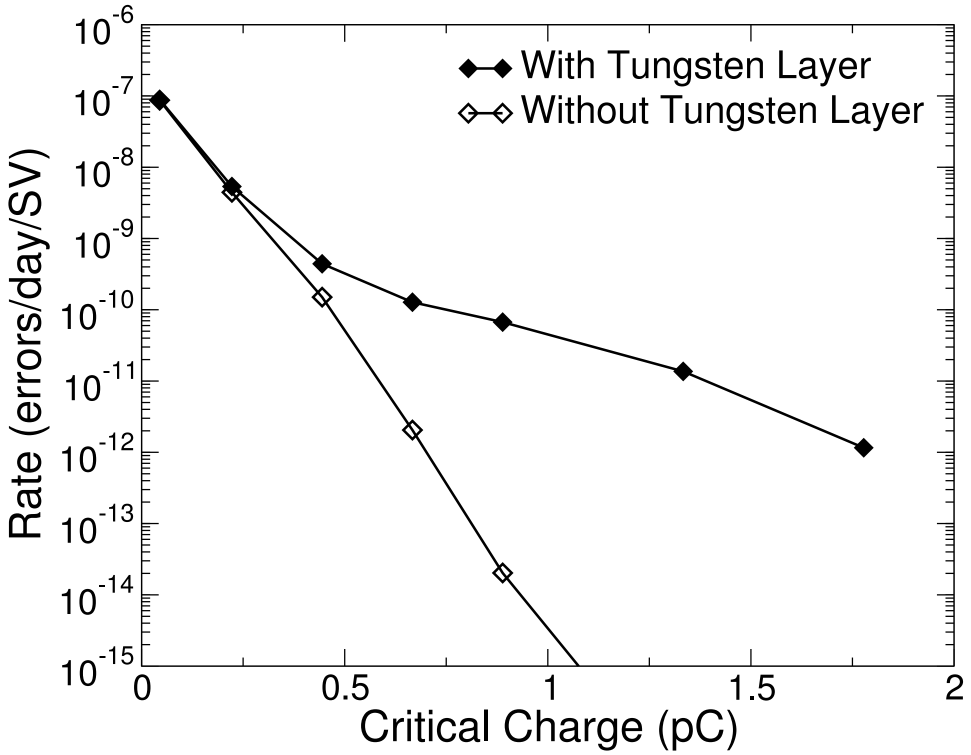


Figure 10: Total SEU error rate for both structures calculated by MRED. When the tungsten layer is present, the calculated rate is orders of magnitude higher for critical charges > 0.65 pC.

CHAPTER VI

CHARGE GENERATION FROM IONS

In this chapter, the effects of the structure, particle type, and particle energy on the integral cross section for charge generation from normally incident particles in described. Throughout this chapter, simulations were done using unidirectional particle fluence and the location of the particles was randomized over the top surface of the structure as depicted in figure 11.

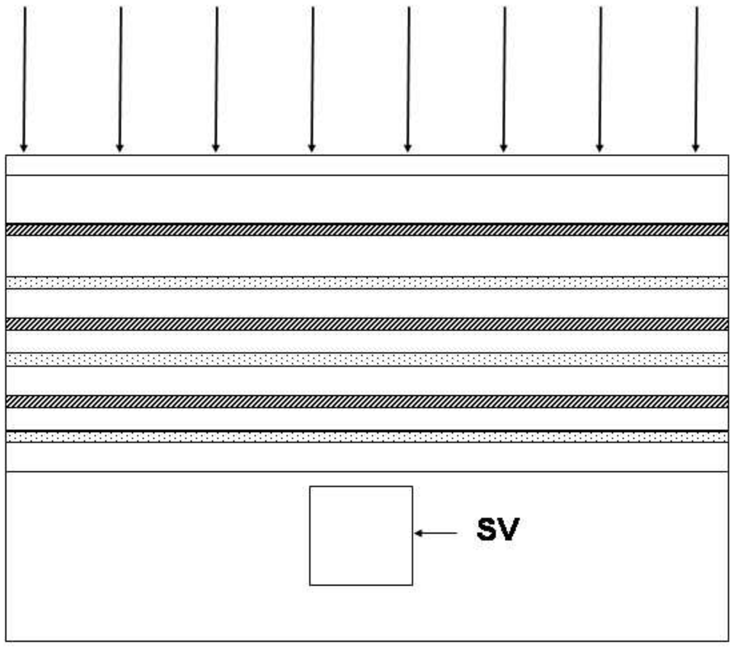


Figure 11: Unidirectional simulations with the ion fluence randomized over the top surface of the structures.

Charge Generation from Heavy Ions (Z > 1)

Figure 12 shows the integral cross section computed using Eq. 1 when 15, 25, and 500 MeV/u oxygen ions are incident on the structure with the tungsten layer. Energies of 15 and 25 MeV/u are representative of typical ground test energies and 500 MeV/u is the value where the peak flux is found in the space environment.

The dramatic decrease in the integral cross section from 10^{-7} to 10^{-12} cm^2/SV near 0.05 pC is due to the limited amount of energy that can be deposited from direct ionization by the primary particle. In [1] this dramatic decrease is shown to be the point where direct ionization falls to zero and nuclear reactions take over. Most of the observable results in this plot are due to indirect ionization processes since direct ionization only results in smaller amounts of charge generation as seen in [1].

Note that the integral cross sections for all energies are of the same order of magnitude up to approximately 0.55 pC of generated charge. At higher amounts of generated charge, the 15 and 25 MeV/u ions result in similar trends in cross section as charge generation increases, approaching zero around 0.8 pC. However, simulations of 500 MeV/u ions shows that the cross section is nonzero until nearly 2 pC of generated charged, after which the cross section falls rapidly. High energy testing is required to capture the response of the circuit fully. This could have dramatic implications for space flight applications of modern technologies tested at typical ground test energies since many potential errors from high energy particles would not be considered.

In Figure 13 the comparison between typical ground test energies and the value near the peak in the flux at GEO is extended to neon for the structure with the tungsten layer. As with the oxygen ions, the cross section falls rapidly for 15 and 25 MeV/u ions at a much lower amount of generated charge than for the 500 MeV/u

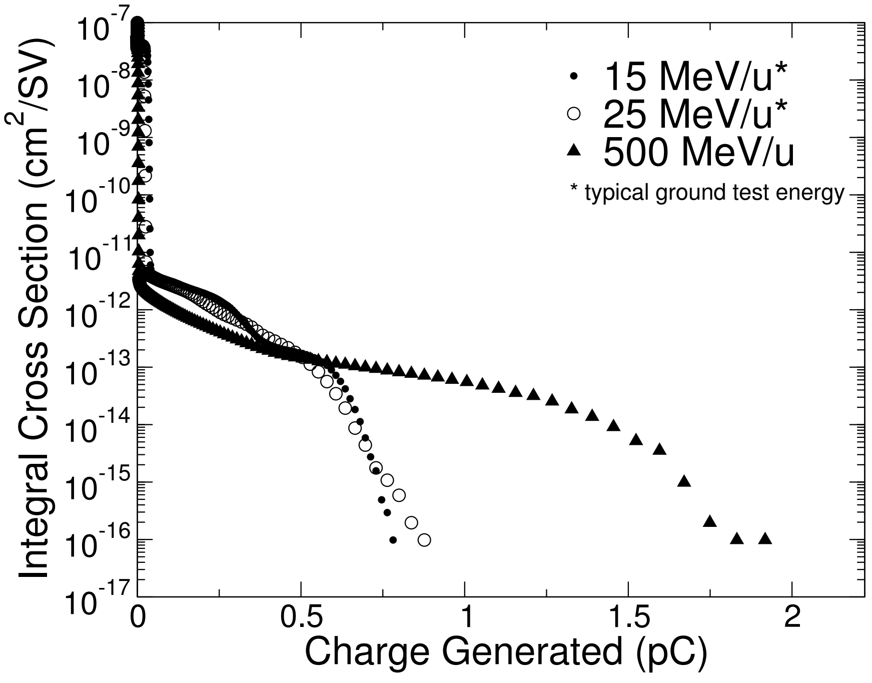


Figure 12: Integral cross section for 15, 25, and 500 MeV/u oxygen ions on the structure with W layer. High energy testing is required to fully capture the response of the circuit.

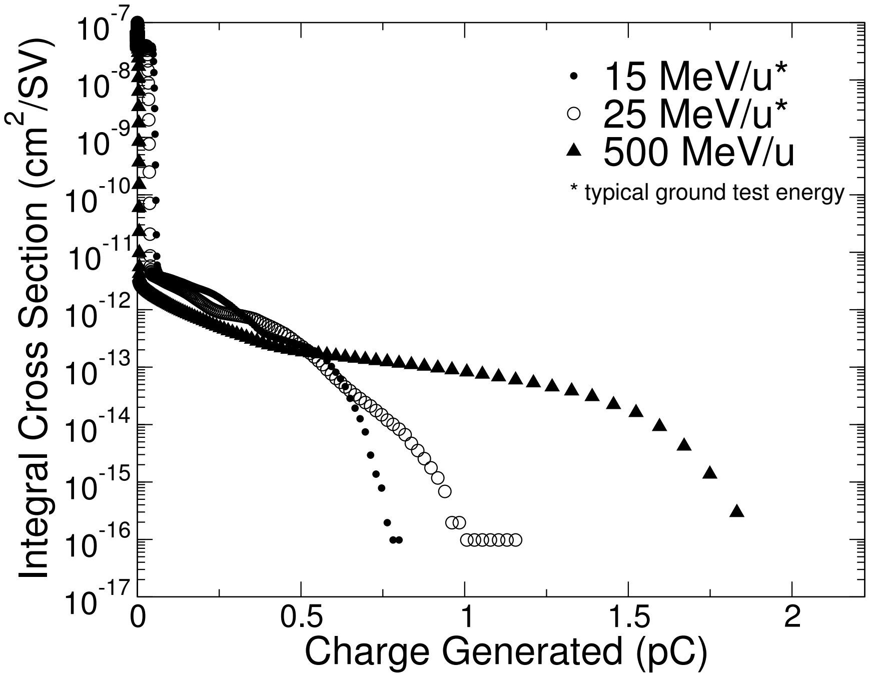


Figure 13: Integral cross section for 15, 25, and 500 MeV/u neon ions on the structure with W layer. High energy testing is required to fully capture the response of the circuit.

