DIRECT IONIZATION-INDUCED TRANSIENT FAULT ANALYSIS FOR COMBINATIONAL LOGIC AND SEQUENTIAL CAPTURE IN DIGITAL INTEGRATED CIRCUITS FOR LIGHTLY-IONIZING ENVIRONMENTS

By

Dolores A. Black

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

In partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

December 2011

Nashville, TN

Approved:

Professor William H. Robinson (Co-Chair) Date

Professor Robert A. Reed (Co-Chair) Date

Professor Gautam Biswas Date

Professor Marcus H. Mendenhall Date

Professor Ronald D. Schrimpf Date

DEDICATION

To my husband, Jeffrey, with whom all things are possible and without whom none of this would have been possible

ACKNOWLEDGEMENTS

"We keep moving forward, opening new doors, and doing new things, because we're curious and curiosity keeps leading us down new paths. You can design and create, and build the most wonderful place in the world. But it takes people to make the dream a reality." – Walt Disney

This work would not have been completed without the support of others. Therefore, I would like to acknowledge those who made this possible.

To begin with I would like to thank my co-chairs Prof. Robert Reed for his relentless encouragement and Prof. William Robinson for his unending patience, and their continued guidance and advice throughout my work. I would like to thank Prof. Ronald Schrimpf for his support and advice in helping me set boundaries and always helping me know that they were achievable. I want to thank Prof. Marcus Mendenhall for his unique and extensive perspective and I appreciate his positive attitude when at times things seemed impossible. I want to thank Prof. Gautam Biswas for his time, knowledge and support for this work. I was able to grow professionally and complete this work because I had an outstanding committee and to all of you I am eternally grateful.

Many thanks are due to Prof. Al Strauss and the NASA Tennessee Space Grant Consortium and Prof. Dan Fleetwood who provided continued support for my education. Also, I would like to thank to NASA/GSFC especially, Ken LaBel, for their sponsorship of this work.

For their help when I needed it, specific acknowledgment is given to Dr. Andrew Sternberg, Dr. Kevin Warren and Dr. Brian Sierawski who mentored me in the use of the different tools and for sharing your knowledge.

I want to send a special thank you to Prof. Robert Weller, Dean George E. Cook and Dean Kenneth Galloway for your continued encouragement in times when I needed it.

Nobody has been more important or supportive in this pursuit than the members of my family. My father and mother who taught me the importance of an education, my sisters who were there with words of encouragement and finally, my husband, Jeffrey, for his unending love and support that got me through from the dark and into the light.

Thanks to you all.

Dolores A. Black

TABLE OF CONTENTS

Page

Appendix

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

Acronym	**Definition**
ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
AVF	Architectural Vulnerability Factor
CRÈME96	Cosmic Ray Effects on Micro-Electronics Code
DUE	Detected Unrecoverable Error
DUT	Device Under Test
ECC	Error Correction Code
EHP	Electron-Hole Pair
FF	Flip-Flop
FPGA	Field Programmable Gate Array
IC	Integrated Circuit
IRPP	Integrated Rectangular Parallelpiped
LET	Linear Energy Transfer
MRED	Monte Carlo Radiative Energy Deposition
MRED2LOGIC	Coupling MRED to SPICE to Logic simulator (ModelSim®) in the tool flow for modeling

MRED2SPICE	Coupling MRED to SPICE in the tool flow for modeling
NSV	Nested Sensitive Volumes
RF	Register File
RPP	Rectangular Parallelpiped
RTL	Register-Transfer-Logic
SDC	Silent Data Corruption
SE	Single Event
SEE	Single Event Effect
SER	Soft Error Rate
SET	Single Event Transient
SEU	Single Event Upset
SKS	Single Kernel Simulator
SPICE2LOGIC	Coupling SPICE to Logic simulator (ModelSim®) in the tool flow for modeling
SV	Sensitive Volume
TCAD	Technology Computer Aided Design
TVF	Timing Vulnerability Factor

CHAPTER I

INTRODUCTION

The number of transistors per integrated circuit (IC) has doubled approximately every two years, as described by Moore's Law [1]. This growth has brought progress in the form of increased performance and functionality in devices ranging from small memory chips to multi-core microprocessors. However, there are obstacles to maintaining this growth rate. Challenges such as power dissipation, reliability, component cost, and yield make it difficult for designs to achieve performance goals [1]. Power reduction, in particular is especially important, and has been addressed by numerous approaches [2-4]. This dissertation focuses on the system reliability issues associated with transient faults resulting from selected ionizing particle events within the IC [5].

Digital integrated circuits fabricated in advanced semiconductor processes are susceptible to single event effects from lightly ionizing particles, e.g., alpha particles, protons, and muons [6-8]. Furthermore, these ICs exhibit complex responses due to interactions with these particles. Simulation of these complex phenomena, from particle interactions to IC responses, is currently possible only through use of multiple, disconnected tools; this method may miss possible errors produced by radiation events and is not efficient. This dissertation describes an integrated technique to model the impact of a single event transient (SET) generated within a single cell on IC response. The technique begins with the detailed simulation of the energy deposition from a variety of radiation events within a single cell and ends with an aggregated prediction of the IC response to the ensemble of events within that cell. This multi-scale simulation approach requires various simulation tools that operate at different levels of abstraction. It integrates well-defined methods to estimate the collected charge, the circuit level response (including transient width, propagation and capture), and the higher-level simulation of the IC response.

This dissertation describes a complete multi-level simulation approach that accounts for: (1) the generation of transients from the basic physical interaction of a single ionizing particle with semiconductor material, (2) the coupling of the ionizing particle to the response of a library cell, and (3) the contribution of that library cell to the overall response of an integrated circuit. Integrating these simulation techniques eliminates the over-estimation of the soft error response that occurs by assuming that every fault included in the error prediction equation [9, 10] is an error.

The multi-scale simulation technique is demonstrated for SETs generated in three combinational logic cells: (1) an inverter, (2) a NAND gate, and (3) a NOR gate. These cells are contained within a specific implementation for an arithmetic logical unit (ALU). In addition, a dual-complementary D flip-flop was also examined. The Monte Carlo Radiative Energy Deposition (MRED) tool [11-15] is used to compute the energy deposition and provide an estimate of the charge generation. The incident particles are restricted to lightly ionizing particles to reduce the significance of more complex charge-collection mechanisms that may be produced by more lightly ionizing particles. A multi-exponential current source is used to translate the deposited charge to an SET waveform generated on a specific node; SPICE is used to determine the corresponding voltage pulse on the same logical cell. Using these tools together allows one to: (1) characterize a single combinational cell (e.g., inverter, NAND gate, or NOR gate), (2) validate these characterizations to experimental data, and (3) characterize the SET pulse-width distributions that result from the deposited charge generated from an MRED simulation. The resulting SET pulse-widths and cell characterizations are used as an input to a digital circuit simulator or IC modeling tool for functional simulations to determine the response of the digital circuit.

The key results from this work are:

1. Using primarily TCAD results to define the inputs, it is shown that MRED coupled with SPICE can be used to compute the cross section for producing an SET for three circuits fabricated in a 90-nm bulk CMOS technology. TCAD results on a 90-nm single transistor are used to define a multi-volume structure and make a first estimate of the charge collection efficiencies. The efficiencies are refined by comparing the simulation result to experimental cross section results using ions with various LETs less than ~10 MeV-cm^2/mg. Only one efficiency is changed for one of the volumes of the NOR, all others remain identical to those estimated by TCAD. With this single small refinement, the tool is able to predict the measured SET cross section for an inverter, a NAND gate, and a NOR gate fabricated in the same technology.

2. Integrating the simulation of energy deposition, charge collection, circuit-level simulation, and IC-level simulation of SET response eliminates the over-estimation of the soft error response caused by assuming that every fault included in the error prediction equation translates to an error. Using the predicted cross-sections of the inverter, NAND gate, and the NOR gate, a distribution of transient pulses is generated for each of those basic cells to enable analysis at the logic level for transient capture. Typically, a worst-case duration (i.e., pulse width) is used for simulations at the IC level. However, a Monte Carlo method is used in this dissertation to show the probability of error based upon incident particles in lightly ionizing environments. Since complex functions can be synthesized from basic gates, a distribution of pulse widths from those gates can be used to analyze larger integrated circuits. This dissertation leaves the analysis of an entire cell library as future work, but a pathway is established that shows the feasibility of determining the error rate for an IC.

Summary of Document

The dissertation is composed of nine additional chapters. Chapter II, Background, introduces the basic concepts that are the building blocks for the remainder of the document. In this chapter, some basic concepts in semiconductor physics and single event response, as well as previous approaches for modeling single events in semiconductors are presented.

Chapter III, Research Overview, provides a brief description of the specific tools used to complete this research. The simulation methodology is presented, as well as a block diagram to illustrate the approach for each step in the tool flow.

Chapter IV, Calibration of MRED to Compute SET Response, explains the charge collection processes that occur when lightly ionizing particles pass through sensitive volumes of the IC. The sensitive volumes are mapped to regions within the selected library cells to determine charge collection. From this information, a sampling is obtained of the deposited charge based on a randomization of the strike location for an ensemble of particles for a specified number of ionizing events.

Chapter V, Modeling Collected Charge in SPICE, provides the building blocks and background for SET modeling at the circuit level. Characterization of library combinational cell SPICE netlists is described, using a new current source model developed through this research. The impact of transistor design characteristics on SET response is described.

Chapter VI, SPICE Circuit Analysis for Data Comparison, details the process to characterize combinational cells from a library. The results of the implementation are reported, i.e., how many SETs are generated, how many are latched, and the error-cross section (cm^2/logical cell) vs. LET (MeV-cm^2/mg). The results are compared to actual experimental data reported by Cannon et al. in 2009 [16]. This chapter demonstrates that simple TCAD results can be used to define MRED sensitive volumes and charge collection efficiencies in a way that enables prediction of SET cross sections.

Chapter VII, Transient Fault Analysis for Sequential Capture in Dual-Complementary Flip-Flops, addresses the process to capture a transient within a storage element. Transient faults can only become visible to the system if they are latched within a storage element. This chapter examines a flip-flop design constructed from NAND gates. A new clock-dependent upset mechanism due to ion strikes internal to the dual-complementary flip-flop is discussed in this chapter. The mechanism prevents the cell from writing new data into the cell.

Chapter VIII, SPICE Circuit Analysis for Logical and Timing Simulations, explains the required inputs necessary to determine if the SET generated will propagate through a complex digital IC. The main consideration is determining the logic depth between registers or memory cells for a target combinational cell. Key results include the SET pulse-width distribution for the combinational cells investigated in this dissertation.

Chapter IX, Complex Digital Circuit Analysis Tool (ModelSim®), explains the use of advanced simulation techniques to combine single kernel simulator (SKS) technology with a unified debug environment for Verilog (IEEE standard 1364-2005), VHDL (IEEE standard 1164-2008), and SystemC designs. This chapter explains the following: (1) the background and basic testing approach, (2) the use of a testbench to exercise all inputs, (3) the necessary functions required by an instantiated complex digital design under test, and (4) the resulting outputs for functional and/or correct outputs. An ALU is the design used for this research. However, this research further advances the technique by using a fault injection library [17] that takes into account the pulse-width distributions resulting from Chapter VIII. The details include the implementation for fault injection and random SET generation per combinational cell from the library, but are not required for the user of the tool flow to adjust. Further details on developing a testbench for this implementation, monitoring the outputs for all SETs generated, and those resulting in errors are described. Finally, analysis for determining the resulting IC soft error rate (SER) for each library combinational cell, as well as for the ALU as a system is discussed. Contributions of the three combinational cells are presented proportional to their usage within the entire ALU design.

Chapter X, Conclusions, summarizes the major contributions of the research and discusses potential future work.

CHAPTER II

BACKGROUND

The basic mechanisms for radiation-induced single-event transients are summarized in this chapter. Also, previous methods to predict the soft errors at the IC level are discussed.

Basic Mechanisms

Overview of Single-Event Effects

A single event (SE) is the interaction of a single ionizing particle with a semiconductor device. It is considered to be a localized interaction that does not depend on the particle flux. During irradiation, an ensemble of SEs is randomly incident both spatially and temporally. The effects produced by an SE are related to the circuit or system response to the radiation event. Single-event effects (SEEs) are often classified as either destructive or non-destructive effects that can lead to permanent (hard) or temporary (soft) faults, respectively [18]. Soft faults that result from single radiation induced transients (known as single event transients or SETs) are the focus of this research.

During an SE, energy is transferred from the particle to bound electrons, promoting them to the conduction band and leaving a track of electron-hole pairs (EHPs) in the semiconductor. Linear energy transfer (LET) is defined as the rate of this energy loss per unit path length, - dE/dx, divided by the density of the target material, resulting in units of MeV-cm^2/mg. If the charge is generated near a reversed-biased p-n junction, then the charge can be collected by the junction. The charge collected by the p-n junction may result in a circuit response to the single ion event. Charge generation deep in the bulk semiconductor region, however, may recombine before it is collected by the junction [19].

Overview of Soft Errors

A soft error, e.g., change in logic state, can be produced in a digital IC if a single ionizing particle passes through a sensitive region of the component. An SE may deposit charge at or near a sensitive p-n junction, producing a current pulse due to the junction collecting the excess charge. The transient responses of the circuit to the current pulse is known as an SET. When an SET from the combinational logic in a circuit appears on the input of the storage cell during a sampling time, it may produce an erroneous response on its output.

The research described in this dissertation is focused on the response of an IC to an SET in a single cell. This includes: (1) the response of the combinational elements, (2) the capture within a flip-flop (e.g., an SET), and (3) the propagation of soft errors in a complex digital IC. This study is restricted to environments dominated by lightly ionizing particles, e.g., protons, muons [7, 20], and alpha particles in order to focus on the integration of the various simulation tools and eliminate the complexities associated with mechanisms like multi-node charge collection that affect more than one cell.

Traditionally, the current resulting from charge collected on a sensitive node is modeled as a double exponential waveform. Messenger developed a model for the SE current pulse as a double exponential given by

$$I(t) = I_0\left(e^{-\alpha t} - e^{-\beta t}\right) \tag{1}$$

where α is the time constant of charge collection and β is the time constant for the dissipation of the collected charge [21]. This type of SE current pulse is shown in Figure 1 [6, 22, 23]. The double-exponential form of the SE charge collection is the most common form used in circuit simulations that utilize SPICE.