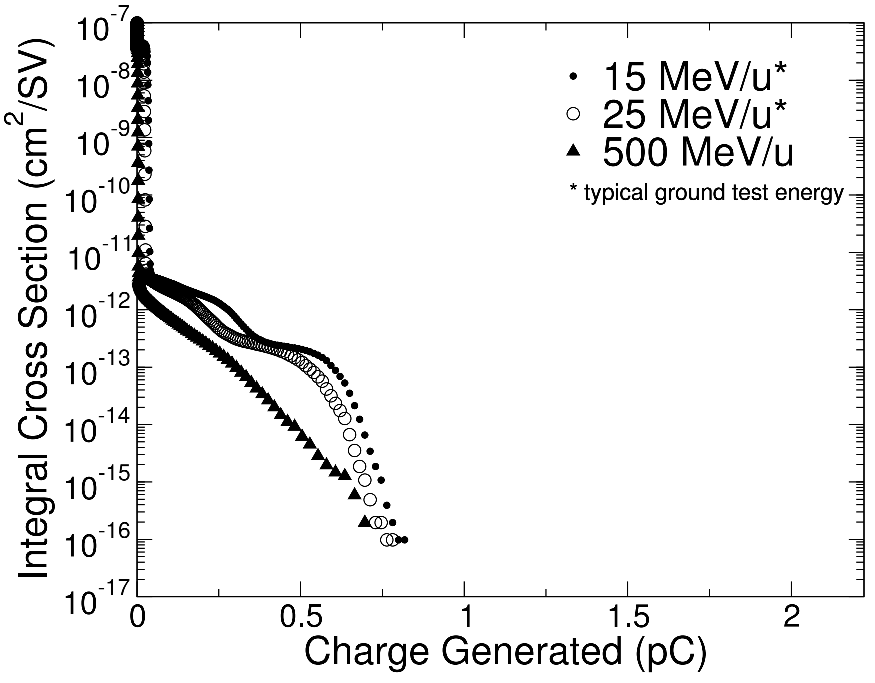


Figure 14: Integral cross section for 15, 25, and 500 MeV/u oxygen ions on the structure without W layer. When the W layer is not present, typical ground test energies sufficiently estimate the expected on-orbit response.

ions. Once again, considering only ground test energies will result in an underestimation of the expected cross section in a real space environment. In [1], the implications of this effect for understanding ground-based measurements on a CMOS SRAM are discussed.

When the tungsten layer is not present, as seen in figure 14 and figure 15, the cross section remains of the same order of magnitude for nearly all amounts of generated charge for the 15, 25, and 500 MeV/u oxygen and neon ions with the 500 MeV/u cases falling well below the other two. Thus, for a circuit lacking high Z materials (e.g., tungsten) in the overlayers, the typical ground test energies would be sufficient to estimate the cross section. This result will be sensitive to the geometry of the target. As described in [24], the location of the high Z material relative to the sensitive

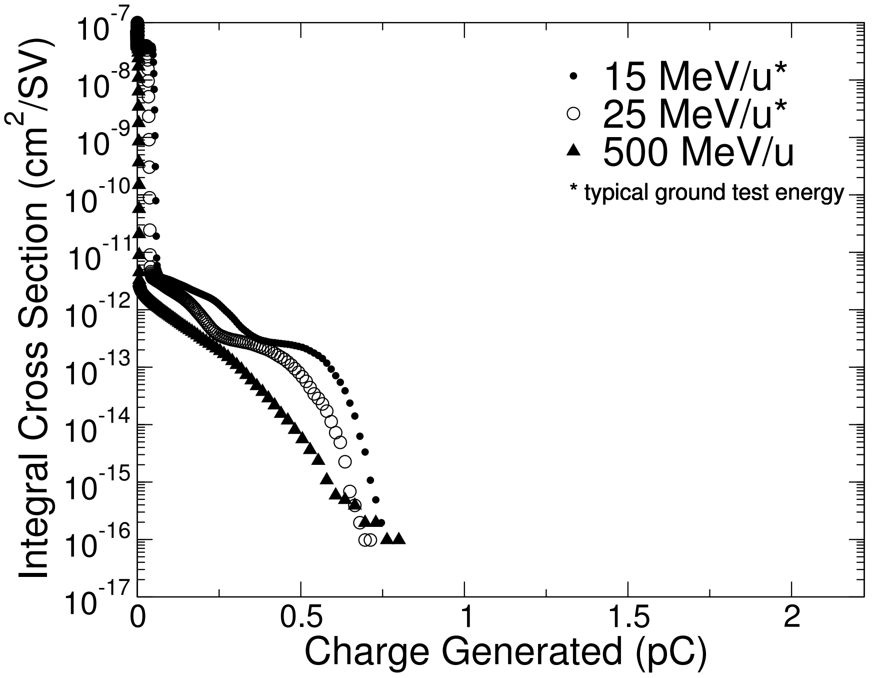


Figure 15: Integral cross section for 15, 25, and 500 MeV/u neon ions on structure without W layer. When the W layer is not present, typical ground test energies sufficiently estimate the on-orbit response.

volume will have a significant effect on the amount of charge generated.

Charge Generation from Protons

Figure 16 shows a comparison between incident protons at three space energies. When comparing these curves we find there is little variation in cross section between the more energetic particles. Also, the cross section approaches zero at a much lower amount of generated charge than it does for the heavy ions. At 500 MeV/u, the cross section from heavy ions falls rapidly around 2 pC of generated charge, while for protons, this occurs at approximately 0.45 pC.

Figure 17 shows a comparison between the structure with and without the tungsten layer at 15 and 500 MeV/u protons. There is an insignificant effect on the cross

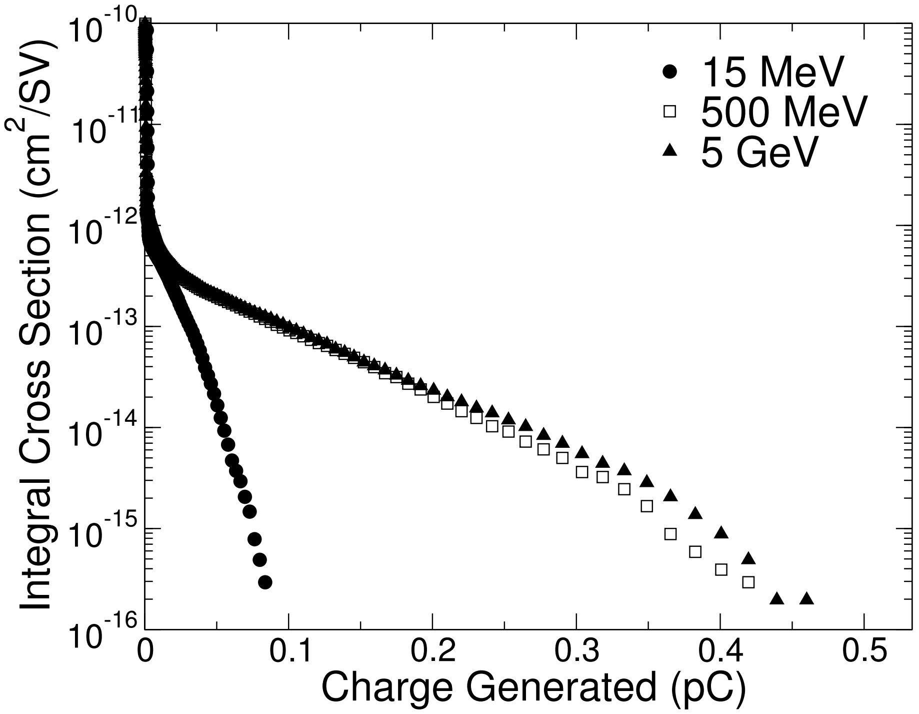


Figure 16: Integral cross section for protons at 15, 500, and 5000 MeV/u for structure with W layer. There is little variation in the cross section between the more energetic particles.

section when the tungsten layer is added. This is much different than the results observed above for the heavy ions where the tungsten layer made a significant difference in the cross section. The lack of strong dependence on the material is due to the limited energy and momentum transfer possible by protons compared to heavier ions. This result is structure dependent and may not hold true for other structures. Schwank, et. al. [43], saw a difference in cross section when comparing 200, 350, and 500 MeV protons on a tungsten structure. As observed by Kobayashi, et. al. [24], the location of the tungsten will determine if the events will generate charge the sensitive volume.

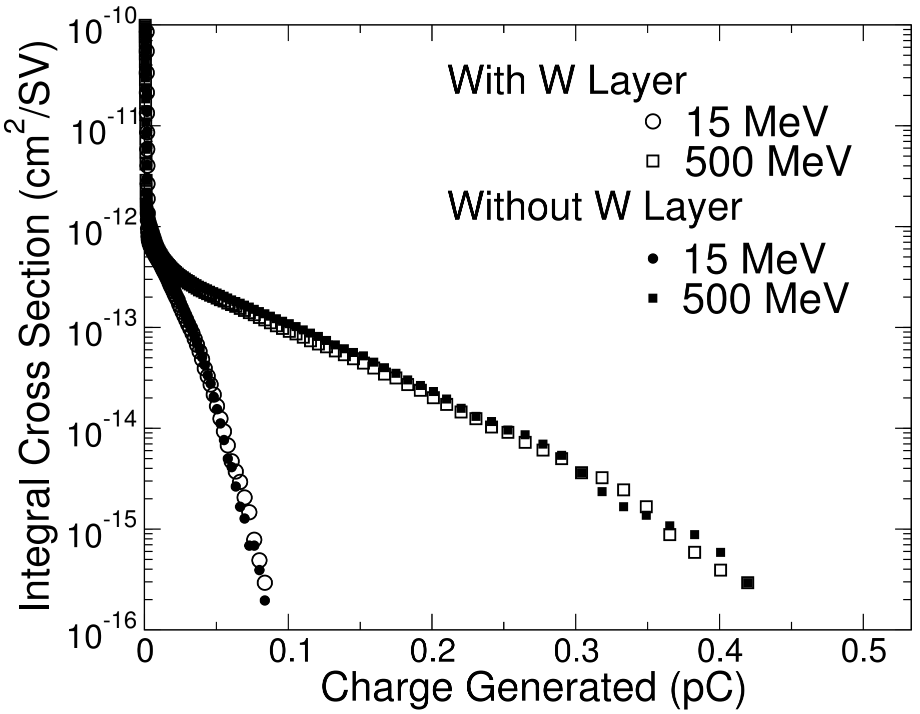


Figure 17: Integral cross section for protons at 15 and 500 MeV/u for both structures. There is little variation in the cross section when the tungsten layer is present.

Ion Species Comparison

Figure 18 makes a comparison of the integral cross section for protons, alphas, oxygen, neon, and iron ions at 500 MeV/u for the structure with the tungsten layer. Oxygen and neon have roughly the same cross section at all amounts of generated charge, while the iron ions have a cross section two to three orders of magnitude larger at higher charge generations. Protons and alphas have a cross section several orders of magnitude lower than the other ions at nearly all charge generations. When referring back to figure 2, we see that while iron ions are the least abundant in space of the particles presented here, it is only by a factor of 4 less than neon ions. Since their cross section is much larger at certain charge generations, they could play a significant role for devices with larger critical charges.

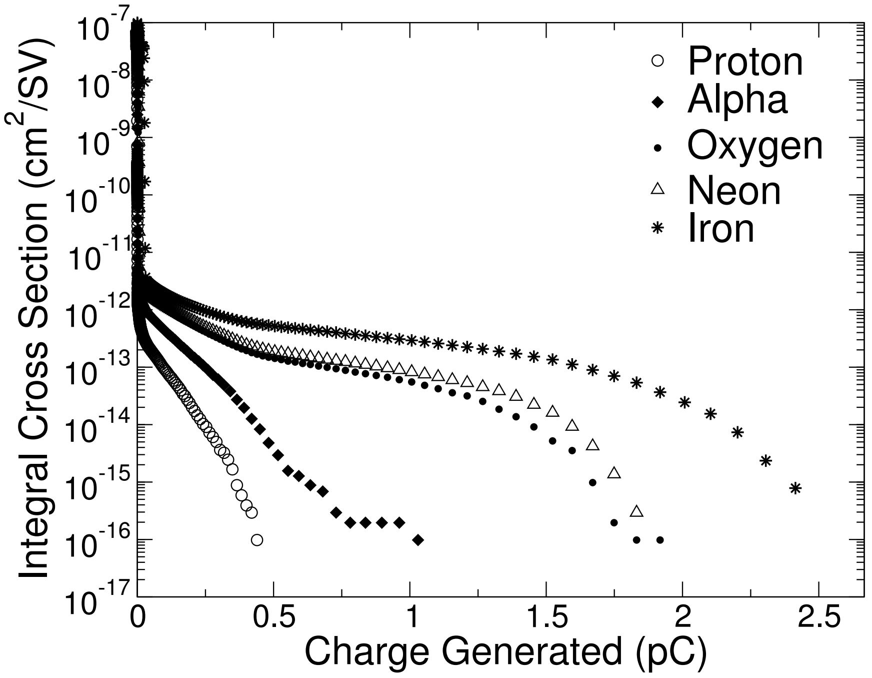


Figure 18: Integral cross section for oxygen, neon, and iron at 500 MeV/u for the structure with W layer. The heavier ions have a large cross section for greater amounts of generated charge.

Figure 19 makes a similar comparison as figure 18 but on the structure without the tungsten layer. When the tungsten is not present, all heavier ion cross sections are within an order of magnitude. Protons are approximately one order of magnitude lower in cross section. The role the tungsten layer plays on charge generation becomes very apparent through the comparison of figure 18 and figure 19. When no tungsten layer is present, the ion type makes very little difference in the cross section at all charge generations.

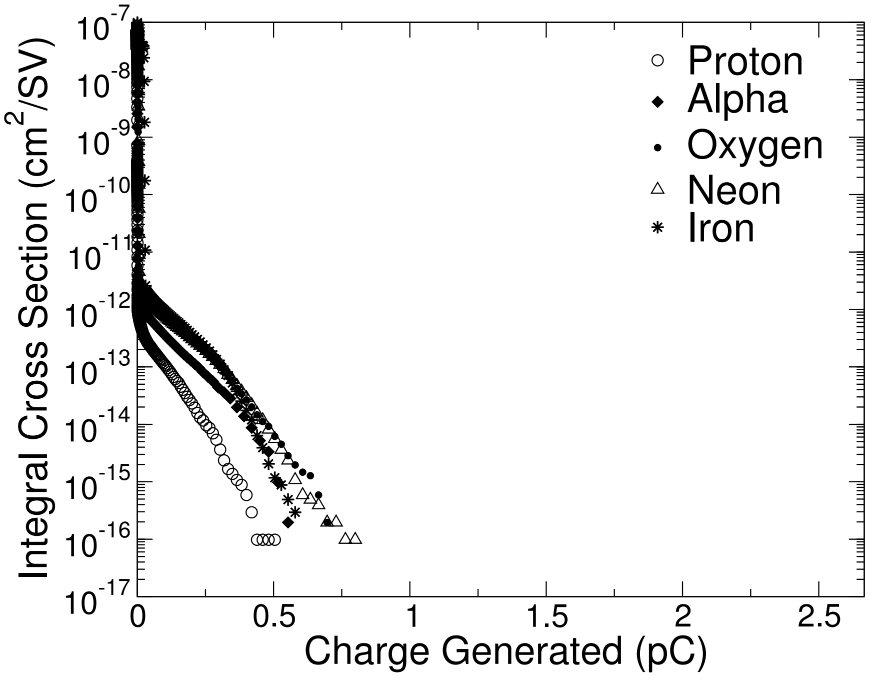


Figure 19: Integral cross section for oxygen, neon, and iron at 500 MeV/u for the structure without W layer. The incident ion type makes an insignificant difference on the cross section.

CHAPTER VII

CHARGE DEPOSITION IN OTHER STRUCTURES

In this chapter, the methods described above are used to investigate charge generation in a modeled SOI technology by ions abundant in space. Comparisons are made over ion species and energy. For protons, results are compared from the Bertini and binary cascade models in Geant4 with equivalent results from the Clemson University Proton Interactions in Devices (CUPID) code.

SOI Structure with Overlayers

Simulation of a modeled SOI device was done using the identical overlayer description used for the scaled CMOS simulations discussed in the previous chapters; however, the sensitive volume size was changed to $2.5 \times 10 \times 0.098\ \mu\text{m}^3$ to represent the thinner sensitive volume found in SOI technologies, see figure 20. Figure 21 compares oxygen ions at 15, 25, and 500 MeV/u for the SOI structure with the tungsten layer. As with the scaled CMOS structure, the low energy ions (representative of ground tests) have a much lower cross section than the more energetic, space-like ions. So again, considering only the ground test energies is not sufficient to predict on-orbit performance.

When comparing the SOI structure with and without the tungsten layer for oxygen ions at the flux maximum as in figure 22, the results are consistent with the CMOS structure in that the tungsten layer plays a much smaller role in determining the cross section below 0.5 pC. Note that the cross section above the 0.5 pC level has very poor

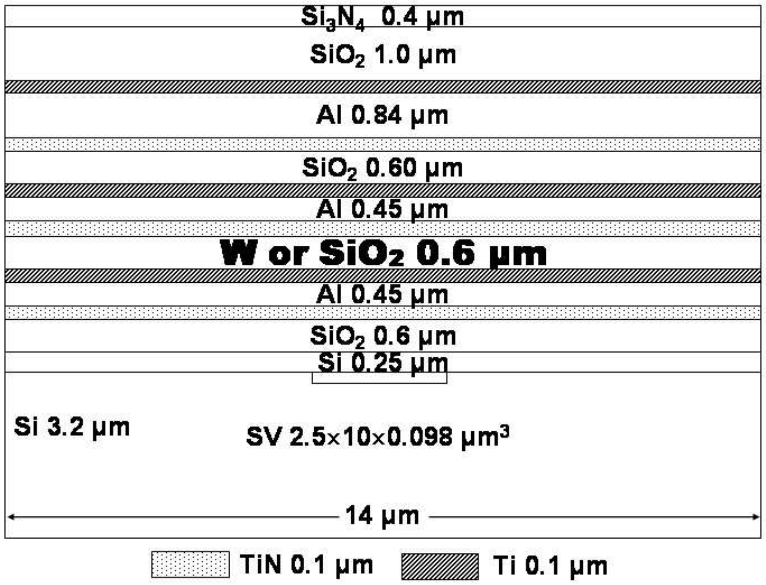


Figure 20: Modeled SOI device with overlayers.

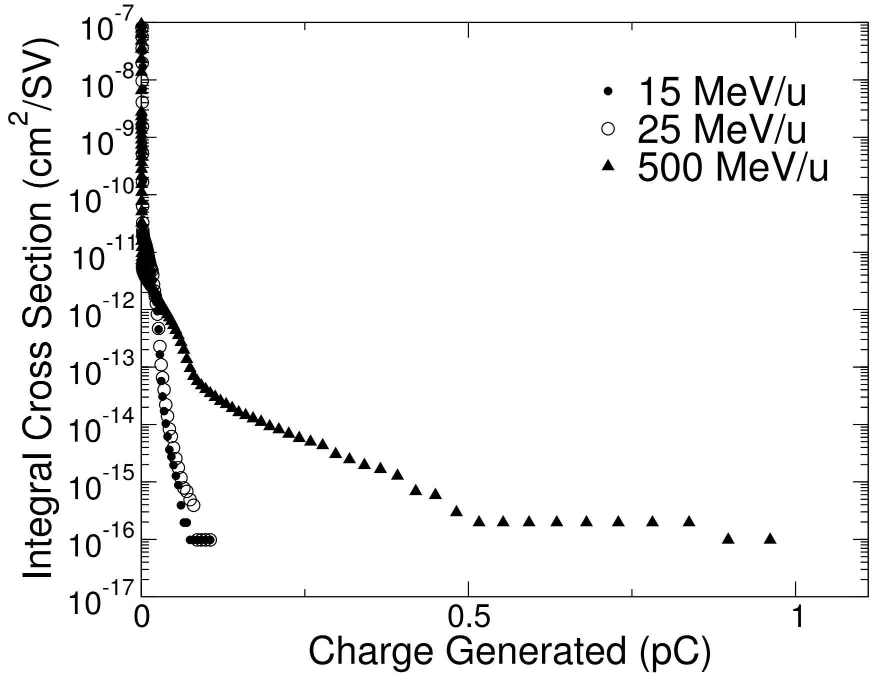


Figure 21: Integral cross section for 15 and 500 MeV/u oxygen ions on SOI structure with W layer. High energy testing is required to fully capture the response of the circuit.