Soft error calculations depend on the circuit characteristics, specifically the impact of charge collection, Q_{coll}, for SET pulse-width generation and propagation through a complex IC. In [6, 11, 12, 19, 22, 23] the authors describe methods to connect energy deposition processes to SPICE simulation in order to estimate the shapes of SET pulses. The collected charge is a function of physical conditions like: (1) the ionizing particle's energy, species, and trajectory, (2) silicon substrate structure, (3) doping, and (4) the electric field. In addition, the strike location and the electrical state of the device will factor into the collected charge. Finally, the IC's sensitivity to the collected charge also needs to be considered. This sensitivity defines the critical charge, Q_{crit} (also known as the threshold charge, Q_{thresh}) required to trigger a change in the state of the node [23] and determine if the SET produces an effect [24].

The impact of soft errors on complex digital ICs depends on the specific nature of the error. It can cause either silent data corruption (SDC) or a detected unrecoverable error (DUE) in cases where the error is neither benign or nor corrected [24]. Soft errors can corrupt data, but when the corrupted data does not affect any external output from the circuit, the effect is benign and can be excluded from the SDC category. Corrupted data that has a direct path to a storage cell and eventually results in a visible error to the circuit output is considered a valid SDC event. A DUE event is one in which the system detects the soft error but avoids corruption of the output data. In general, an SDC event is thought to be more significant or harmful than a DUE event, because it causes loss of data, as opposed to a DUE that results in unavailability of the circuit. An SDC event potentially represents a higher-risk for failure than a DUE.

For simple isolated junctions, such as a memory IC like a Dynamic Random Access Memory (DRAM), a soft error will be induced when: (1) an event occurs at a sensitive node, and (2) Q_{coll} is greater than Q_{thresh}. On the other hand, if the event causes Q_{coll} to be less than Q_{thresh}, then the circuit is assumed to be error free. DRAMs are the first devices where soft errors became a noticeable problem, and studies followed that showed other memory devices were susceptible to soft errors [7, 25-28].

As digital ICs become more complex, the combined soft error effects in combinational and sequential elements are important for a system level error analysis. The sequential elements are the final element in the hardware chain that determines whether or not a fault manifests as an error. Complex digital ICs usually include a software component; however, this study is limited to effects at the hardware level. Manufacturing defects, process imperfections, or interactions with the environment can cause hardware faults. Faults in digital ICs can be classified as permanent (i.e., remain indefinitely until corrective action is taken), intermittent (i.e., appear, disappear, and reappear again), or transient (i.e., appear and disappear in the form of bit flips or gate malfunction from an ion strike) [29].

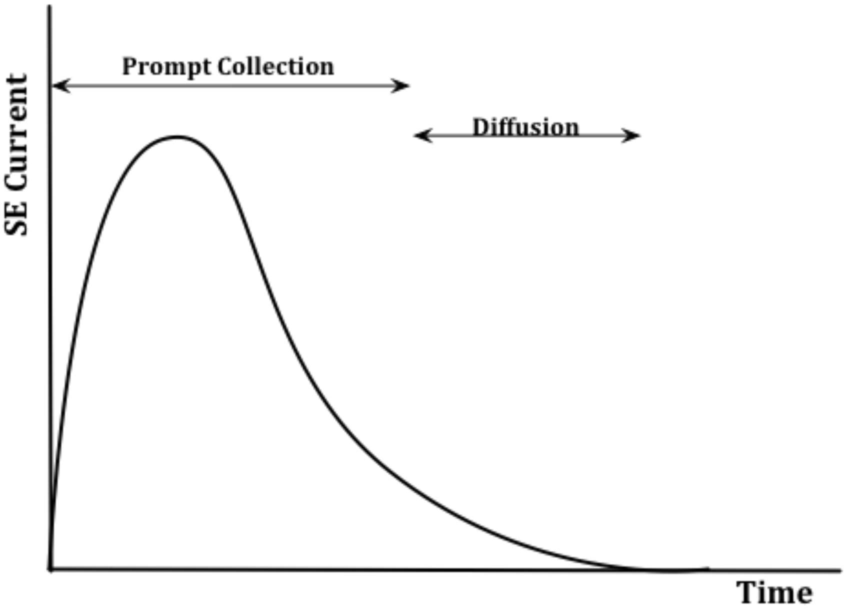


Figure 1. Typical shape of nodal current at a junction

Previous Approaches to IC Single Event Analysis

Recent methods to model soft errors have been proposed that incorporate various masking factors (i.e., electrical, logical, and latch-window) that affect whether a fault ultimately appears as an error in the IC or not. Once a transient is captured in a memory element, the effects can be analyzed like a single event upset (SEU) in a memory element (i.e., an erroneous bit flip). This section reviews several of the more prevalent methods.

SEU_Tool

SEU_Tool was developed to analyze the contribution of combinational logic in the path to a sequential or memory element [9]. This flow is depicted in Figure 2 and shows the steps to predict the transient rate of combinational logic. This tool uses parameterized closed-form circuit models for transient pulse generation, a structural VHDL logic-level simulation for pulse attenuation and propagation, a probabilistic model for transient capture, and a second high-level VHDL logic simulation for bit-error observability. In addition to circuit modeling calculations, this method also contains algorithms at various steps to identify the worst-case contributors to soft errors, which reduces computation time. Given the detail of modeling capability in this method, its accuracy is largely a function of the quality and completeness of the parameters used for input [9, 30].

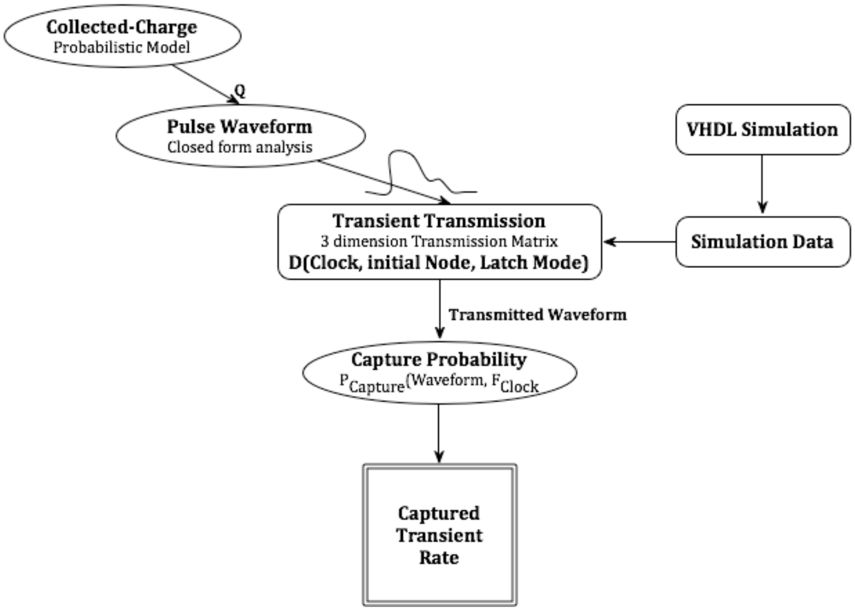


Figure 2. SEU_Tool operations flow chart[9]

SEU_Tool has two parts of the soft error assessment. First, the probability is calculated for each node that causes a soft fault in the circuit system. There is always a chance that the system might not be affected by the change in state of that single bit. SEU_Tool also considers the observability of the soft error in the system output [9, 30].

Intel Method

In [31], Seifert et al. emphasized that the SER of modern microprocessors with large caches or large memory arrays are usually protected with an error correction code (ECC) and therefore the failure rate of the device is dominated by the contribution of sequential elements. Equation (2) can be used to estimate chip level SER for the nodes within the circuit [31]:

$$SER^{circuit} = \sum_{i}^{all\ nodes} SER_i^{nominal}\ x\ TVF_i\ x\ AVF_i \qquad (2)$$

where the nominal *SER^nominal* is the un-derated SER and is independent of the circuit environment, *TVF* refers to the timing vulnerability factor, and *AVF* refers to the architectural vulnerability factor. The *TVF* is defined as the fraction of time a storage element is susceptible to upsets, and *AVF* is equal to the probability that a fault in the storage element will be observed at the output [24]. Mukherjee et al. have developed a methodology that is well understood and accepted for calculating *AVF* and *TVF,* independently [32]. The *AVF* and *TVF* will be contributing factors to soft error analysis, and the methods used to calculate them were considered for this research. The methodologies and techniques identified in Seifert et al. regarding the sequential elements were also considered when implementing the multi-scale simulation approach. This dissertation aimed to include all the factors in a holistic approach to determine the contributions of nodes to the soft error rate. The approach used the same decomposition to determine AVF and TVF.

Other Notable Techniques for IC Soft Error Analysis

Other tools that are used to calculate impact of soft error are listed below, along with a short description of their contribution. A detailed description can be found in the publications noted. The publications are listed chronologically, from earliest to the most current.

Soft-Error Tolerance Analysis and Optimization of Nanometer Circuits [33]

Dhillon et al. presented tools for the analysis and optimization of soft-error tolerance of nanometer combinational circuits. The authors asserted the ability of these tools to calculate accurately the "unreliability" of circuits with less computational time than that of SPICE [33]. Since the focus of the research discussed in this dissertation implements a multi-scale simulation approach using SPICE, the tools identified in this publication are not applicable. However, this multi-scale simulation approach includes all masking factors demonstrated in the paper.

Soft-Error-Rate-Analysis (SERA) Methodology [34]

This publication by Zhang et al. takes into account various approaches for circuit and fault simulation and analysis for probability and graph theory [34]. The authors assert they achieve a higher level of magnitude for speed-up over Monte Carlo-based simulation approaches.

SEAT-LA: A Soft Error Analysis tool for Combinational Logic [35]

Soft Error Analysis Tool – Logic Analyzer tool was developed for a quick and accurate prediction of SER in combinational circuits with the ability to capture the three masking effects concurrently [35] is discussed by Rajaraman et al. The methodology used for the development of this tool used logic cell characterization and flip-flop characterization similar to the techniques in this dissertation. Their modeling of the voltage glitch propagation, however, was done purely analytically by the use of mathematical equations assuming a triangular or trapezoidal pulse, while this dissertation models an SET as a double exponential voltage and simulates it as it propagates through the circuit via SPICE.

Circuit Reliability Analysis Using Symbolic Techniques [36]

In [36], Miskov-Zivanov et al., discuss a purely analytical model using binary decision diagrams (BDD) and algebraic decision diagrams (ADD) for a unified symbolic analysis for circuit reliability. There are some important differences between the dissertation research and the publication by Miskov-Zivanov et al. While both techniques review and take into account all forms of masking, i.e., logical, electrical, and latch-window, this publication treats them as dependent on one another as they apply to a specific design and feeds into the BDD and ADD decision trees. For the current work, each element is evaluated for its contribution to the overall design.

Modeling and Optimization for Soft-Error Reliability of Sequential Circuits [37]

This publication is follow-on work by Miskov-Zivanov et al. that takes into account the sequential elements not presented in their previous work. Like its predecessor publication, it is a purely symbolic approach for efficient estimation of the soft error susceptibility of sequential circuits [37]. Two methods were compared in this paper, a Markov chain (MC) method and the binary decision and algebraic decision diagram (BDD/ADD) method mentioned in the previous section. It was shown that the MC approach could only provide steady-state behavior information, but the BDD/ADD could be done on both transient and steady-state effects. This paper, like its predecessor is entirely a symbolic model that is mathematically intensive using BDD/ADDs, but was verified to a Markov Chain and HSPICE™ circuit simulation.

Summary

The multi-scale simulation approach described in this dissertation provides the framework to consider all aspects that contribute to soft errors, including charge deposition, circuit simulation, and IC simulation. Previous methods made crude assumptions about energy deposition from radiation transport. Integration with MRED enables improved accuracy in the energy deposition calculations and the resulting charge collection. Coupling with SPICE enables the statistical distribution of pulse widths to be considered instead of the fixed pulse width used in previous methods. Finally, the IC simulation provides the framework to study the contributions of individual cells based upon their usage within a synthesized design.

CHAPTER III

OVERVIEW OF MULTI-SCALE SIMULATION OF SINGLE EVENT TRANSIENTS

This chapter provides an overview of the multi-scale simulation approach developed during this research. The chapter briefly discusses the connection of previous soft error prediction methods for storage elements to the research topic of this thesis (which includes combinational cells).

Overview of the Multi-Scale Simulation Approach

Most research on soft error predictions prior to 1990 was conducted on static memory elements only [38], static analysis of logic elements, and flip-flops (FFs). The more recent research, SEU_Tool, presented the effects of clock-dependent (i.e., dynamic) soft errors in logic elements and FFs [9]. Typically, these soft-error, static-upset predictions were calculated using tools that were limited to the assumption that the entire drain area was sensitive [13].

This work expands this concept significantly to include SET-induced soft errors observed at the complex digital IC output. This requires coupling of three tools: (1) radiation transport and charge collection estimates (labeled MRED process in Figure 4), (2) circuit level simulation (SPICE process), and (3) high-level IC simulation (IC Modeling Tool). *Decision Point 1* (identified in red in Figure 3), between the MRED process and the SPICE process indicates whether or not the charge deposited was large enough to generate a sufficient current pulse. If not, then this event is assumed to be negligible and the next MRED event is evaluated. These data are stored in a data array where each specific event from MRED is always associated to the current source it generates. The key contribution of this work is to develop and demonstrate a method of predicting SET cross-sections (or the number of transients per particle fluence) using MRED coupled with SPICE.

A link between SPICE and the IC modeling tool requires that the generated SET be represented by a compatible form, specifically a digital signal equal to a logic 1 or logic 0, and inherit the pulse duration from the SPICE output. Therefore, the SETs are converted to equivalent-rail-to-rail voltage signals. *Decision Point 2* (Figure 3), indicates whether or not the SET pulse has sufficient duration to be propagated to the IC modeling tool for further simulation. If not, then the next SET pulse is evaluated. Each specific soft error can be traced back to a specific event from MRED. The analysis takes place from the viewpoint of individual library cells. The key contribution of this work demonstrates the coupling of specific particle strikes to a latched error.

Once the signals are in the form that ModelSim® uses, the tool analyzes SET propagation through the combinational logic to memory elements to identify if errors occur at the outputs of the IC (Figure 3). ModelSim® was used for this research because of its availability, but similar IC modeling tools could also be used.