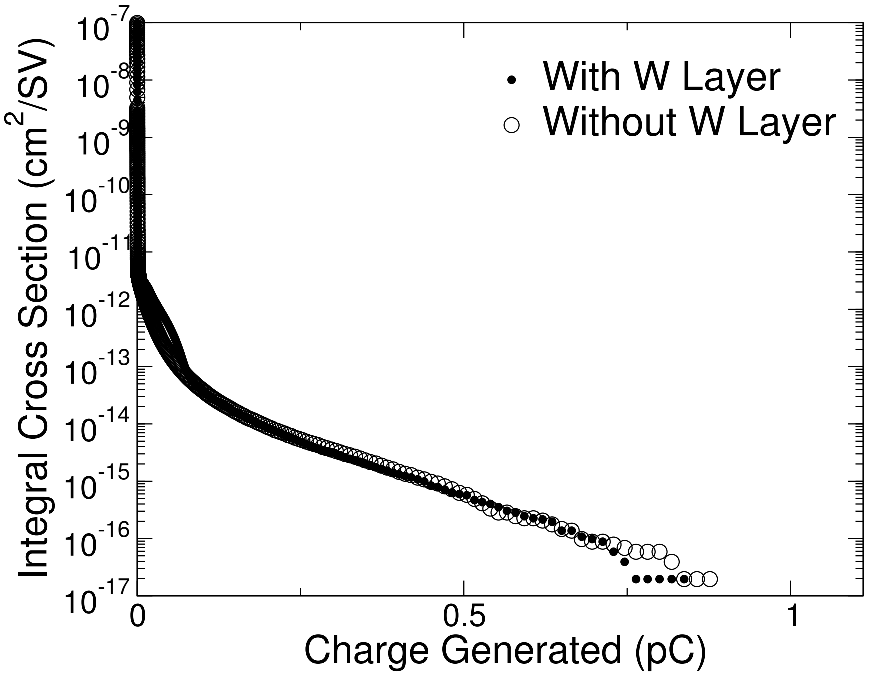


Figure 22: Integral cross section for 500 MeV/u oxygen ions on SOI with and without W layer. The W layer plays a small role in determining the cross section below 0.5 pC of generated charge.

statistics. This is due to the selected number of incident particles and the reduced feature size of the SOI technology.

Simple SOI Device

In this section a comparison between MRED and output published in [2] from the CUPID Monte Carlo code is presented. The target used for this comparison is a rectangular parallelepiped (RPP) composed entirely of silicon with a high aspect ratio sensitive volume ($2.5 \times 10 \times 0.098\ \mu m^3$) in the middle of a larger surrounding volume ($22 \times 30 \times 20\ \mu m^3$) representing a simple SOI device, see figure 23. This structure was used to investigate charge collection volumes that have one dimension much smaller than the others and is identical to that used in [2]. For this analysis,

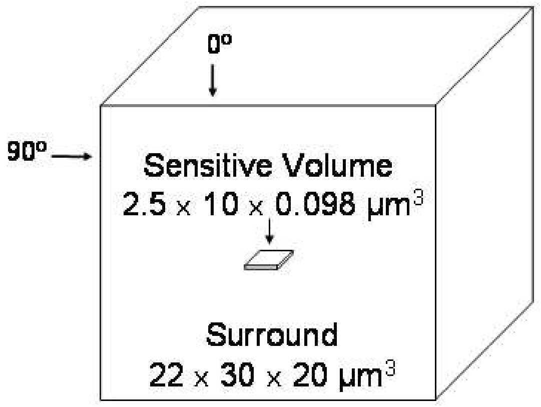


Figure 23: Simple SOI device cross section and angle of incident references.

63 and 200 MeV protons were simulated on the silicon RPP target for a range of incident angles on the large surrounding surface.

Figure 24 shows the integral cross section for incident projectile angles of 0 and 90^o computed with MRED (using binary and Bertini intranuclear cascade models) and CUPID for 63 MeV protons. The binary cascade model is a theoretical model while Bertini is based on empirical data. Zero degrees corresponds to the particles incident normal to the 22×30 μm surface, and rotation to 90^o was done consistent with that in [2]. There is good agreement between all models at 90^o but poorer agreement at 0^o. The two MRED models compared show good agreement for both angles. The CUPID cross sections are slightly lower than those calculated from MRED for energy depositions > 0.4 MeV.

In figure 25 a comparison is made for 200 MeV protons for angles of 0 and 90^o. As with the 63 MeV protons, there is better agreement in the models at 90^o. Comparing the 63 and 200 MeV protons shows there is less of a difference in cross section between the varying angles for the 200 MeV protons.

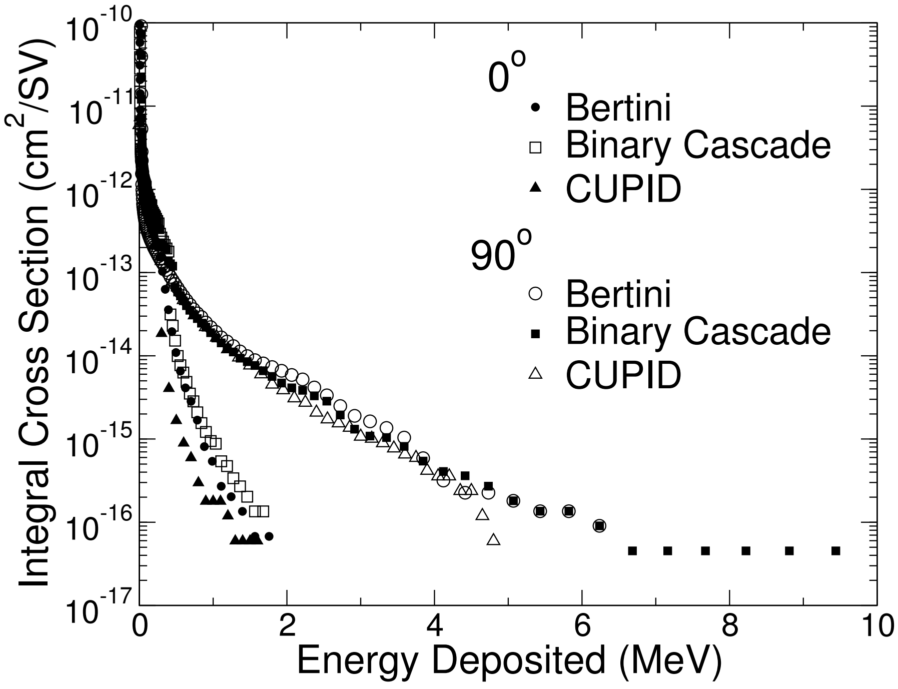


Figure 24: Integral cross section for 63 MeV protons on the RPP target. A comparison between CUPID and two MRED models: binary cascade and Bertini.

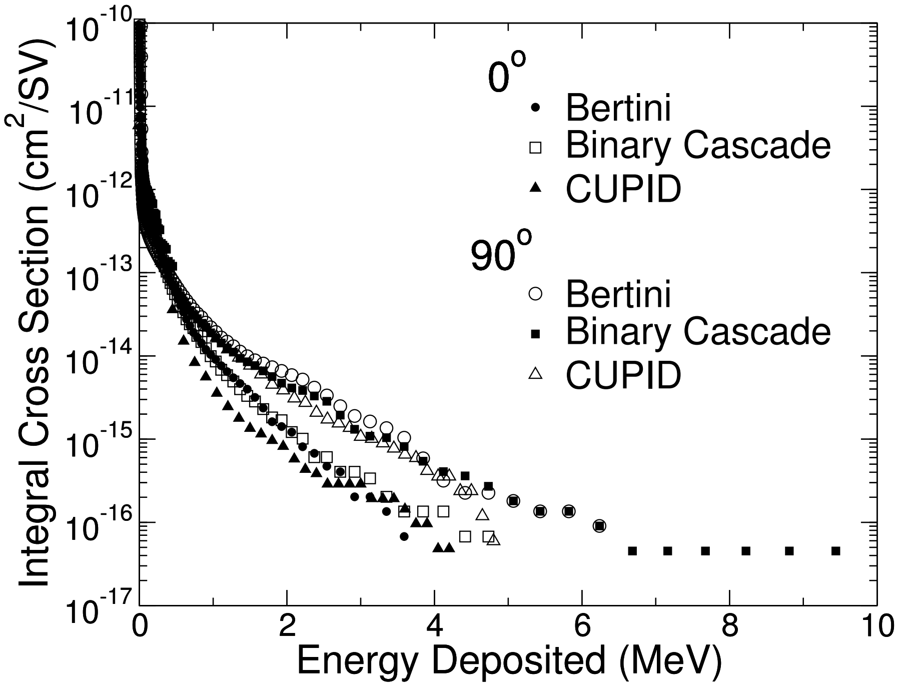


Figure 25: Integral cross section for 200 MeV protons on the RPP target. A comparison between CUPID and two MRED models: binary cascade and Bertini.

CHAPTER VIII

CONCLUSION

Computations using MRED (a code based on a modified Geant4 toolkit) showed that, for heavy ions incident on a structure including a tungsten layer, there is a range of energies where failure to consider all physical processes results in a significant underestimation of events that have the potential to produce errors. If the traditional RPP methods that consider only direction ionization are used for SEU error rate prediction on structures containing tungsten, the on-orbit SEU rate will be underestimated by orders of magnitude. When the tungsten layer is replaced by SiO_2, the traditional methods will sufficiently estimate the rate.