The integrated simulation approach for soft error analysis encompasses the spatial (energy deposition to charge generation), timing (current pulse capture) and logic (pulse propagation) vulnerabilities all in one multi-scale simulation approach. The remainder of this dissertation refers to the energy-to-charge process as MRED, circuit-level simulation as SPICE, and IC-level simulation as IC Modeling.

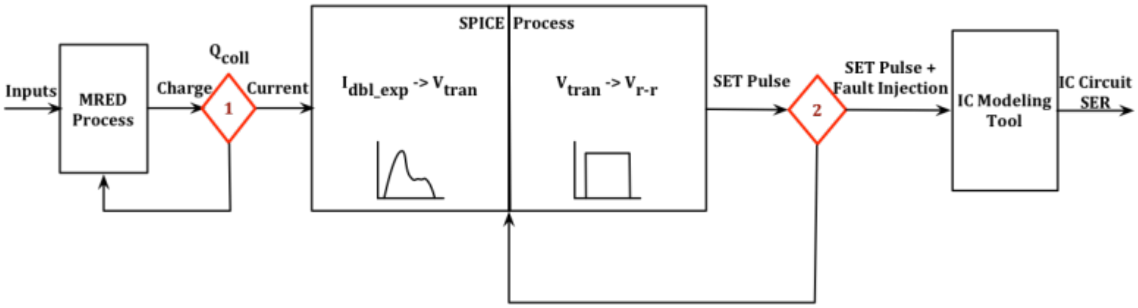


Figure 3. Multi-scale simulation approach black box block diagram

Overview of Energy Deposition and Charge Collection Processes (MRED)

Researchers at Vanderbilt have developed a tool for predicting soft errors that uses Monte Carlo radiation transport techniques along with complex sensitive volumes [14, 15] called MRED (this tool is briefly described below and in more detail in Chapter IV). Previous research described methods for coupling MRED to SPICE for prediction SEU for space and terrestrial environments [11, 12]. The MRED method for SEU prediction is a Monte Carlo numerical integration of a set of general equations [13]. The power of this approach is that it remains tractable in the absence of simplifying assumptions, and therefore in principle, it is more precise and accurate to predict errors [12, 13, 15]. One key advantage of migrating from the typical RPP or Integrated RPP (IRPP) analysis to MRED is the capability to consider charge collection as opposed to charge generation. Two main capabilities that have been implemented recently in MRED are composite sensitive volume models and multiple sensitive volumes. Composite sensitive volume models are used to relate deposited energy to collected charge [11, 12]. These capabilities enable a more physical representation of the charge collection process resulting from a single event. MRED also enables the use of multiple sensitive volumes to model multiple node charge collection, which cannot be considered in the traditional RPP/IRPP analysis. (The work described in this dissertation does not consider multi-node charge collection directly because of the limited ability to model these effects, but if models were developed, then they could be integrated into the approach described in this thesis.)

MRED can be used to estimate the collected charge, $Q_{coll,}$ from an ionizing radiation event by defining these inputs to MRED: (1) the ionizing particle's energy, species, and trajectory, (2) sensitive volume size, locations, and charge collection efficiency (3) the strike location of the ion within the specified combinational cell, and (4) the number of events to execute. During a MRED run, the beam is randomized over strike location. The output of the run is the collected charge (defined by the energy deposited in the volume and its charge collection efficiency) for each volume for each event. A basic block diagram of this process is seen in Figure 4 with: (1) Q_{coll} defined as charge collected, (2) i defined as MRED event number, (3) j defined as the circuit node number, (4) k defined as the individual sensitive volume, (5) α defined as the sensitive volume efficiency, and (5) E_{dep} defined as the energy deposited in the individual sensitive volume.

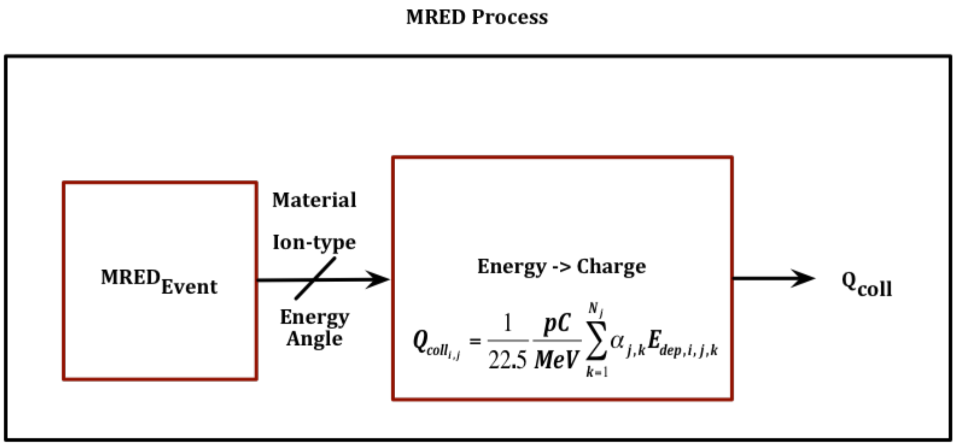


Figure 4. MRED process basic block diagram

Overview of Circuit Simulation (SPICE Process)

The MRED process provides the input to the SPICE process where the conversion from collected charge to the node current model is calculated for each cell. The output charge for each ion strike event, Q_{coll}, from MRED is converted to a current source (I_{event}) (Figure 5).

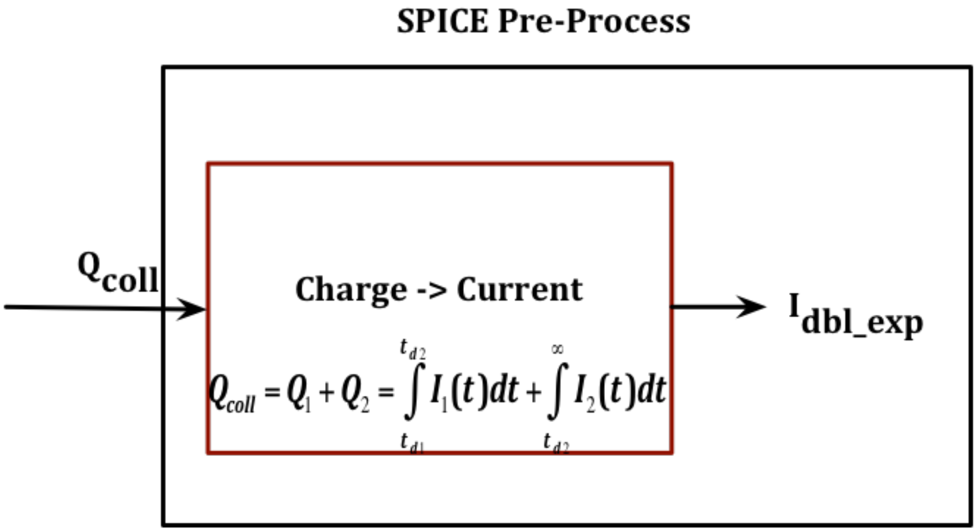

SPICE Pre-Process
Q_{coll}
Charge -> Current
$Q_{coll} = Q_1 + Q_2 = \int_{t_{d1}}^{t_{d2}} I_1(t)dt + \int_{t_{d2}}^{\infty} I_2(t)dt$
I_{dbl_exp}

Figure 5. SPICE pre-process basic block diagram

After the I_{event} current source models are calculated, then conversion to the associated transient voltage, V_{tran}, is simulated. The V_{tran} is then passed through a random number of stages, *n*, of the corresponding combinational cell (Figure 6) using a SPICE circuit netlist, and filtered into an equivalent rail-to-rail voltage, $V_{r\text{-}r}$. The $V_{r\text{-}r}$ is now a new model for the SET pulse and is a typical digital transitioning signal – logic 1 to logic 0 and vice versa. Figure 6 shows V_{tran} as an arbitrary input voltage for illustration only.

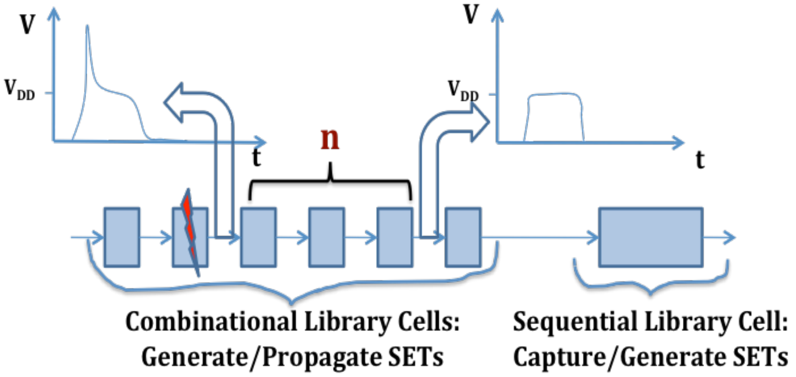


Figure 6. Conversion of V_{tran} to $V_{r\text{-}r}$ after a random number of stages of combinational library cells

Multiple cases of the SET pulse generation and propagation through the combinational cells must be considered to determine the $V_{r\text{-}r}$ signal. This helps to determine the number of subsequent cells that are required to convert the V_{tran} signal into a square pulse and determine the equivalent rail-to-rail voltage transient (called an SET pulse). The $V_{r\text{-}r}$ is also simulated and analyzed in SPICE for the associated full-width, half-maximum rail-to-rail voltage output that produces a particular SET pulse with a given duration. An ensemble of SETs is generated for an ensemble of radiation events; each individual SET in this ensemble is traceable back to an individual radiation event with a specific Q_{coll} (Figure 7).

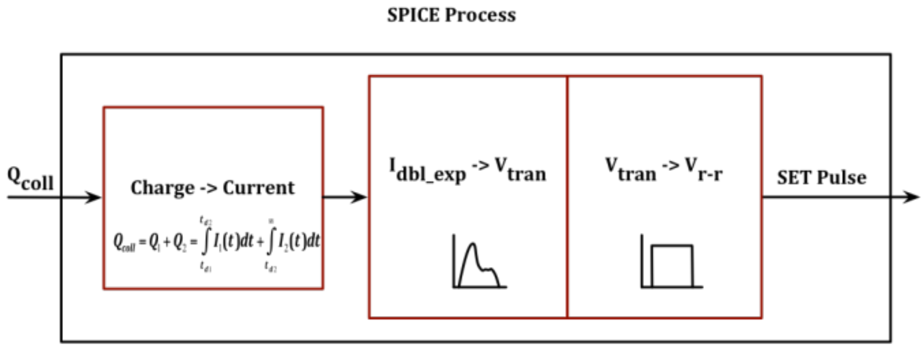

SPICE Process
Q_{coll}
Charge -> Current
$Q_{coll} = Q_1 + Q_2 = \int_{t_{d1}}^{t_{d2}} I_1(t)dt + \int_{t_{d2}}^{\infty} I_2(t)dt$
I_{dbl_exp} -> V_{tran}
V_{tran} -> $V_{r\text{-}r}$
SET Pulse

Figure 7. SPICE process complete block diagram

Overview of IC Modeling Tool - ModelSim®

The SPICE analysis converts Q_{coll} to V_{tran} then to $V_{r\text{-}r}$ so that it can be used effectively as an input to the IC simulation tool, ModelSim®, using logic 1's and 0's. The benefit of integrating ModelSim® with MRED and SPICE is the ability to have a multi-scale simulation comprised of radiation transport to circuit-level simulation to IC-level simulation. An ALU is used as an example circuit. The simulation uses both combinational elements and storage elements to trace the SET through each stage of the circuit.

ModelSim® simulates the execution of an operational circuit via a testbench and determines if the generated SET results in a fault and eventually a soft error. Soft errors are the manifestation of the faults. That is, not all faults show up as errors (i.e., some faults are benign), but errors can lead to SDC or DUE. Figure 8 illustrates that a fault within a specific scope may or may not show up as an error to the outer scope if the fault is masked [24].

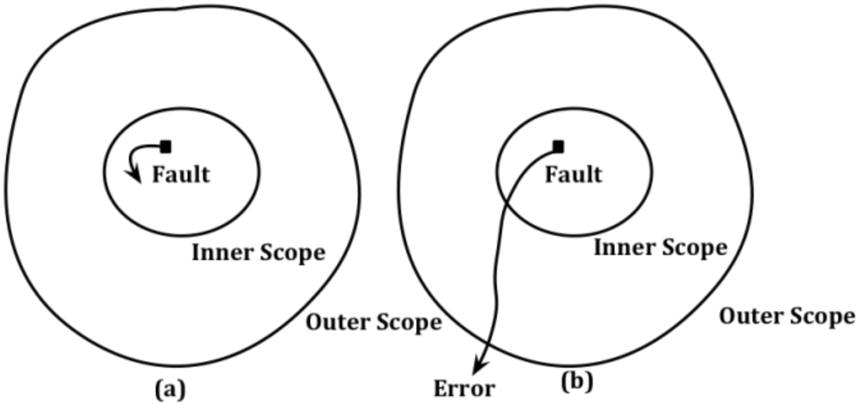


Figure 8. (a) Fault within the inner scope masked and not visible to an IC output (b) Fault propagated outside the inner scope to the outer scope and visible as a soft error to the output of the IC. [24]

If the fault makes it through the first stages of simulation of the combinational elements and migrates through to a sequential or memory element, then the resulting error can be verified by monitoring the output through the testbench. This task is accomplished by running an operational circuit and a functional testbench in this multi-scale simulation approach.

Verification is a process used to demonstrate the functional correctness of a design. A block diagram of an IC design using a testbench (Figure 9) shows how the testbench interacts with the design under verification.