Computations performed for a selection of heavy ions abundant in the GCR spectrum were given, demonstrating that heavy ion atomic and nuclear scattering events can dominate on-orbit performance. This result has significant implications for test methods and rate prediction approaches. For certain structures in which a tungsten layer is present, the cross section increases by several orders of magnitude for certain charge generations from heavy ions. Typical ground test energies (≤ 25 MeV/u) were shown to be insufficient to predict the space performance for structures with tungsten layers.

The integral cross section was shown to depend on particle type, energy, and the structure which it strikes. Proton-induced charge generation in the structures considered in this work did not depend significantly on the presence of tungsten in the overlayers. This is not necessarily a general result; other structures may show sensitivity to tungsten if it is placed closer to the sensitive volume. Charge generation from various ions were compared that showed a greater variance in the cross section for the structure with tungsten. When the tungsten was not present, the cross section had little variation between the varying ions.

A structure representative of an SOI technology had similar dependence on the presence of a tungsten layer: large charge generations were not observed in these structures. Also, results from the MRED application were shown to agree with earlier Monte Carlo simulations on similar targets for two proton energies at various angles.

Appendix A

EXAMPLE MRED INPUT FILES

These examples include only those commands of interest to the work in this thesis and should be used as a reference only. Note that many other commands are available in MRED and input files for future versions of the application will appear differently. Comments are denoted by a # at the beginning of the line. Some comments come directly from the example initialization file provided by MRED's creators.

Initialization File - init.ed.in

Example initialization file for MRED when running in the interactive mode. Physics, target and visualization are defined in other files which follow within this appendix and are pointed to by the initialization file.

```
# Example init file for MRED

# Sets some default verbose states
/control/verbose 2
/control/saveHistory
/run/verbose 2

#----------------------------------------------
# Standard physics is Screened Scattering, NucleonInelasaticD, the binary cascade
# and IonInelastic
/control/execute Std.Physics.in

#----------------------------------------------
# These statements define the target geometry.
# Three alternatives are illustrated.
/control/execute Std.Target.CMOSB.in

#-----------------------------------------------
# The following command can be executed to escape to a command line
# and modify any of the PreInit commands above before proceeding.
# Type "exit" to continue the script.
/control/tcsh
```

```
#----------------------------------------------
# Initializing freezes the physics and geometry for the rest of the run
/run/initialize

#----------------------------------------------
# Include OpenGL visualization or not.
/control/execute Std.Vis.in

#----------------------------------------------
# These control the particle, energy and type of irradiation
/gun/default
/gun/direction 0 0 1

/gun/particle ion
/gun/ion 8 16
/gun/energy 100 MeV
/gun/randomFlux true

# Set the follow two commands to true for hemispherical runs
/gun/randomIsotropic false
/gun/randomHemisphere false

#----------------------------------------------
# These commands control biasing of hadronic interactions
# The default bias factor is 1 and the argument must be positive.
# Values larger than 1 increase the rate of hadronic events.
# Values between 0 and 1 suppress them. Another command, controls
# the "primary only" flag. However /physics/setBiasPrimaryOnly
# should always be true under normal conditions.
/physics/setBiasFactor 200.0
/physics/setUseWeighting true
/physics/printBiasFactor
/physics/snr/setMFPScale 500.

#----------------------------------------------
# These control the output filters
/output/filterEvents false
/output/showFilter

#---------------------------------------------
# These set file outputs on or off
/output/writeHistogramFile true

#---------------------------------------------
# These control the number of events
/output/progressInterval 100000

# A batch file can either exectute a series of runs or drop into a terminal.
# A terminal is always invoked after the default initialization file.
/run/beamOn 1000000

# -Or open a command line.
/control/tcsh
```

Physics File - Std.Physics.in

This is the "standard" phsyics file used for the work in this thesis.

```
# 29/Jan/2005
#---------------------------------------------
# This file contains the commands that set up the physics for a simulation.
# It includes both the commands that have to be issued at PreInit time and
# those that are needed to establish a default condition.
# ----------------------------------------------------------------------------

# The following commands set up the Vanderbilt custom physics list.
# These statements set optional physics processes. Transportation and
# decay are mandatory and not exposed to user interface commands.
# ----------------------------------------------------------------------------

# Select one EM process from the following list. "Standard" is Geant4 standard
# EM physics. "StandardScreened" is the same except that the VU screened
# Coulomb scattering is added for ions. "LowEnergy" is a simple Geant4
# low energy EM physics implementation without screened scattering.
# "EmLowEnergyQED" and "EmPenelopeQED" include screened scattering for ions
# and were intended to be the most comprehensive, using Geant4 low energy
# and Penelope physics respectively. As of this writing, StandardScreened
# appears to have a more compelling treatment of ion stopping and is faster.
# Therefore, it is selected as the default for most semiconductor device
# applications pending improvements in the Geant4 low energy and Pendelope
# code.

#/physics/addPhysics Standard
/physics/addPhysics StandardScreened
#/physics/addPhysics LowEnergy
#/physics/addPhysics EmLowEnergyQED
#/physics/addPhysics EmPenelopeQED
# ----------------------------------------------------------------------------

# These are for elementary particles. Normally use them all.

/physics/addPhysics HadronElastic
/physics/addPhysics HadronInelastic
/physics/addPhysics PiKInelastic
# ----------------------------------------------------------------------------

# The following are alternative physics lists for nucleons. The main ones are
# "NucleonInelastic" and the "B" and "C" variants. "NucleonInelastic" uses
# the binary cascade and high precision neutron code. The "B" variation also
# uses the high precision neutron code but uses the Bertini cascade. Finally,
# the "C" variant uses the Bertini cascade and standard neutron physics.

#/physics/addPhysics NucleonInelastic
#/physics/addPhysics NucleonInelasticB
#/physics/addPhysics NucleonInelasticC
/physics/addPhysics NucleonInelasticD
#/physics/addPhysics Binary
#/physics/addPhysics BinaryLE
```

```
#/physics/addPhysics HPBinaryLE
#/physics/addPhysics Bertini
#/physics/addPhysics BertiniLE
#/physics/addPhysics HPBertiniLE
# -------------------------------------------------------------------------------

# These are the ion-ion nuclear reaction modules. IonAbrasion is the code
# from Qinetiq. For most of our applications our older IonInelastic is
# identical. IonAbrasionEMD is the code from Qinetiq that includes
# electromagnetic dissociation. This was broken as of Geant4.6.2.p02 and
# should not be used, since it's only valid significantly above our usual
# energy range anyway.

/physics/addPhysics IonInelastic
#/physics/addPhysics IonAbrasion
#/physics/addPhysics IonAbrasionEMD
# --------------------------------------------------------------------------------
# Cuts may be modified after run initialization. A value here overrides the
# hard-wired default of 100 nm.

/physics/setCuts 8. micrometer
```

Target File - Std.Target.CMOSB.in

This file defines the target geometry for MRED input. The example provided here sets up the representative CMOS stack containing a tungsten layer.

```
# 16/Jan/2005
#---------------------------------------------
# These statements define the geometry of the target designated CMOSB. It
# has an tungsten layer, where CMOSA has SiO2.
#
# The first layer is a full definition. Setting the number of layers after
# defining the first layer propagates the lateral dimensions downward
# with the default material and sensitivity turned off. Each layer below
# should minimally define the thickness, material and sensitivity for
# calorimeter layers.
#
/sample/setDeviceType rpp
/sample/setLayerPointer 1
/sample/setLayerX 14 micrometer
/sample/setLayerY 14 micrometer
/sample/setLayerZ 0.4 micrometer
/sample/setLayerMaterial Si3N4
/sample/setLayerSensitive false
/sample/setNumberOfLayers 16
#
/sample/setLayerPointer 2
/sample/setLayerZ 1.0 micrometer
/sample/setLayerMaterial SiO2
```

```
#
/sample/setLayerPointer 3
/sample/setLayerZ 0.1 micrometer
/sample/setLayerMaterial TiN
#
/sample/setLayerPointer 4
/sample/setLayerZ 0.84 micrometer
/sample/setLayerMaterial aluminum
#
/sample/setLayerPointer 5
/sample/setLayerZ 0.1 micrometer
/sample/setLayerMaterial titanium
#
/sample/setLayerPointer 6
/sample/setLayerZ 0.6 micrometer
/sample/setLayerMaterial SiO2
#
/sample/setLayerPointer 7
/sample/setLayerZ 0.1 micrometer
/sample/setLayerMaterial TiN
#
/sample/setLayerPointer 8
/sample/setLayerZ 0.45 micrometer
/sample/setLayerMaterial aluminum
#
/sample/setLayerPointer 9
/sample/setLayerZ 0.1 micrometer
/sample/setLayerMaterial titanium
#
/sample/setLayerPointer 10
/sample/setLayerZ 0.6 micrometer
/sample/setLayerMaterial tungsten
#
/sample/setLayerPointer 11
/sample/setLayerZ 0.1 micrometer
/sample/setLayerMaterial TiN
#
/sample/setLayerPointer 12
/sample/setLayerZ 0.45 micrometer
/sample/setLayerMaterial aluminum
#
/sample/setLayerPointer 13
/sample/setLayerZ 0.1 micrometer
/sample/setLayerMaterial titanium
#
/sample/setLayerPointer 14
/sample/setLayerZ 0.6 micrometer
/sample/setLayerMaterial SiO2
#
/sample/setLayerPointer 15
/sample/setLayerZ 0.25 micrometer
/sample/setLayerMaterial silicon
#
/sample/setLayerPointer 16
```

```
/sample/setLayerX 2.0 micrometer
/sample/setLayerY 2.0 micrometer
/sample/setLayerZ 2.0 micrometer
/sample/setLayerMaterial silicon
/sample/setLayerSensitive true
#
/sample/setSensitiveDepth 0 micrometer
#
/sample/setWaferX 14 micrometer
/sample/setWaferY 14 micrometer
/sample/setWaferZ 9 micrometer
/sample/setWaferMaterial silicon
/sample/setWaferOffset 0 micrometer
#
/sample/setWorldRadiusFactor 1.1
#
```

Visualization File - Std.Vis.in

This file provides visualization for the MRED appliation so the user can see the target and the reactions created by incident particles. Visualization was used for debugging purposes.

```
# 6/Jan/2005

#--------------------------------------------------------------------------------
# These statements set up standard visualization.
# They are not needed for batch runs on Vampire and should be
# commented out when running there.
# Never use the built-in commands vis/scene/add/volume and
# vis/scene/add/trajectories. The latter bypasses the code
# in EndOfEventAction and writes trajectories for events that
# have been rejected as invalid.

# create empty scene
#/vis/verbose 0
/vis/scene/create

# Create a scene handler for a specific graphics system
# (Edit the next line(s) to choose another graphic system)

# Dawn file creation
#/vis/open DAWNFILE

/vis/open OGLIX 500

#/vis/viewer/flush
```

```
# for drawing the tracks
# (if too many tracks cause core dump => storeTrajectory 0)
/tracking/storeTrajectory 0
/vis/scene/endOfEventAction accumulate

/vis/viewer/zoom 1
/vis/viewer/set/viewpointVector -1 0 0
#/vis/viewer/set/style surface
# /vis/viewer/set/style wireframe
#------------------------------------------------
```

Appendix B

PYTHON SCRIPTS FOR PARRELLEL MRED APPLICATIONS

The following are a series of files used to duplicate MRED runs and distribute the jobs in parallel on a high performance computing cluster. All files were developed by Dr. Marcus Mendenhall at Vanderbilt University Free Electron Laser Lab for support of this and future work. Following the section titles will be a brief description of each file then the python code contained within the file.

splitrun.py

This file is the low-level workhorse that handles the division and submission of jobs onto the computing cluster. It handles all the necessary bookkeeping and creates the commands that will do the actual submission. Script was written by Marcus Mendenhall.

```
#!/usr/bin/env python
"""$Id: splitrun.py,v 1.53 2005/07/01 17:24:27 marcus Exp $
splitrun handles  division of jobs on vampire.
It runs itself as a script with various functions depending on the first argument
(see code at the bottom). The main user functions used when it is imported are
splitrun_binary(), splitrun_batch(), summarize_histograms() and
summarize_batch_histograms()."""

_CVSVers="$Id: splitrun.py,v 1.53 2005/07/01 17:24:27 marcus Exp $"

import sys
import os
import time
import cPickle
import base64
import types

def splitrun_binary_tree(runvars):
     "splitrun_binary_tree is the working kernel of splitrun_binary, and only for
        internal use"
```

```
    import base64
    import cPickle
    import os
    nodelist=runvars['nodelist']

    pids=[]

    #pick off processes assigned to run locally and to launch via rsh from here
    thisnode=nodelist[0][1]
    localruns=[x for x in nodelist if x[1] == thisnode]
    for x in localruns: #take them out of the main list
         nodelist.remove(x)

    # and we will start first 10 from here, too
    localruns += nodelist[:min(10, len(nodelist))]
    del  nodelist[:min(10, len(nodelist))]

    #split any remaining process launches recursively onto some other processors
    half=len(nodelist)//2
    for nodes in (nodelist[:half], nodelist[half:]):
         if nodes:
              node=nodes[0][1]
              runvars['nodelist']=nodes
              runstring=
         ' '.join(base64.encodestring(cPickle.dumps(runvars,-1)).split())
      #convert all white space to spaces
              command='python '+__file__+" split_binary "+runstring
              result=
         os.spawnl(os.P_NOWAIT,"/usr/bin/rsh","/usr/bin/rsh",node, command)
              pids.append(result)

    #now, execute local and direct rsh launches
    for index, node in localruns:
         runvars['index']=index
         runvars['node']=node
         command="%s &> %s"%(runvars['baseCommand'] % runvars,
                     runvars['baseOutfile'] % runvars)
         if node == thisnode:
              result=
         os.spawnl(os.P_NOWAIT, '/bin/bash', '/bin/bash', '-c', command)
              print "locally running ", command
         else:
              result=
         os.spawnl(os.P_NOWAIT,"/usr/bin/rsh","/usr/bin/rsh",node,command)
              print "running on node ", node, command

         pids.append(result)

    while pids:
         done, stat=os.waitpid(-1, 0)
         pids.remove(done)

def splitrun_binary(baseCommand=None,
                    baseOutfile=None, sleepTime=1, extra_keys={},**kwargs):
```

```
    """runinfo=splitrun_binary() splits a run over multiple processors.
%(isotime)s is replaced with yyyymmdd.hhmmss
%(index)d is replaced with a sub-process index.
%(node)s is replaced with the vampire node name on which this process ran
any keys in extra_keys or anywhere else are also available for subst. in command
and outfile strings. It returns a list with information about the subprocesses."""

    isotime=time.strftime("%Y%m%d.%H%M%S")

    #merge old-style extra_keys dict into generic keywords
    kwargs.update(extra_keys)
    del extra_keys

    runvars=locals().copy()
    runvars.update(kwargs)
    del runvars['kwargs'] #and delete redundant nested copy

    nodelistname=os.environ['PBS_NODEFILE']
    f=open(nodelistname,'r')
    runinfo=[]
    nodelist=[]

    for index, nodes in enumerate(f):
        node=nodes.strip()
        rundict={'index': index, 'isotime': isotime, 'node':node }
        rundict.update(kwargs) #pick up all keys from parent, too
        command="%s &> %s"%(baseCommand % rundict, baseOutfile % rundict)
        runinfo.append((rundict,0,baseCommand % rundict,baseOutfile % rundict))
        nodelist.append((index,node))

    f.close()
    runvars['nodelist']=nodelist

    splitrun_binary_tree(runvars)
    return runinfo

def splitrun_batch(template=None, nCopies=5, finalTask=None,
    jobName="RUN.%(isotime)s.%(index)03d", parameterDictList=[],
    cleanupName="RUN.%(isotime)s.END", sanitize_extra_keys=1,
                make_killer_script=1, **extra_keys):
    """splitrun_batch() splits a run over multiple processors in separate batch
        jobs, and queues a dependent batch job (if desired) to run at the end,
  which gets info about the runs.
                %(isotime)s is replaced with yyyymmdd.hhmmss
                %(index)d is replaced with a sub-process index.
                %(runVarsString)s is replaced with a pickled &
        encoded copy of most of the run information, for use by finalTask,
  typically any extra keys are also available for substitution in the
  command and outfile strings.

                Also, if parameterDictList is not empty, each run will receive
  parameters from corresponding dict.

                The main template can also contain a %(dictString)s declaration
```

```
which  will be formatted with a base64 encoded pickle of the complete set
of variables to be passed to the run.

                For easier use, the template %(variablesDict)s will be replaced
with the entire python command to decode the dictString i.e.
'cPickle.loads(base64.decodestring(<dictionary string data>))'.
Make sure that you have done import cPickle; import base64; before using this.
Using this, then, in a template running a python script (python -c "script..."),
the sequence  vars=%(variablesDict)s;
        will generate the code to assign all the variables to the dictionary 'vars'

   """

   if parameterDictList:
        nCopies=len(parameterDictList)

   isotime=time.strftime("%Y%m%d.%H%M%S")

   if sanitize_extra_keys:
        #remove modules, functions, and private
# names (beginning with _) from extra_keys
        for k in extra_keys.keys():
             if (
                      k[0]=="_" or
                      type(extra_keys[k]) in (
                         types.ModuleType, types.FunctionType, types.ClassType,
 types.BuiltinFunctionType)
                  ):
                  del extra_keys[k]

   runvars=locals().copy()
   runvars.update(extra_keys) #merge this into runvars
   del runvars['extra_keys'] #and delete redundant nested copy
   runstring=
        ' '.join(base64.encodestring(cPickle.dumps(runvars,-1)).split())
   #convert all white space to spaces
   runvars['runVarsString']=runstring
   runvars['runVarsDict']="cPickle.loads(base64.decodestring('"+runstring+"'))"

   joblist=[]
   for index in range(nCopies):
        rundict={'index': index, 'isotime': isotime }
        rundict.update(extra_keys) #pick up all keys from parent, too
        if parameterDictList:
             rundict.update(parameterDictList[index])

        rundict['dictString']=
          ''.join(base64.encodestring(cPickle.dumps(rundict,-1)).split())
        rundict['variablesDict']=("cPickle.loads(base64.decodestring('"+
             rundict['dictString']+"'))")

        jobtries=3
        while jobtries:
             #the following line is slightly pathological, but probably useful
```

```
                #on vampire, tcsh is generally set up with more complete path
        #support, so to find where the current copy of qsub lives,
                #ask for its path via tcsh, and embed the result in the sh command
        #line
                send, recv, err=os.popen3(
                   "`tcsh -c 'which qsub'` -V -N %s -" % (jobName % rundict),'b',0)
                send.write(template % rundict)
        #send keyword-substituted string to qsub as batch file to run
                send.close() #force EOF to trigger qsub action
                jobid=recv.read().strip()
                recv.close()
                errmsg=err.read().strip()
                err.close()
                if (not jobid) and errmsg: #only fail if got a message & no job id
                     print jobtries, errmsg
                     jobtries-=1 #count retries
                     time.sleep(5) #let scheduler logjam clear before trying again
                else:
                     if errmsg: print "qsub warning issued: ", jobid, errmsg
                     joblist.append(jobid)
                     break #end retries on success

      if finalTask:
           jobtries=3
           while jobtries:
                send, recv, err=os.popen3(
                     "`tcsh -c 'which qsub'` -V -N
   %s -W depend=afterany:%s - " % (cleanupName % runvars,
      ':'.join(joblist) ),'b',0)
                send.write(finalTask % runvars)
                send.close() #force EOF to trigger qsub action
                jobid=recv.read().strip()
                recv.close()
                errmsg=err.read().strip()
                err.close()
                if (not jobid) and errmsg: #only fail if got a message & no job id
                     print "CLEANUP", errmsg
                     jobtries-=1
                     time.sleep(5) #let scheduler logjam clear before trying again
                else:
                     if errmsg: print "qsub warning issued on cleanup task: ",
                       jobid, errmsg
                     joblist.append(jobid)
                     break

      if make_killer_script: make_batch_killer(runvars, joblist)

      return runvars, joblist #In case someone wants to use this

def make_batch_killer(runvars=None, joblist=None,
                      killerName="killbatch.%(runName)s.%(isotime)s"):
     "make_batch_killer(...) makes a script file with the appropriate command
      to kill all jobs in a batch"
     f=file(killerName % runvars, "w")
     print >> f, "qdel "," ".join(joblist)
     f.close()

def splitrun(baseCommand=None, baseOutfile=None, sleepTime=1, extra_keys={}):
   """splitrun() splits a run over multiple processors.
      It is superseded by splitrun_binary().
%(isotime)s is replaced with yyyymmdd.hhmmss
%(index)d is replaced with a sub-process index
%(node)s is replaced with the vampire node name on which this process ran."""
     nodelistname=os.environ['PBS_NODEFILE']
     f=open(nodelistname,'r')
     pids=[]
     runinfo=[]
     isotime=time.strftime("%Y%m%d.%H%M%S")
     for index, nodes in enumerate(f):
          node=nodes.strip()
          rundict={'index': index, 'isotime': isotime, 'node':node }
          rundict.update(extra_keys) #pick up all keys from parent, too
          command="%s &> %s"%(baseCommand % rundict, baseOutfile % rundict)
          result=
      os.spawnl(os.P_NOWAIT,"/usr/bin/rsh", "/usr/bin/rsh", node,command)
  #the '-n' option to rsh redirects stdin from /dev/null
          pids.append(result)
          print "starting run on node", node, result, command
          time.sleep(sleepTime)
          runinfo.append((rundict, result, baseCommand % rundict,
                  baseOutfile % rundict))

     f.close()

     while pids:
          done, stat=os.waitpid(-1, 0)
          print done, stat
          pids.remove(done)

     return runinfo

def summarize_histograms(runinfo=None, summary_path='.',
     summary_template="summary.%(isotime)s.aida", filenameKey='outFile',
                    normalize=0, **args):
     """summarize_histograms() takes a runinfo list of the type produced by
     splitrun_binary() and cycles through files in the outFile keyword of the
     commands to find AIDA histograms. It sums all histograms with the same name
     into a master copy, and writes a summary histogram file containing these
     results"""

     import AIDASupport
     import AIDA
     rundict=runinfo[0][0]
     dataTree=AIDASupport.AIDATree(os.path.join(summary_path, summary_template
     % rundict), createNew=1, options="compress=yes")
     histdict={}

     for dict, pid, cmd, logfile in runinfo:
          cmdlist=cmd.split()
  #data file name follow outFile key
          data=cmdlist[cmdlist.index(filenameKey)+1]
          if not os.path.exists(data): continue#process bailed & didn’t leave file
          inputTree=AIDASupport.AIDATree(data, createNew=0, readOnly=1,
                                       options="compress=yes")
          objects=inputTree.listObjectNames(".",0)
          for obj in objects:
               hist=inputTree.findTyped(obj)
               if not hist or not isinstance(hist, AIDA.IBaseHistogram): continue
       #apparently not any recognized histogram
               if not histdict.has_key(obj):
                    histdict[obj]=
       dataTree.GetHistogramFactory().createCopyTyped(obj, hist)
               else:
                    histdict[obj].add(hist)

          #scale the area of the histograms to all the counts that went into it,
  # including overflow and underflow bins
          if normalize:
               for hist in histdict.values():
                    sum=hist.sumAllBinHeights()
                    if sum: hist.scale(1.0/sum) #don’t normalize empty hists!

          inputTree.close()

     dataTree.commit()

     return histdict #in case anyone wants to use it still

def summarize_batch_histograms(runvars):
     """reformat the packed information in the encoded dictionary on command line
        into runinfo-like structure and feed it to summarize_histograms()"""
     runinfo=[]
     for i in xrange(runvars[’nCopies’]):
          runvars[’index’]=i
          runinfo.append((runvars, 0, runvars[’template’]%runvars, None))

     return summarize_histograms(runinfo=runinfo, **runvars)

if __name__==’__main__’:
     import cPickle
     import base64
     #this is a creative way to dispatch an incoming command to various code in
     #this module when it is run as a script
     commands={
          ’split_binary’: splitrun_binary_tree,
          ’summarize_batch’: summarize_batch_histograms
     }
     if len(sys.argv) > 2 and sys.argv[1] in commands.keys():
          runvars=cPickle.loads(base64.decodestring(’’.join(sys.argv[2:])))
  #this should be an encoded dictionary of everything
          commands[sys.argv[1]](runvars)
     else:
          print "don't know what to do with command", sys.argv
```