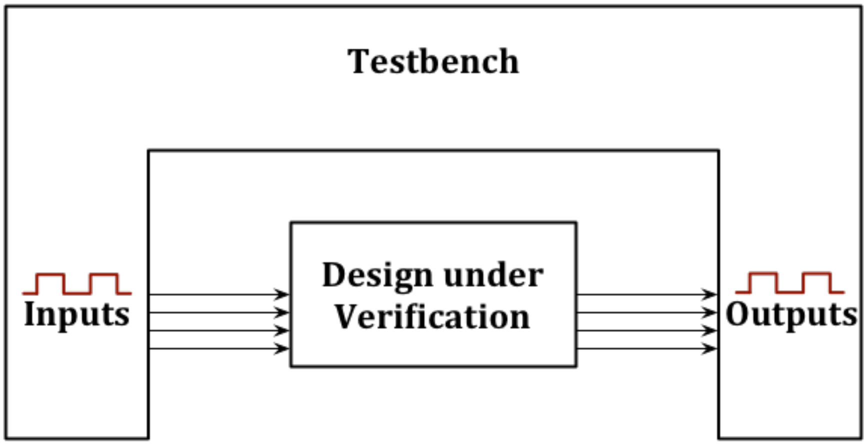


Figure 9. Generic structure of a testbench and an IC design under verification

The term testbench usually refers to the code used to provide a pre-determined input sequence to a design and then to observe the response. The testbench is a completely closed system. When using higher-level IC modeling tools, the testbench is effectively a model of the entire design. The verification challenge is to determine what input patterns to supply to the design and what output patterns should be expected from a properly working design [39].

Languages such as VHDL or Verilog are used to implement the testbench wrappers, fault injection and stimuli for ModelSim®. All input signals are expected to be logic 1 or logic 0. The SET pulse from the SPICE analysis is an equivalent rail-to-rail voltage (1 or 0).

The inputs for the design under verification are the SET pulses generated for the characterized cells. These SET pulses form a distribution of pulse widths to be used in conjunction with a fault injection library [17], which is described in detail in Chapter IX. This procedure allows for both randomization of an SET pulse-width and selection of the corresponding combinational library cell to strike. The ALU circuit is monitored through a comparator in the testbench for a soft error. The circuit is verified at clock speed so that the resulting error contributions from individual cells take into account the dynamic operation of the circuit. This process produces a soft error analysis for each cell that does not differentiate between logical-masked or timing-masked errors for a full IC circuit, mimicking a true experiment. This process is the first method to demonstrate soft error contributions from individual cells with direct traceability to particle strikes from the specified environment.

Benefits of Multi-Scale Simulation Approach

Integrating simulations of energy deposition, charge collection, circuit response, and IC functionality to compute the overall SET response eliminates over-estimation of the soft error rate caused when one assumes that every fault included in the error prediction equation [9, 10] is an error. The multi-scale simulation approach uses the various tools to identify the different vulnerabilities due to spatial energy deposition and charge collection, timing of pulse capture, and propagation of pulse through the logic for complex digital ICs. A more refined soft error analysis that counts only those necessary electrical, latch-window, or logical-masked errors can be determined using this process (Figure 10).

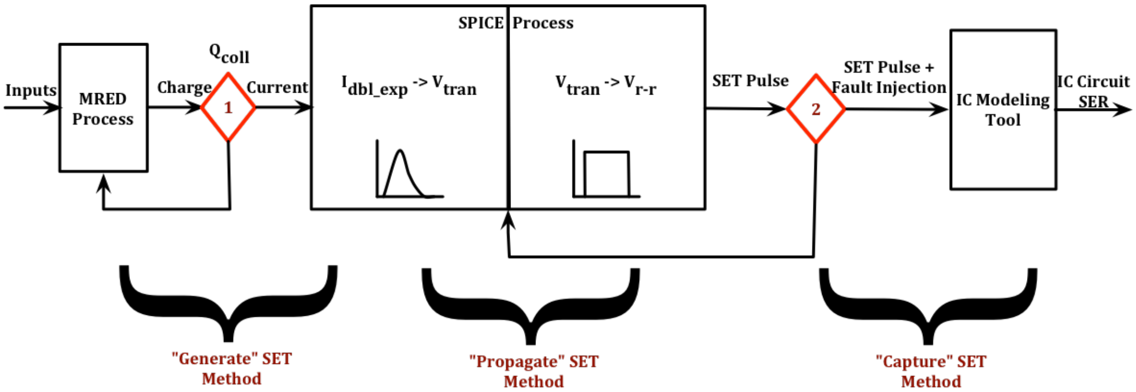


Figure 10. Multi-scale simulation for generation, propagation and capture of an SET

CHAPTER IV

CALIBRATION OF MRED TO COMPUTE SET RESPONSE

This chapter describes preliminary calibration of MRED to simulate the energy deposition and charge collection processes important for SET prediction in three logic cells. Calibration of nested sensitive charge-collection volumes for SETs in logic cells is described. The chapter concludes with an example of how to develop nested charge-collection volumes for an inverter cell and examples of outputs from MRED simulations. Chapter VI presents the methods used to refine this initial guess using experimental data.

Charge Collection Simulation using MRED

Background

Circuits designed in a 90-nm IBM Complementary Metal-Oxide-Semiconductor (CMOS) process were selected to demonstrate the multi-scale simulation approach. The methods used to estimate the collected charge from energy deposited by a radiation event begin with TCAD simulations performed on the IBM 90-nm bulk transistors [6]. Warren, et. al, used these simulations to support

MRED prediction of the single event upset response of radiation-hardened FFs. The accuracy depends upon adjusting the charge collection efficiency of each volume so that predictions agree with experimental data. The research in this dissertation uses the same TCAD results to calibrate SET pulse-widths predicted by coupling MRED to SPICE against measured data for various combinational library cell elements of an ALU fabricated in the IMB 90 nm process.

Figure 11 and Figure 12 [11] provide the results of the TCAD simulations for particles with an LET of approximately 10 MeV-cm^2/mg. Figure 11 shows the resulting nested sensitive volumes for the 90-nm NMOSFET. At this LET, TCAD predicts the charge collection efficiency in the active area of the transistor as 100% (this is the drain/source region above the well). The figure also shows the set of nested sensitive volumes associated with the well structure. The volume nearest the drain has an efficiency of 54%. The collected charge drops rapidly as the volumes get farther from the active region, this is shown by the change in coloring in the figure going from red to blue. The charge collection efficiency decreases rapidly as the distance increases from the active area. Figure 12 shows the TCAD results for PMOSFETs simulated with the same LET. The efficiency in the active area is 80%. The charge collection efficiency of the next region in the well is 25%, and it decreases as the volumes are farther from the active region.

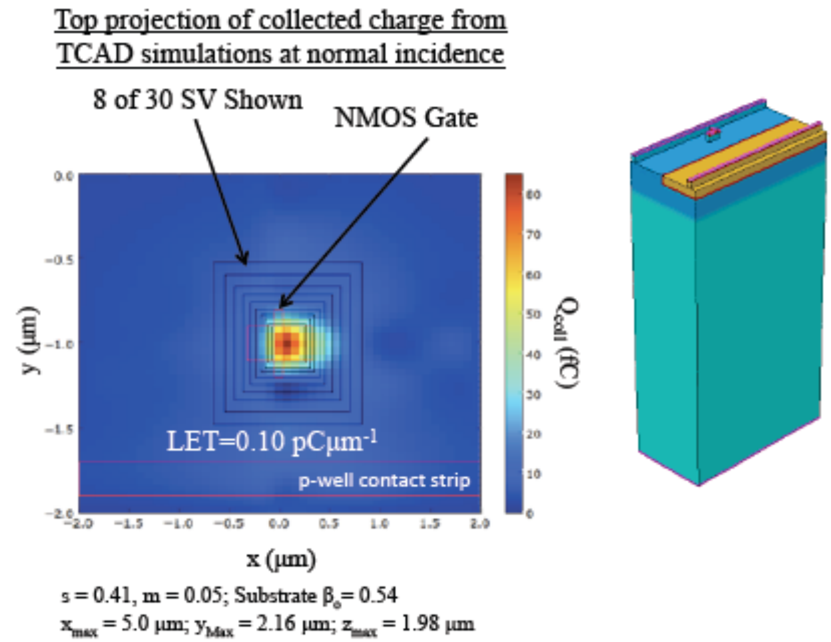


Figure 11. TCAD-generated spatial distribution of the collected charge as a function of strike location with 8 of 30 sensitive volumes (SV) drawn for the single node NMOS device at 0.1 pC/mm (top down view). [11]

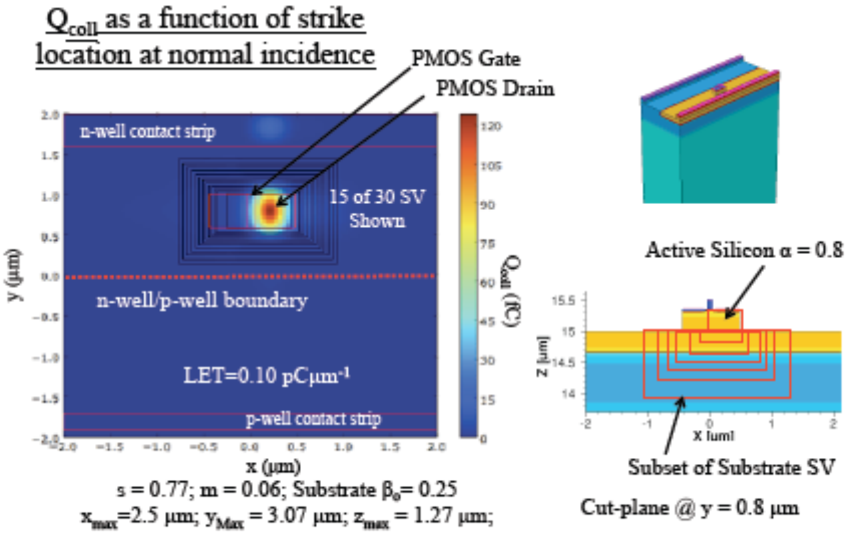


Figure 12. Spatial distribution of the collected charge as a function of strike location for the single node PMOS device at 0.1 pC/mm (top down view). [11]

Nested Sensitive Volumes from Library Cell Layouts

The bulk of the charge collection for lightly ionizing particle events is due to charge generated in the active area and nearby in the well. So, the restriction of the environment to lightly ionizing particles allows decreasing the number of volumes necessary for simulation over that used in [6]. Four volumes are chosen to represent the composite sensitive volume: two in the active area and two in the well. Each pair is nested, but the active volumes are kept distinct from the well volumes.

The dimensions of the charge-collection volumes are determined from the layouts of the cells. A top view of a transistor, either NMOSFET or PMOSFET, is shown in Figure 13. The source and drain are shown in blue, and together (along with the region below the gate) they form the active area of the transistor. The figure also shows the well that surrounds the transistor. In the figure, the thick lines at the top and bottom represent the well boundary, while the absence of thick lines on the left and right represent the fact that the well extends over large distances in those directions. The well contact is also shown in Figure 13, but its location is not relevant for lightly ionizing particles. (Note that the size and location of the well contact is relevant for highly ionizing particles [40].) The side view of the CMOS transistor is shown in Figure 14. This view is referenced along the cut-line, *X*, in Figure 13. The active region is the source and drain regions down to the top of the well. The active region is surrounded by trench oxide, charge generated in this region does not contribute to charge collected by the drain.

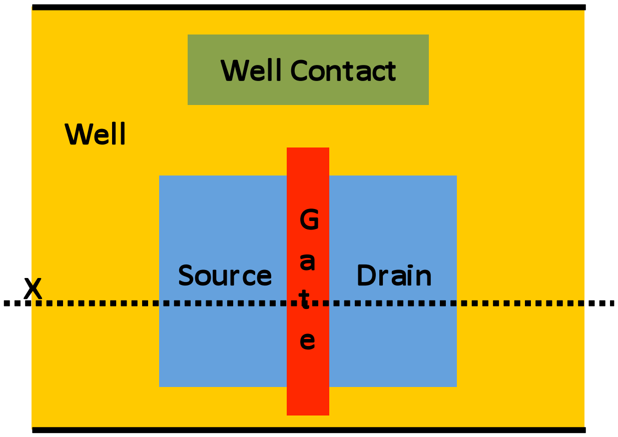


Figure 13. Top view of CMOS transistor that is representative of layout

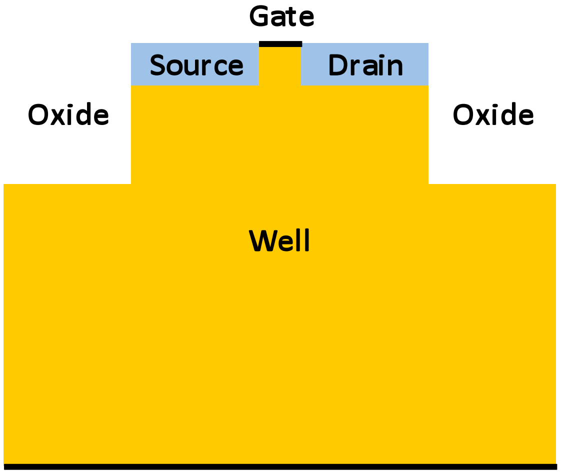


Figure 14. Side view of CMOS transistor from cut along the line X in Figure 13

Figure 15 and Figure 16 show the top and side views respectively of the four charge-collection volumes for the basic transistor, two of which are in the active region and two that are in the well. The two volumes in the active region are called drain and src_drain for this discussion. The drain volume is the most efficient charge collection volume and is defined by the *x-y* plane in the layout of the drain. This is drawn on the right of the figures with the darkest blue. The depth of this charge collection volume is equal to the thickness of the trench oxide, which is 0.35 μm for this process. The src_drain volume consists of the whole opening of the trench oxide and also extends down to the top of the well. This is drawn on the right of the figures with the second darkest blue. It is noted that the src_drain volume contains the drain volume in its entirety. The two well volumes are called well-1 and well-2. The well-1 volume is formed by starting under the drain volume and extending out in the positive and negative *x* directions in the figure by one half of the lateral length of the drain. These extensions of the positive and negative *x* direction are consistent with the TCAD simulations in Warren [11]. The well-1 volume extends in the positive and negative *y*-directions by a constant width, since the charge collection is bound by the well boundaries in that dimension. Well-1 extends 0.3 μm down into well, and that distance is based upon the TCAD simulations [11]. The well-2 volume starts with the active area, includes the well-1 volume, and extends out in the positive and negative *x*-directions another one-half of the lateral drain length. It extends a constant amount in the positive and negative *y*-directions. Finally, it extends 0.4 μm down into the well and that distance is based upon TCAD simulations [11]. The well-2 volume is the least efficient in terms of charge collection.

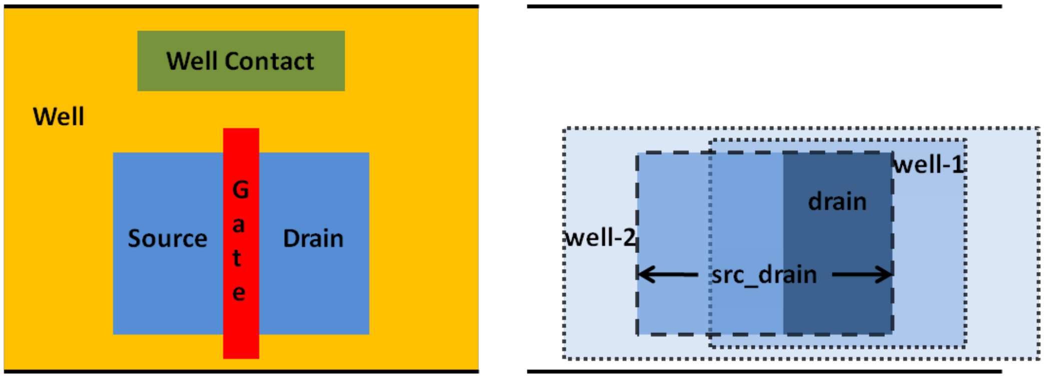


Figure 15. Top view of CMOS transistor with top view of charge-collection volumes

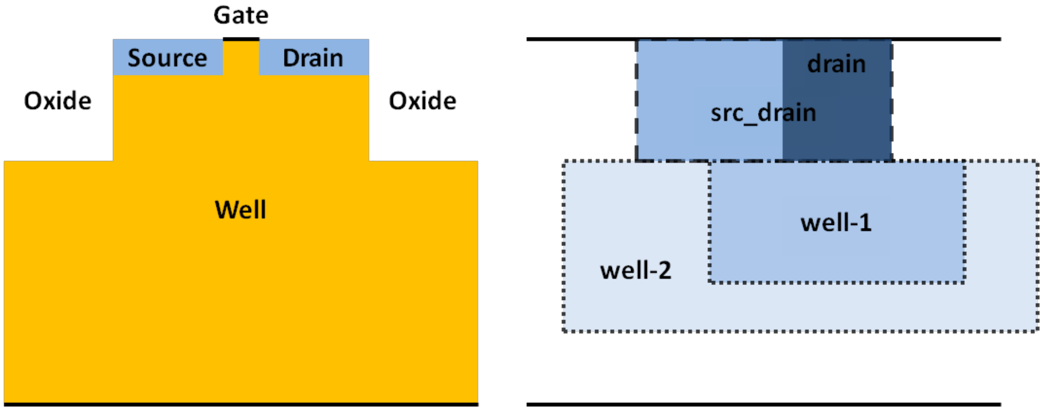


Figure 16. Side view of CMOS transistor with side view of charge-collection volumes

There are a couple of variations on the basic CMOS transistor that must be included in order to build charge-collection volumes for any combinational cell of a library. Transistor designs can either be connected in series or in parallel. For parallel connections, there are a couple of ways this can be handled, and these are depicted in Figure 17 and Figure 18. Figure 17 shows two CMOS transistors in parallel with a shared drain in the middle. Charge-collection volumes for these parallel transistors are similar to the basic transistor. Figure 18 shows the common source for the parallel transistors located in the middle and the shared drain on each side. Though not shown in the figure, metal wires would short the shared drain regions. An additional drain volume is required for this type of parallel connection. Two well-1 volumes are also required beneath each shared drain. However, the extension from each drain in the positive and negative *x*-directions may revert the two well-1 volumes back to a single well-1 volume. Figure 19 shows the series connection of CMOS transistors. With series connections, a new volume must be introduced for the intermediate drain. The intermediate drain may collect charge like the drain or the source depending on the bias in the intermediate drain. This intermediate drain bias will vary depending on input conditions. An additional well-1 volume must also be included, and its efficiency will likewise depend upon the bias of the intermediate drain.

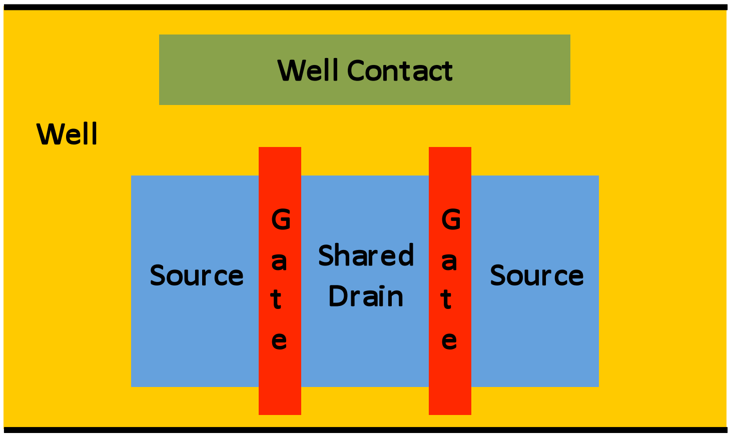


Figure 17. Top view of two CMOS transistors connected in parallel with a shared drain in the middle

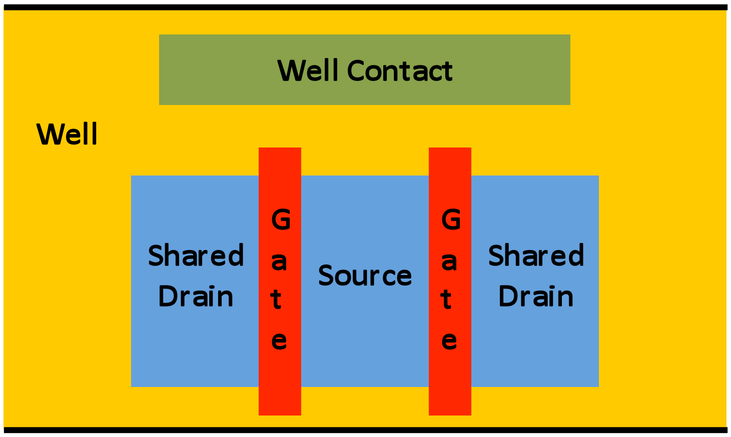


Figure 18. Top View of two CMOS transistors connected in parallel with a shared drain on the outside. Metal lines connecting the shared drain are not shown.

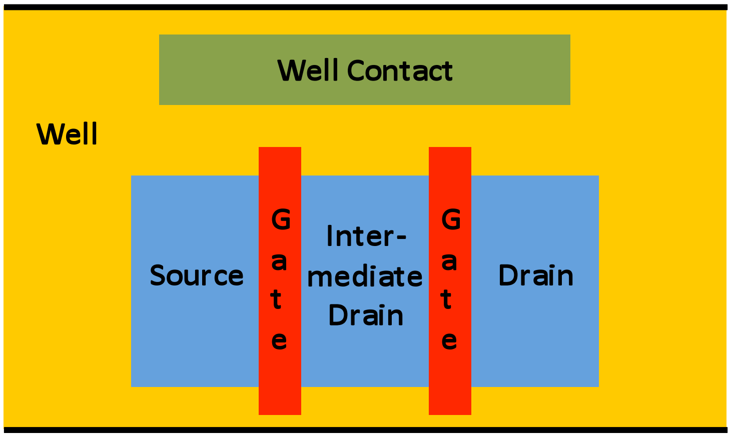


Figure 19. Top view of two CMOS transistors connected is series with an intermediate drain

Example Geometry of the Nested Sensitive Volumes In MRED

MRED uses a Python script (Appendix A) to define various inputs including: (1) the radiation environment (particle species, energies, and directions), (2) the location and size of the sensitive volumes, (3) the number of particles to be simulated, and (4) a threshold energy (or charge) for recording of events per MRED run. The application of the charge collection efficiency per sensitive volume can occur either in the MRED Python script or in the pre-processing Python script for SPICE; both methods were utilized for this research. One was used for charge calibration (MRED only), and the other was used for charge conversion (SPICE only).

Sensitive volumes are defined in Python with a vector pointing to the center and a three-dimensional size. Figure 20 was extracted from the layout of the basic inverter cell (INV) in the 90-nm library and will be used to demonstrate how to obtain the sensitive volume center and size in two-dimensions. The (x, y) coordinates of each drain are shown on the left (Drain Area), and the (x, y) coordinates of the active area are shown on the right (Src_Drain Area). All coordinates are in μm. From these coordinates, the four sensitive volumes for each transistor are determined. For example, the NMOSFET drain is bounded by the coordinates (0.46, 0.94) and (0.72, 1.22). The sensitive volume center for the NMOSFET drain in two-dimensions is (0.59, 1.08) and the size in two-dimensions is (0.26, 0.28). So, the sensitive volume is defined by the center coordinates plus and minus one half of the x-size in the positive and negative x directions and plus and minus one half of the y-size in the positive and negative y directions. The sensitive volume definitions (center and size) for the INV cell are provided in Table 1. This same process can be applied to all cells in a library. The MRED Python script for the INV for all these conditions is provided in the APPENDIX A.

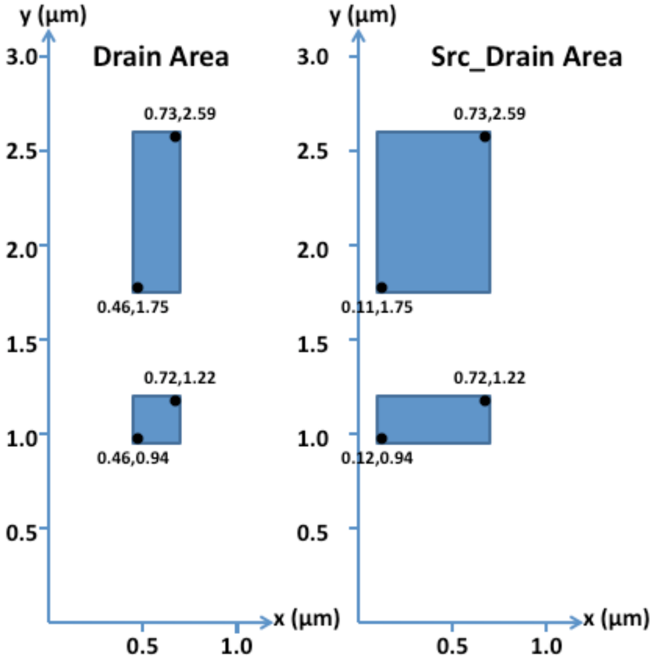


Figure 20. Top view of drain area and active area for basic INV cell (PMOSFET on top and NMOSFET on bottom)

Table 1. Sensitive volume definitions for MRED Python script including the (x, y, z) coordinates in mm for the volume center and the length, width, and depth for the volume, also in mm

Sensitive Volume Name	Sensitive Volume Center (μm)	Sensitive Volume Size (μm)
PMOSFET Drain	(0.59, 2.17, 0.175)	(0.26, 0.84, 0.35)
PMOSFET Src_Drain	(0.42, 2.17, 0.175)	(0.62, 0.84, 0.35)
PMOSFET Well-1	(0.59, 2.17, 0.50)	(0.52, 1.14, 0.30)
PMOSFET Well-2	(0.42, 2.17, 0.55)	(0.88, 1.24, 0.40)
NMOSFET Drain	(0.59, 1.08, 0.175)	(0.26, 0.28, 0.35)
NMOSFET Src_Drain	(0.42, 1.08, 0.175)	(0.60, 0.28, 0.35)
NMOSFET Well-1	(0.59, 1.08, 0.50)	(0.52, 0.48, 0.30)
NMOSFET Well-2	(0.42, 1.08, 0.55)	(0.86, 0.58, 0.40)

Converting Deposited Energy to Collected Charge

To convert the charge deposited in the nested sensitive volumes to collected charge, each deposited charge is multiplied by the corresponding efficiency. A best first guess for the efficiencies were made based on TCAD simulations [11], as listed in Table 2. This table lists the multiplier used in the Python script to determine the collected charge in each volume. The table also lists a cumulative efficiency, because the volumes are nested. For example, the drain volume is completely contained within the src_drain volume, so any charge deposited in the drain will be found in both volumes. Therefore, the cumulative charge collection efficiency of the drain is the sum of the drain and src_drain multipliers.

An iterative process was used to refine the values of the efficiencies by comparing successive MRED coupled with SPICE predictions to experimental data on a set of defined circuits (more details on calibration of the multi-scale simulation approach to experimental data appears in Chapter VI, the interested reader can go directly to that chapter). Table 2 gives the final efficiencies that provided the best match (i.e., least squares) of the multi-scale simulation approach to the SET experimental data. It should be noted that the only item that changed from the start were the well-1 sensitive volumes. They both increased, but the largest difference was the final collection efficiency in the PMOSFET well-1 volume.

Table 2. Charge collection efficiencies listed for each sensitive volume in the INV cell

Sensitive Volume Name	Beginning Efficiencies		Final Efficiencies	
	Multiplier	Cumulative	Multiplier	Cumulative
PMOSFET Drain	0.60	0.80	0.60	0.80
PMOSFET Src_Drain	0.20	0.20	0.20	0.20
PMOSFET Well-1	0.15	0.25	0.70	0.80
PMOSFET Well-2	0.10	0.10	0.10	0.10
NMOSFET Drain	0.75	1.00	0.75	1.00
NMOSFET Src_Drain	0.25	0.25	0.25	0.25
NMOSFET Well-1	0.44	0.54	0.55	0.65
NMOSFET Well-2	0.10	0.10	0.10	0.10

MRED Python scripts can be tailored to provide a wide range of outputs. A sample of an output file for the INV cell simulated with particles for an LET equal to 2.2 MeV-cm^2/mg is provided in Table 3. This table shows: (1) the MRED event number, (2) the “weight” or value multiplied to the number of events for each particle, (3) the charge collected in each of the eight volumes, (4) the total charge collected in the PMOSFET, and (5) the total charge collected in the NMOSFET. The actual script ran for 100,000 events.