runed_batch.py

This is the high-level file that tells splitrun.py (above) what to do. It handles the command line arguments. Script was written by Marcus Mendenhall.

```
#!/usr/bin/env python

print "$Id: runed_batch.py,v 1.7 2005/04/06 14:26:45 marcus Exp $"

def getFlag(tag=None, default=None, datatype=int):
     """getFlag() extracts a keyword/value pair from sys.argv (w/o equals sign)
        and returns it, handling type conversion and defaulting. \n
     Datatpe=None returns a boolean which requires no
             value for the keyword, just its presence."""
     try:
          countpos=sys.argv.index(tag)
          gotit=1
     except:
          gotit=0
     if datatype is None:
          if gotit: del sys.argv[countpos]
          result=gotit
     elif gotit:
          result=datatype(sys.argv[countpos+1]) #return converted token
          del sys.argv[countpos:countpos+2]
     else:
          result=default
     print "flag ", tag, "=", result
     return result

import sys
import os

import splitrun

#variable naming here is important... parameters which are used directly as
# keyword arguments by splitrun_batch must have the correct names assigned here,
#since these variables are passed through as a **kwargs dictionary to
#splitrun_batch
runName=getFlag('runName', 'default_output', str)
runTime=getFlag('runTime', 120, int)
nCopies=getFlag('nCopies', 5, int)

#these variables are particular to actual running program, and handled opaquely
 by splitrun_batch
targetFile=getFlag('targetFile', 'Std.Target.BAEA.in', str)
physicsFile=getFlag('physicsFile', 'Std.Physics.in', str)
inputTemplate=getFlag('inputTemplate', 'BatchIn.in', str)
```

```
beamZ=getFlag('beamZ', 1, int)
beamA=getFlag('beamA', 1, int)
biasFactor=getFlag('biasFactor', 200, float)
nIons=getFlag('nIons', 10000, int)
beamEnergy=getFlag('beamEnergy', 100, float)
angle=getFlag('angle', '0', str)
filterEvents=getFlag('filterEvents', 'Std.Filter.in', str)

os.environ["MRED_MMAFILE_DEST_DIR"]=runName

if os.path.exists(runName):
     if not os.path.isdir(runName):
          raise RuntimeError, "Cannot create directory for run data...name exists
                       and is a file"
else:
     os.mkdir(runName, 0750) #create directory group readable, world invisible

template="""
#PBS -l nodes=1
#PBS -l walltime=%(wallTime)d
#PBS -l cput=%(wallTime)d
# All output goes to the same file
#PBS -j oe

cd $PBS_O_WORKDIR
cat %(inputTemplate)s | \
python -c "import sys; import cPickle; import base64;
           print sys.stdin.read() %% %(variablesDict)s" | \
./mr.ed  -s output.%(isotime)s.%(index)03d Interactive.in &>
            %(runName)s/output.%(isotime)s.%(index)03d.txt
"""

#passed_variables is a dictionary of most of our local variables, which will be
#passed as anonymous keyword arguments to splitrun_batch
#if there are local variables you do not want to pass on, delete them from this
# dictionary
passed_variables=locals().copy()
#for example, del passed_variables['foo'] if local variable foo is not useful
 to running program

splitrun.splitrun_batch(
     summary_path= runName , wallTime=runTime+60,
     **passed_variables
)
```

BatchIn.in

This file is a modified MRED input file that accepts arguments given to it by runed_batch.py (see above). This script was modified for use with the above python scripts by Marcus Mendenhall.

```
# Sets some default verbose states
/control/verbose 2
/control/saveHistory
/run/verbose 2
#--------------------------------------------
/control/execute /home/howecl/MrEd/%(physicsFile)s

#--------------------------------------------

/control/execute /home/howecl/MrEd/%(targetFile)s

#---------------------------------------------
# Initializing freezes the physics and geometry for the rest of the run
/run/initialize

#---------------------------------------------
/gun/default

/gun/particle ion
/gun/ion %(beamZ)d %(beamA)d
/gun/energy %(beamEnergy)f  MeV
/gun/randomFlux true

# Set both to false for unidirectional runs
# Set both to true for omnidirectional runs.
/gun/randomIsotropic false
/gun/randomHemisphere false

#---------------------------------------------
/physics/setBiasFactor %(biasFactor)f
/physics/setUseWeighting true
/physics/printBiasFactor
/physics/snr/setMFPScale 500.

#---------------------------------------------

/control/execute /home/howecl/MrEd/%(filterEvents)s

#--------------------------------------------
# These set file outputs on or off
/output/writeSeedFile      false
/output/writeEnergyFile    false
/output/writeTrackFile     false
/output/writeHistogramFile true

#--------------------------------------------
# These control the number of events
/output/progressInterval 100000

/run/beamOn %(nIons)d
exit
```

Interactive.in

This file starts MRED in the interactive mode so it can then accept commands passed to it from BatchIn.in by runed_batch.py.

```
# Command used to send MRED into interactive mode.
/control/tcsh
```

REFERENCES

[1] K. M. Warren, R. A. Weller, M. H. Mendenhall, R. A. Reed, D. R. Ball, C. L. Howe, B. D. Olson, M. L. Alles, L. W. Massengill, R. D. Schrimpf, N. F. Haddad, S. E. Doyle, D. McMorrow, J. S. Melinger, and W. T. Lotshawand, "The contribution of nuclear reactions to single event upset cross-section measurements in a high-density SEU hardened SRAM technology," *IEEE Trans. Nucl. Sci.*, Manuscript in press for the December 2005 issue.