Table 3. Sample outputs from the MRED Python script with charge collection calculations (charge given in fC)

Event #	Weight	PMOSFET Volumes				NMOSFET Volumes				PMOSFET	NMOSFET
		Drain	Src-Drain	Well-1	Well-2	Drain	Src-Drain	Well-1	Well-2	Charge	Charge
11	1.00E+00	0.0000	0.0000	0.0000	0.0113	0.1543	0.1580	0.1392	7.6224	0.0011	0.9940
50	1.00E+00	0.0478	0.0795	6.3753	8.1480	0.0000	0.0000	0.0000	0.0000	5.3221	0.0000
70	1.00E+00	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	7.1190	0.0000	0.7119
78	1.00E+00	0.0000	0.0000	0.0170	0.3299	0.1195	0.1747	0.2118	7.4301	0.0449	0.9928
100	1.00E+00	0.0000	0.1580	6.5487	9.5263	0.0000	0.0000	0.0000	0.0154	5.5683	0.0015
124	1.00E+00	0.0000	0.0000	0.0000	0.0000	0.0535	0.1129	0.0334	1.3418	0.0000	0.2209
137	1.00E+00	0.0000	6.9549	6.0394	7.8156	0.0000	0.0000	0.0000	0.0000	6.4001	0.0000
164	1.00E+00	0.0000	0.0000	0.0000	0.0000	0.3183	0.5146	6.4071	7.6815	0.0000	4.6594
181	1.00E+00	6.2057	6.8611	5.9113	9.6009	0.0000	0.0000	0.0000	0.0000	10.1936	0.0000
216	1.00E+00	0.0000	0.0000	0.0000	0.0000	0.0000	0.0316	0.4386	7.8000	0.0000	1.0291
265	1.00E+00	0.1639	0.4690	6.3636	8.2951	0.0000	0.0000	0.0000	0.0594	5.4762	0.0059
275	1.00E+00	0.5416	0.0748	0.1334	9.3955	0.0000	0.0000	0.0000	0.0000	1.3729	0.0000
292	1.00E+00	0.0000	6.9953	7.0074	8.3963	0.0000	0.1370	0.1059	0.0000	7.1439	0.0925
395	1.00E+00	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0297	7.2068	0.0000	0.7370
399	1.00E+00	0.0000	0.0000	0.0489	0.0071	6.2529	6.2670	5.5741	9.8968	0.0349	10.3119
401	1.00E+00	0.0000	0.0000	0.0000	0.0000	0.0000	0.2383	0.2479	1.0124	0.0000	0.2972
506	1.00E+00	0.0000	0.0000	0.0000	0.0000	0.0066	0.0066	0.0000	8.4673	0.0000	0.8533
637	1.00E+00	0.0712	5.7400	5.6690	9.6301	0.0000	0.0000	0.0000	0.4330	6.1220	0.0433
685	1.00E+00	0.0042	0.0000	0.0358	0.0616	0.0204	0.7004	5.6042	9.6170	0.0337	4.2344
784	1.00E+00	0.1746	0.1600	0.4777	8.2528	0.0000	0.0000	0.0000	0.0000	1.2964	0.0000
875	1.00E+00	0.1575	6.7334	5.7045	6.5224	0.0000	0.0000	0.0000	0.0000	6.0866	0.0000
895	1.00E+00	0.0000	0.0000	0.0000	10.0163	0.0000	0.0000	0.0000	0.2284	1.0016	0.0228
905	1.00E+00	7.9053	0.4718	7.1202	7.1501	0.0000	0.0000	0.0000	0.0000	10.5367	0.0000
928	1.00E+00	0.0000	0.0000	0.0303	8.6935	0.1373	0.2104	0.2060	0.0336	0.8906	0.2722
989	1.00E+00	1.0810	0.0000	0.3374	7.7787	0.0000	0.0000	0.0000	0.0000	1.6627	0.0000

CHAPTER V

MODELING OF SINGLE EVENT TRANSIENTS WITH DUAL DOUBLE-EXPONENTIAL CURRENT SOURCES

This chapter describes a simple, yet effective method to model the current waveform resulting from a charge collection event for digital single event transient (SET) circuit simulations. The model uses two double-exponential current sources in parallel, and the results illustrate why a conventional model based on one double-exponential source is insufficient to model long single event transients. A small set of logic cells with varying input conditions, drive strength, and output loading are simulated to extract the parameters for the dual double-exponential current sources. The parameters are based upon both the node capacitance and the restoring current (i.e., drive strength) of the logic cell.

Background

If a radiation event traverses close enough to the depletion region of a sensitive junction, then the non-equilibrium charge distribution can induce a temporary modulation of the potential along the trajectory of the event. A period of prompt collection typically follows as the potential collapses to the normal state. Subsequently, motion of carriers by diffusion to the p-n junction dominates the collection process until all the excess carriers are collected, recombine, or diffuse away from the junction area. The charge collected from the radiation event produces a current pulse at the node. The time constants depend strongly on the type of ion, its energy, and the properties of the specific technology.

After an ion passes through a sensitive volume in a combinational library cell a voltage SET appears at the cell's output – modeled as a double-exponential pulse Figure 21. This voltage waveform is the fundamental component for the MRED to SPICE integration. After the transient propagates through a few library cells, the response of the circuit shapes the SET into a square-wave (Figure 25).

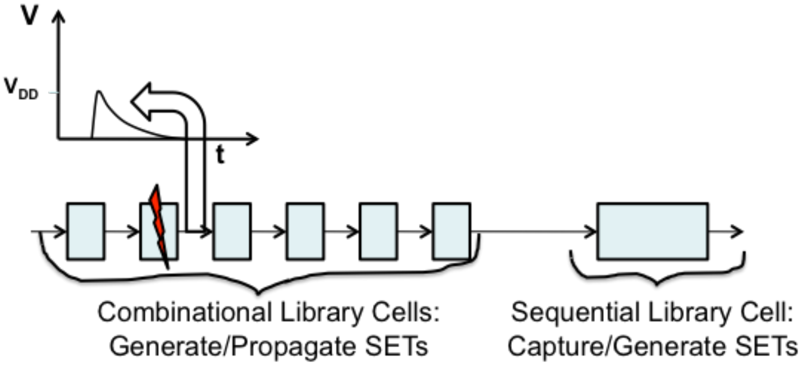


Figure 21. Ion strike on combinational library cell modeled as double exponential current source

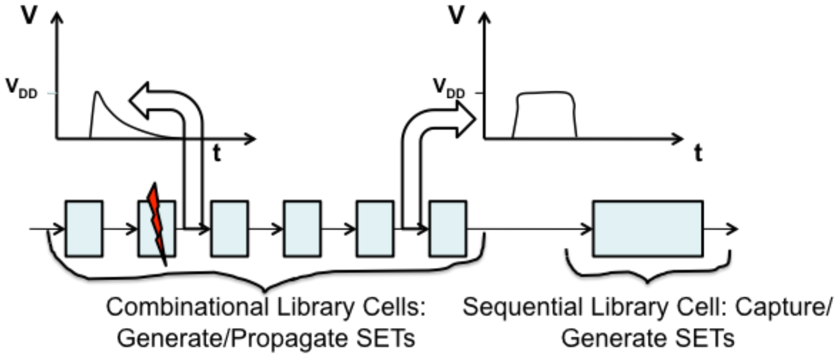

V
V_{DD}
t
V
V_{DD}
t
Combinational Library Cells:
Generate/Propagate SETs
Sequential Library Cell: Capture/
Generate SETs

Figure 22. Propagation of double-exponential current source to square wave

In advanced ICs, the circuit response time can be comparable to the characteristic time for single-event charge collection event; the charge collection process dynamically interacts with the cell's circuit response to the event [41, 42]. Correctly modeling the transient shape of the pulse is critical to providing accurate circuit soft error predictions.

A commonly used analytical model to approximate the induced transient current waveform is the double-exponential function with a rapid rise time and gradual fall time (Figure 23) [6, 23, 43]. This waveform is the most common form used in transistor-level simulations. The equation for this current pulse, *I(t)*, in SPICE is:

$$I(t)=\begin{cases} 0 & ;t<t_{d1} \\ I_{Peak}\left(1-e^{\frac{-(t-t_{d1})}{\tau_1}}\right) & ;t_{d1}<t<t_{d2} \\ I_{Peak}\left(e^{\frac{-(t-t_{d2})}{\tau_2}}-e^{\frac{-(t-t_{d1})}{\tau_1}}\right) & ;t>t_{d2} \end{cases} \tag{3}$$

where, t_{d1} is the onset of the rise of the current, t_{d2} is the onset of the fall of the current, I_{Peak} is the maximum current to be approached, τ_1 is the rise time constant, and τ_2 is the fall time constant. The total charge delivered by the current pulse, Q_{Total}, is the integral over time of *I(t)*:

$$Q_{Total}=I_{Peak}\left(\tau_1+\tau_2+\left(t_{d2}-t_{d1}\right)-\tau_1 e^{\frac{-(t_{d2}-t_{d1})}{\tau_1}}\right) \tag{4}$$

If τ_1 is small compared to the difference in time between the rising and falling edges of the double-exponential waveform (t_{d2} - t_{d1}), then the last term is insignificant, and the calculation of total charge is simple.

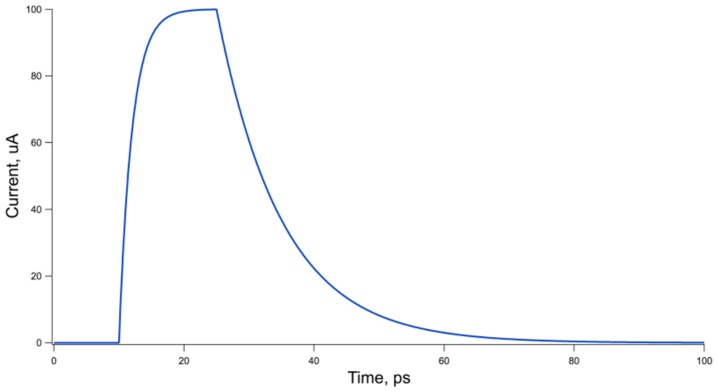


Figure 23. Example of a Double-Exponential Current Pulse (I_{Peak} = 100 μA, t_{d1} = 10 ps, t_{d2} = 5 ps, τ_1 = 2 ps, τ_2 = 10 ps)

Limitations of Double-Exponential Current Source

Previous research has shown that SET pulse-widths may be upwards of hundreds of picoseconds under some conditions [40]. When these long pulses are modeled with one double-exponential current source, the resulting voltage transient either overdrives the circuit significantly or has a very slow leading edge, depending on the selection of parameters. If (t_{d2} – t_{d1}) is increased or if I_{Peak} is increased, then the current pulse can overdrive the circuit, forward-biasing the source-body junction(s). This property is shown in Figure 24, where the transient voltage drops below V_{SS} = 0 volts. This overdrive will result in the simulation over-predicting the amount of charge needed to produce longer SET pulse-widths. On the other hand, if (t_{d2} – t_{d1}) is increased and if I_{Peak} is decreased to compensate for the overdrive, then the resulting voltage transient will have a slow leading edge (Figure 25). Neither of these results describes SET pulses accurately [41, 42].

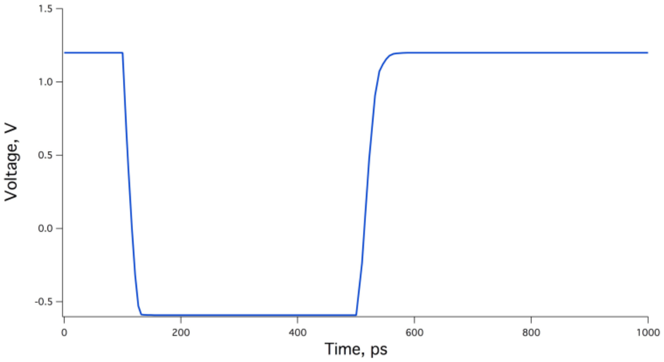


Figure 24. Example of a voltage transient that overdrives the circuit (i.e., the voltage drops below V_{SS} = 0 volts)

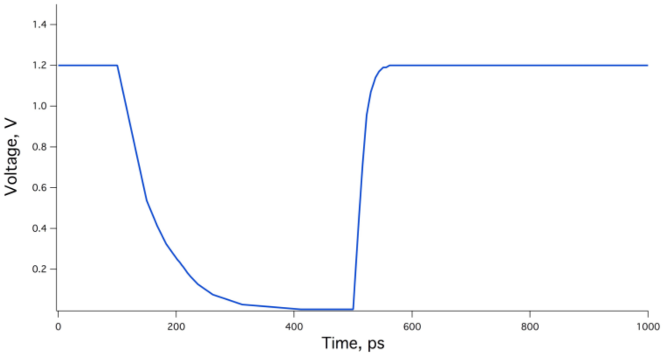


Figure 25. Example of a voltage transient with a slow leading edge

Other researchers have proposed single event models that make use of the node voltage to control the single event current source to overcome these limitations, but the implementation in a transistor simulation is no longer simple [41]. This chapter introduces an extension of the double-exponential current source: the dual double-exponential current source. The dual double-exponential current source model is composed of two parallel double-exponential current sources, one for prompt charge collection and one for sustained charge collection. This model can be used to perform SET simulations.

Dual Double-Exponential Current Source Model

The dual double-exponential current source is based upon single event device-level simulations, as shown in Figure 26 [41, 42]. There is a short high current peak, followed by a sustained shelf of lower current. This behavior can be described by a long double-exponential current source with I_{Peak} equal to the shelf current and a short double-exponential current source to add the extra current for the short peak. Figure 27 shows an example of the two individual current sources and the result of their parallel combination for the dual double-exponential current source model.

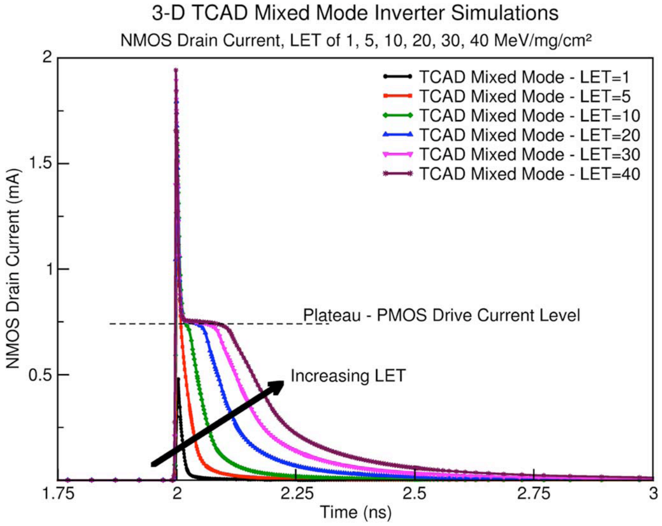


Figure 26. Device-level simulation results showing short burst of high current followed by a sustained shelf of lower current (after [41])

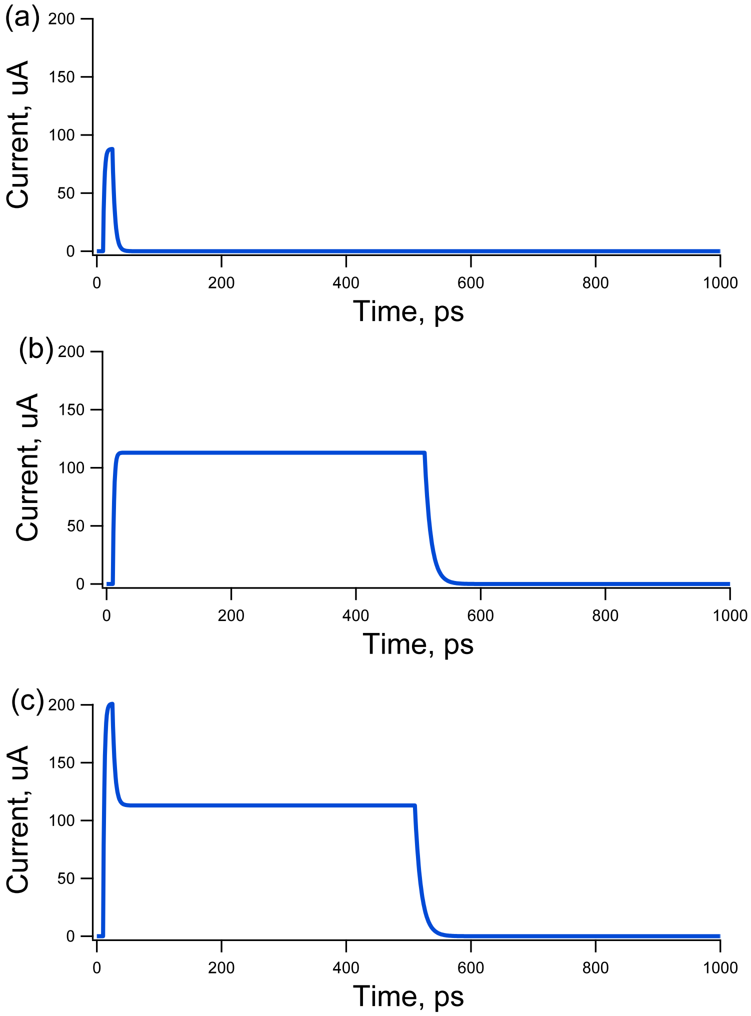


Figure 27. Example of: (a) short peak, $I_{Prompt}(t)$, (b) sustained, $I_{Hold}(t)$, and (c) dual double-exponential current sources

Both current sources have four parameters that need to be determined: I_{Peak}, ($t_{d2} – t_{d1}$), τ_1, and τ_2. For the short duration current source, $I_{Prompt}(t)$, the three time parameters are set from device-level single event simulations of a single transistor. Based on results obtained for a 90-nm technology, these are ($t_{d2} – t_{d1}$) = 15 ps, τ_1 = 2 ps, and τ_2 = 4 ps [44]. For the longer duration current source, $I_{Hold}(t)$, τ_1 = 2 ps and τ_2 = 10 ps provide a good fit, and ($t_{d2} – t_{d1}$) is used as a variable that depends on the amount of deposited charge.

The peak values for I_{Prompt} and I_{Hold} are determined through transistor-level simulations. The first set of simulations determines the peak current in a short-duration, double-exponential current source that causes the voltage output to change from one voltage rail to the other. For a basic inverter with its input held low, we simulated one double-exponential current source with (t_{d2} – t_{d1}) = 15 ps, τ_1 = 2 ps, and τ_2 = 10 ps and determined what I_{Peak} value drives the loaded inverter's voltage output to switch from V_{DD} to V_{SS}. This peak value is defined as I_{Thresh}, which equals the sum of I_{Prompt} and I_{Hold}. The second set of simulations determines I_{Hold} by applying the dual double-exponential current sources. We define the $I_{Prompt}(t)$ current source with the timing parameters from the previous paragraph and I_{Prompt} = I_{Thresh} – I_{Hold}. We define the $I_{Hold}(t)$ current source with τ_1 = 2 ps, τ_2 = 10 ps, and (t_{d2} – t_{d1}) = 500 ps. We identify the I_{Hold} that will result in a transient voltage near the opposite rail at the end of the 500 ps. The transient voltage does not remain near the opposite rail for the entire duration of the hold current.

Once the I_{Prompt} and I_{Hold} values have been extracted for the circuit, the total charge for the injected current is obtained as the sum of the charge from $I_{Prompt}(t)$ and the charge from $I_{Hold}(t)$, which are calculated from equation (4). Determining the current sources from a given charge is a little more complicated, but still straightforward. For long SETs, $I_{Prompt}(t)$ does not depend on the total charge, so the charge from $I_{Prompt}(t)$ is independent of the details of the pulse plateau. The charge associated with $I_{Prompt}(t)$, namely Q_{Prompt}, is subtracted from the total charge (Q_{Total}) to obtain Q_{Hold}, the charge provided by $I_{Hold}(t)$:

$$Q_{Hold} = Q_{Total} - Q_{Prompt} \tag{5}$$

The second step is to apply equation (4) to Q_{Hold}, using ($t_{d2} - t_{d1}$) as a variable that depends on the total charge. As an example, consider I_{Prompt} = 97 μA, I_{Hold} = 114 μA, Q_{Total} = 25 fC, and the timing parameters given in this section. The calculation for Q_{Prompt} using equation (4) without the last term is:

$$Q_{Prompt} = \left(97.0\ x\ 10^{-6}\right)\left(2.0\ x\ 10^{-12} + 4.0\ x\ 10^{-12} + 15.0\ x\ 10^{-12}\right)C = 2.04 fC \tag{6}$$

This means that Q_{Hold} is 2.04 fC less than 25 fC, or 22.96 fC. Applying equation (4) again for Q_{Hold} gives:

$$Q_{Hold} = \left(114.0\ x\ 10^{-6}\right)\left(2.0\ x\ 10^{-12} + 4.0\ x\ 10^{-12} + \left(t_{d2} - t_{d1}\right)\right)C = 22.96 fC \tag{7}$$

Solving equation (7) for ($t_{d2} - t_{d1}$) results in 151 ps.

SET Pulse Shape in Combinational Cells

A SPICE deck was created implementing both a baseline 4-inverter chain and the dual double-exponential current source model. A schematic of the baseline 4-inverter chain is shown below (Figure 28) and its SPICE netlist is found in APPENDIX B:

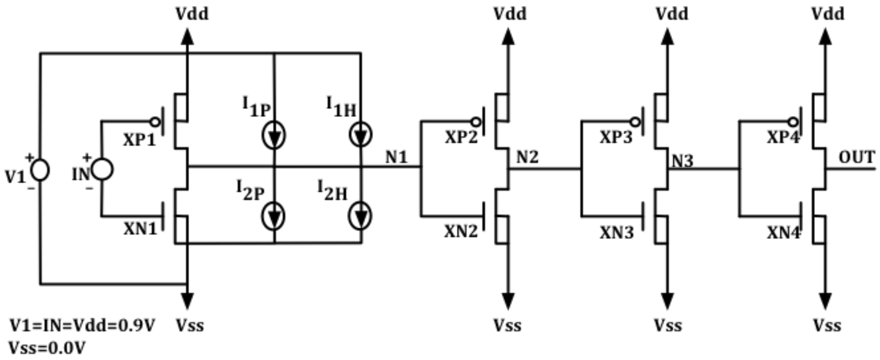


Figure 28. Baseline 4-inverter chain schematic

Automated scripting is used to determine the important parameters for the simulations. This process incorporates the implementation of the dual double-exponential current source model for SET pulses for both the NMOSFETs and PMOSFETs in the 90-nm inverter design. The flowchart that searches for I_{thresh} is seen in Figure 29. A flow chart for I_{Prompt} and I_{Hold} is seen in Figure 30, and the Python script can be found in APPENDIX C.

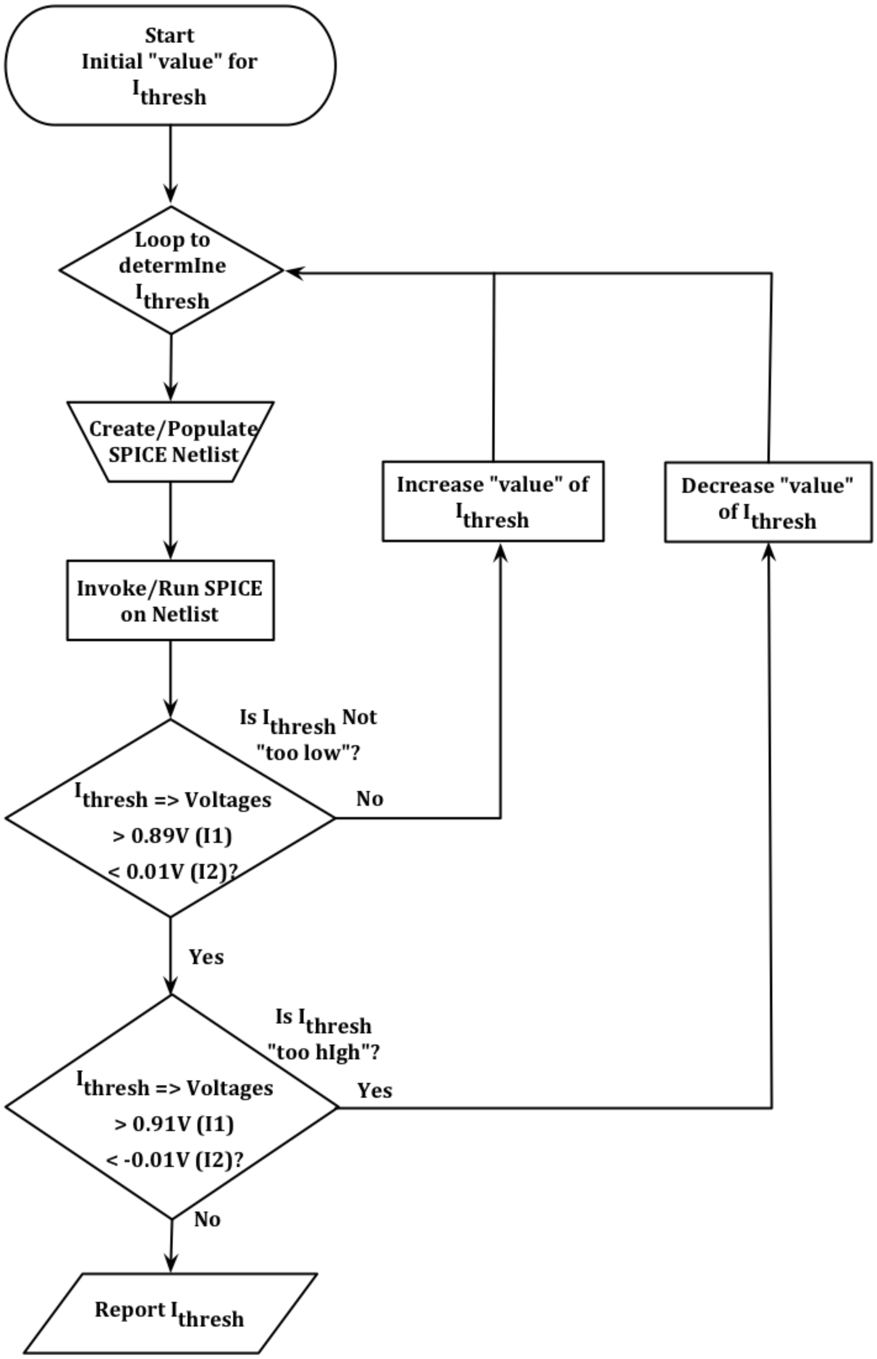


Figure 29. Flowchart to Identify *I_{thresh}* variable for implementation with the dual double-exponential current source model

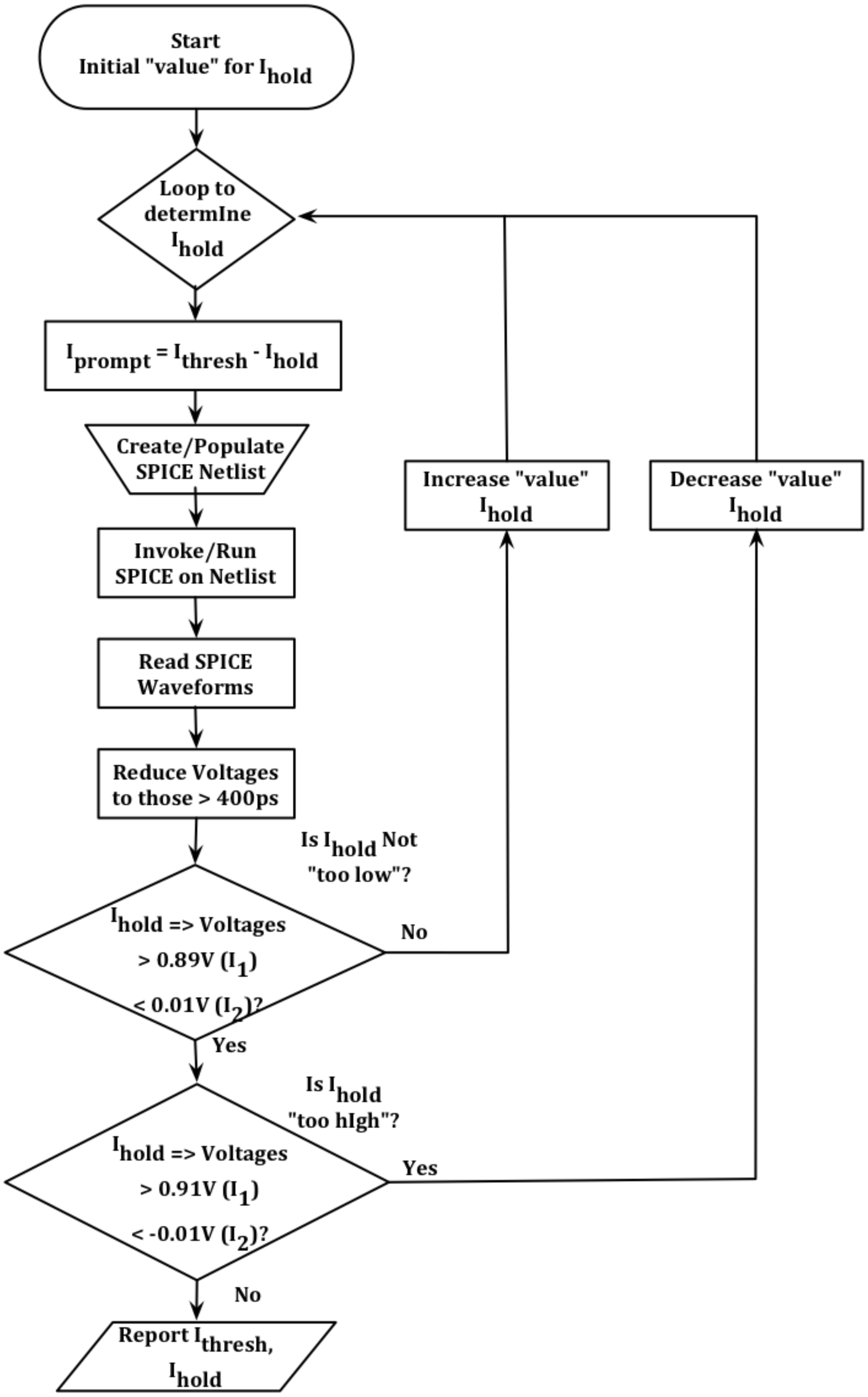

Start
Initial "value" for I_{hold}
Loop to determIne I_{hold}
$I_{prompt} = I_{thresh} - I_{hold}$
Create/Populate SPICE Netlist
Invoke/Run SPICE on Netlist
Read SPICE Waveforms
Reduce Voltages to those > 400ps
Is I_{hold} Not "too low"?
I_{hold} => Voltages > 0.89V (I_1) < 0.01V (I_2)?
No
Yes
Increase "value" I_{hold}
Is I_{hold} "too hIgh"?
I_{hold} => Voltages > 0.91V (I_1) < -0.01V (I_2)?
Yes
No
Decrease "value" I_{hold}
Report I_{thresh}, I_{hold}