[2] R. A. Reed, P. W. Marshall, H. S. Kim, P. J. McNulty, B. Fodness, T. M. Jordan, R. Reedy, C. Tabbert, M. S. T. Liu, W. Heikkila, S. Buchner, R. Ladbury, and K. A. LaBel, "Evidence of angular effects in proton-induced single-event upsets," *IEEE Trans. Nucl. Sci.*, vol. 49, pp. 3038–3044, Dec. 2002.

[3] R. A. Reed, J. Kinnison, J. C. Pickel, S. Buchner, P. W. Marshall, S. Kniffin, and K. A. LaBel, "Single-event effects ground testing and on-orbit rate prediction methods: the past, present, and future," *IEEE Trans. Nucl. Sci.*, vol. 50, pp. 622–634, June 2003.

[4] O. Musseau, "Charge collection and SEU mechanisms," *Radiat. Phys. Chem.*, vol. 43, pp. 151–163, 1994.

[5] P. E. Dodd and L. W. Massengill, "Basic mechanisms and modeling of single-event upset in digital microelectronics," *IEEE Trans. Nucl. Sci.*, vol. 50, pp. 583–602, June 2003.

[6] J. C. Pickel, "Single-event effects rate prediction," *IEEE Trans. Nucl. Sci.*, vol. 43, pp. 483–495, Apr. 1996.

[7] L. L. House and L. W. Avery, "The Monte Carlo technique applied to radiative transfer," *J Quant Spectrosc Radiat Transf*, vol. 9, pp. 1579–1591, Dec. 1969.

[8] L. W. Avery, L. L. House, and A. Skumanich, "Radiative transport in finite homogeneous cylinders by the Monte Carlo technique," *J. Quant. Spectrosc. Radiat. Transf.*, vol. 9, pp. 519–532, Apr. 1969.

[9] A. Razani, "A Monte Carlo method for radiation transport calculations," *J. Nucl. Sci. Technol.*, vol. 9, pp. 551–554, Sept. 1972.

[10] R. C. Allen *et al.*, *Computational Science Education Project*, 1993. [Online]. Available: http://www.phy.ornl.gov/csep/

[11] C. Jacoboni and L. Reggiani, "The Monte Carlo method for the solution of charge transport in semiconductors with applications to covalent materials," *Rev. Mod. Phys.*, vol. 55, pp. 645–705, July 1983.

[12] H. Duarte, "Improvement of the Intranuclear Cascade code of Bruyéres-le-Châtel (BRIC) at low intermediate energy," in *Proceedings of the 10th International Cconference on Nuclear Reaction Mechanisms*, E. Gadioli, Ed., vol. 122, Varenna, June 2003, pp. 607–616.

[13] P. J. McNulty, W. G. Abdel-Kader, and J. M. Bisgrove, "Methods for calculating SEU rates for bipolar and NMOS circuits," *IEEE Trans. Nucl. Sci.*, vol. 32, pp. 4180–4184, Dec. 1985.

[14] I. Kawrakow and D. W. O. Rogers, *The EGSnrc Code System: Monte Carlo Simulation of Eletron and Photon Transport*, Ionizing Radiation Standards National Research Council of Canada, 2003.

[15] G. A. Wright, E. Shuttleworth, M. J. Grimstone, and A. J. Bird, "The status of the general radiation transport code MCBEND," *Nucl. Instrum. Methods Phys. Res. B*, vol. 213, pp. 162–166, 2004.

[16] J. S. Hendricks *et al.*, *MCNPX Extensions Version 2.5.0*, Los Alamos National Laboratory, 2005.

[17] M. B. Emmett, "MORSE: present capabilities and future directions," *Appl. Radiat. Isot.*, vol. 53, pp. 863–866, July 2000.

[18] C. Inguimbert, S. Duzellier, R. Ecoffet, and J. Bourrieau, "Proton upset rate simulation by a Monte Carlo method: Importance of the elastic scattering mechanism," *IEEE Trans. Nucl. Sci.*, vol. 44, pp. 2243–2249, Dec. 1997.

[19] J. L. Shinn, F. A. Cucinotta, L. C. Simonsen, J. W. Wilson, F. F. Badavi, G. D. Badhwar, J. Miller, C. Zeitlin, L. Heilbronn, R. K. Tripathi, M. S. Clowdsley, J. H. Heinbockel, and M. A. Xapsos, "Validation of a comprehensive space radiation transport code," *IEEE Trans. Nucl. Sci.*, vol. 45, pp. 2711–2719, 1998.

[20] H. Iwase, K. Niita, and T. Nakamura, "Development of general-purpose particle and heavy ion transport Monte Carlo code," *J. Nucl. Sci. Technol.*, vol. 39, pp. 1142–1151, Nov. 2002.

[21] S. Agostinelli *et al.*, "Geant4-a simulation toolkit," *Nucl. Instrum. Methods Phys. Res. A*, vol. 506, pp. 250–303, 2003.

[22] M. H. Mendenhall and R. A. Weller, "An algorithm for computing screened Coulomb scattering in Geant4," *Nucl. Instrum. Methods Phys. Res. A*, vol. 227, pp. 420–430, 2005.

[23] *Physics Reference Manual*, Geant4, 2005. [Online]. Available: http://geant4.web.cern.ch/geant4/

[24] A. S. Kobayashi, D. R. Ball, K. M. Warren, R. A. Reed, M. H. Mendenhall, R. D. Schrimpf, and R. A. Weller, "The effect of metallization layers on single event susceptibility," *IEEE Trans. Nucl. Sci.*, Manuscript in press for the December 2005 issue.

[25] D. R. Ball, K. M. Warren, R. A. Weller, R. A. Reed, A. Kobayashi, J. A. Pellish, M. H. Mendenhall, C. L. Howe, L. W. Massengill, R. D. Schrimpf, and N. F. Haddad, "Simulating nuclear events in a TCAD model of a high-density SEU hardened SRAM technology," *IEEE Trans. Nucl. Sci.*, Submitted for publication in the June 2003 issue.

[26] C. L. Howe, R. A. Weller, R. A. Reed, M. H. Mendenhall, R. D. Schrimpf, K. M. Warren, D. R. Ball, L. W. Massengill, K. A. LaBel, J. W. Howard, Jr., and N. F. Haddad, "Role of heavy-ion nuclear reactions in determining on-orbit single event error rates," *IEEE Trans. Nucl. Sci.*, Manuscript in press for the December 2005 issue.

[27] T. Koi, "Ion transport simulation using Geant4 hadronic physics," presented at Monte Carlo 2005 Topical Meeting, Chattanooga, TN, USA, April 17-21, 2005.

[28] P. Truscott and F. Lei, "Ion-nuclear models for the analysis of radiation shielding and effects (IONMARSE)-contract final report," QinetiQ Ltd, Tech. Rep. QINETIQ/KI/SPACE/CR041585, June 2004.

[29] J. W. Wilson, R. K. Tripathi, F. A. Cucinotta, J. L. Shinn, F. F. Badavi, S. Y. Chun, J. W. Norbury, C. J. Zeitlin, L. Heilbronn, and J. Miller, "NUCFRG2: An evaluation of the semiempirical nuclear fragmentation database," NASA, Tech. Rep. 3533, Oct. 1995.

[30] Wolfram Research, Inc., "Mathematica," Version 5.1, Champaign, IL, 2004.

[31] J. R. Schwank, "Basic mechanisms of radiation effects in the natural space environment," 1994 IEEE Nuclear and Space Radiation Effects Conference Short Course, Tucson, AZ, July 1994.

[32] J. Barth, "Modeling space radiation environments," 1997 IEEE Nuclear and Space Radiation Effects Conference Short Course, Snowmass Village, CO, July 1997.

[33] E. Petersen, "Single-event analysis and prediction," 1997 IEEE Nuclear and Space Radiation Effects Conference Short Course, Snowmass Village, CO, July 1997.

[34] F. W. Sexton, "Measurements of single event phenomenon in devices and ICs," 1992 IEEE Nuclear and Space Radiation Effects Conference Short Course, New Orleans, LA, July 1992.

[35] (1997) Cosmic ray effects on micro electronics website. [Online]. Available: https://creme96.nrl.navy.mil/

[36] P. S. Winokur, "Total-dose radiation effects," 1992 IEEE Nuclear and Space Radiation Effects Conference Short Course, New Orleans, LA, July 1992.

[37] L. W. Massengill, "SEU modeling and prediction techniques," 1993 IEEE Nuclear and Space Radiation Effects Conference Short Course, Snowbird, UT, July 1993.

[38] A. A. Istratov and E. R. Weber, "Physics of copper in silicon," *J. Electrochem. Soc.*, vol. 149, pp. G21–G30, 2005.

[39] A. L. S. Loke, "Process integration issues of low-permittivity dielectrics with copper for high-performance interconnects," Ph.D. dissertation, Stanford University, Stanford, Mar. 1999.

[40] R. A. Reed, P. J. McNulty, W. J. Beauvais, W. G. Abdel-Mader, E. G. Stassinopoulos, and J. Barth, "A simple algorithm for predicting proton SEU rates in space compared to the rates measured on the CRRES satellite," *IEEE Trans. Nucl. Sci.*, vol. 41, pp. 2389–2395, Dec. 1994.

[41] C. P. Poole, Jr. and F. J. Owens, *Introduction to Nanotechnology.* Hoboken, NJ: John Wiley and Sons Inc, 2003.

[42] J. C. Pickel and J. T. Blandford, Jr., "Cosmic ray induced errors in MOS memory cells," *IEEE Trans. Nucl. Sci.*, vol. NS-25, p. 1166, 1978.

[43] J. R. Schwank, M. R. Shaneyfelt, J. Baggio, P. E. Dodd, J. A. Felix, V. Ferlet-Cavrois, P. Paillet, D. Lambert, F. W. Sexton, G. L. Hash, and E. Blackmore, "Effects of particle energy on proton-induced single-event latchup," *IEEE Trans. Nucl. Sci.*, Manuscript in press for the December 2005 issue.