Figure 30. Flowchart to identify I_{Prompt} and I_{Hold} variables for implementation into the dual double-exponential current source model

Results and Discussion

We constructed several library cells for the IBM 90-nm technology and simulated them to determine I_{Prompt} and I_{Hold} for different input conditions and loads. The results are given in Tables 1 through 4. The first column of each table gives the cell name. For the inverter (INV1) cell, the W/L for the PMOSFET was 480 nm / 100 nm, and the NMOSFET was 200 nm / 100 nm. The INV2 cell used transistors with double-width transistors and the INV4 cell used quadruple-width transistors. The two-input NAND (NAND2) cell used the same width for the PMOSFETs and double the width for the NMOSFETs as the INV1 cell. Likewise, the two-input NOR (NOR2) cell used double the width for the PMOSFETs and the same width for the NMOSFETs as the INV1 cell. The second column gives the input condition. For IN or IN1 = V_{DD}, the current sources were injected on the PMOSFET drain, and for IN or IN1 = V_{SS}, the current sources were injected on the NMOSFET drain. The third column lists the load simulated with the number in parentheses giving the number of loads. The NAND2 and NOR2 cells were loaded by connecting the output to Input1 of the next cell. Input2 tied to appropriate rail voltage. The fourth column provides the estimated node capacitance in fC. This value was calculated from model parameters and used output drain and load gate capacitance. The fifth and sixth columns provide the determined I_{Prompt} and I_{Hold} levels in μA. The sixth column provides the calculated Q_{Prompt} for the $I_{Prompt}(t)$ current source. Finally, the seventh column gives an estimate for the total charge, Q_{Total}, to provide an SET with a 200 ps pulse-width.

Example injected currents and resulting SET voltage waveforms are provided in Figure 31 and Figure 32, respectively.

Table 4. Simulation Results for INV1, NAND2, NOR2 cells for the V_{DD} input configuration

Cell Name	Input Configuration	Load	~C-Node, fC,	I_{Prompt}, μA	I_{Hold}, μA	Q_{Prompt}, fC	~Q_{Total}, fC 200 ps
INV1	IN = V_{DD}	INV1 (1)	1.38	97	141	2.0	30.2
INV1	IN = V_{DD}	INV1 (2)	2.20	169	141	3.5	31.7
INV1	IN = V_{DD}	INV1 (4)	3.83	270	140	5.7	33.7
NAND2	IN1 = V_{DD}, IN2 = V_{DD}	NAND2 (1)	1.74	157	153	3.3	33.9
NOR2	IN1 = V_{DD}, IN2 = V_{SS}	NOR2 (1)	2.23	169	141	3.5	31.7

Table 5. Simulation Results for INV1, NAND2, NOR2 cells for the V_{SS} input configuration

Cell Name	Input Configuration	Load	~C-Node, fC,	I_{Prompt}, μA	I_{Hold}, μA	Q_{Prompt}, fC	~Q_{Total}, fC 200 ps
INV1	IN = V_{SS}	INV1 (1)	1.38	88	113	1.8	24.4
INV1	IN = V_{SS}	INV1 (2)	2.20	173	113	3.6	26.2
INV1	IN = V_{SS}	INV1 (4)	3.83	271	113	5.7	28.3
NAND2	IN1 = V_{SS}, IN2 = V_{DD}	NAND2 (1)	1.74	131	113	2.8	25.4
NOR2	IN1 = V_{SS}, IN2 = V_{SS}	NOR2 (1)	2.23	180	110	3.8	25.8

Table 6. Simulation Results for inverter cells of increasing drive strength for the V_{DD} input configuration

Cell Name	Input Configuration	Load	~C-Node, fC,	I_{Prompt}, μA	I_{Hold}, μA	Q_{Prompt}, fC	~Q_{Total}, fC 200 ps
INV1	IN = V_{DD}	INV1 (1)	1.38	97	141	2.0	30.2
INV2	IN = V_{DD}	INV2 (1)	2.59	196	264	4.1	56.9
INV4	IN = V_{DD}	INV4 (1)	4.99	388	512	8.1	110.5

Table 7. Simulation Results for inverter cells of increasing drive strength for the V_{SS} input configuration

Cell Name	Input Configuration	Load	~C-Node, fC,	I_{Prompt}, μA	I_{Hold}, μA	Q_{Prompt}, fC	~Q_{Total}, fC 200 ps
INV1	IN = V_{SS}	INV1 (1)	1.38	88	113	1.8	24.4
INV2	IN = V_{SS}	INV2 (1)	2.59	175	225	3.7	48.7
INV4	IN = V_{SS}	INV4 (1)	4.99	356	454	7.5	98.3

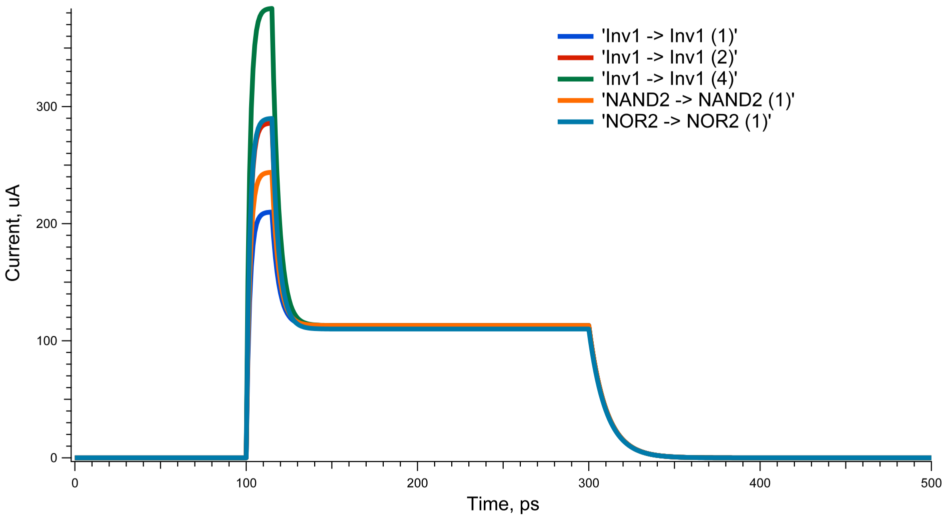


Figure 31. Injected current waveforms for circuit configurations with loads listed in Table 5 to produce ~200 ps SET

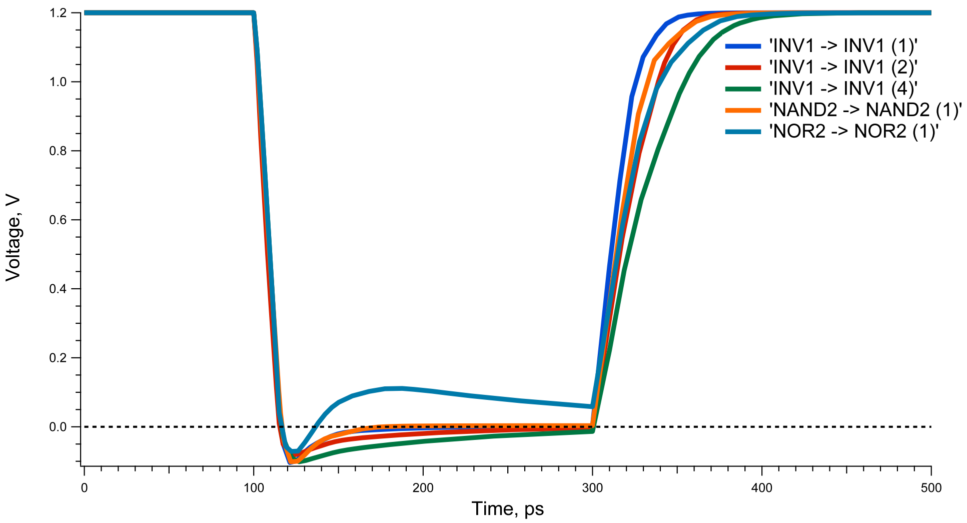


Figure 32. Resulting SET voltage waveforms for the circuit configurations with loads listed in Table 5 and for the injected current waveforms shown in Figure 31.

I_{Hold} is a strong function of the restoring current in the circuit (i.e., drive strength). The INV1, NAND2, and NOR2 cells have similar restoring currents, and I_{Hold} is nearly constant in these cells, as seen in Table 4 and Table 5. INV2 has twice the restoring current and INV4 has four times the restoring current. I_{Hold} in these cells generally scales with the increase in restoring current (Table 6 and Table 7). I_{Hold} does not depend upon the load, demonstrated by the first three rows in each of Table 4 and Table 5. On the other hand, I_{Prompt} is a strong function of the node capacitance, as the ratio between I_{Hold} and the node capacitance remains fairly constant throughout all tables. For all base cells (INV1, NAND2, NOR2), the charge that results in a 200 ps SET varies between 24.4 and 28.3 fC for NMOSFET simulations and 30.2 and 33.9 fC for PMOSFET simulations. As a result, SET pulse-widths show little variation for different loads and different cell types for logic cells with similar drive strengths.

Figure 32 shows the resulting voltage waveforms from the simple model proposed in the paper, and Figure 33 shows similar results from device-level modeling. A comparison illustrates the potential inaccuracies of this simple model. The inaccuracies arise from driving the voltage to the opposite voltage rail at the end of a long SET, where the device-level simulations show a slow drift away from the opposite voltage rail. All but one of the voltage transients shown in Figure 32 will drive the output voltage slightly below V_{SS} following the initiation of the single event current sources. These simulation results will produce a slight over prediction of the amount of charge needed to produce that SET pulse. The other voltage transient, NOR2 –> NOR2 (1), shows the output voltage going back above V_{SS} and staying above until the end of the transient. This simulation result will produce a slight under-prediction of the amount of charge needed to produce that SET pulse. However, the dual double-exponential current source model will still be more accurate than the one double-exponential current source model. Typically, a simulation with the one double-exponential current source would extend the SET pulse-width by holding the peak current level for a longer time. Since much less current is actually needed to sustain the transient, the one double-exponential current source model can result in a significant over-prediction of the amount of charge required to produce a particular SET, or an under-estimation of the resulting SET pulse-width from a given amount of charge.

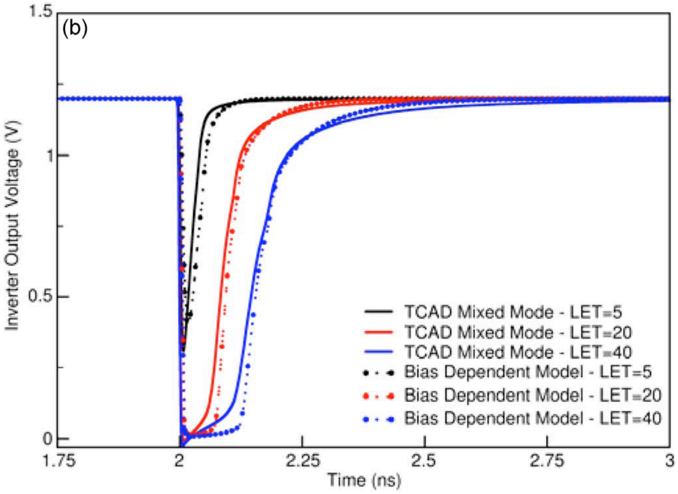


Figure 33. Device-level simulation results showing voltage transients (solid lines) for various deposited charges (after [41])

Timing analysis with pulse broadening

In [45-47], the authors investigate SET propagation with a focus on long SET pulses through large inverter chains that contained several thousand inverters. They reported little to no pulse-broadening in bulk devices, but, SOI designs demonstrated a significant pulse-broadening per inverter [46,47].

Massengill et al., used a simple level-one generic model to investigate the theory of pulse-broadening [47]. They identified two critical conditions that need to be considered for the propagation of SETs that go from rail-to-rail (called *strong SETs*). The conditions for strong SET propagation are: (1) the slowest of the rise or fall time constant of the originating SET voltage transient is faster than the characteristic rise/fall time of the combinational cell, and (2) the pulse-width of the originating SET voltage pulse, measured at the input voltage is greater than the rise time, τ_r plus the fall time, τ_f. A fast rise and slow fall time results in pulse-broadening, while a slow rise and fast fall time results in attenuation.

The characteristic rise and fall times, τ_r and τ_f, of the combinational cells are determined by using a ring oscillator design. This design is a string of 83-inverters that are daisy-chained to one another. This is shown as 8-INV10s, comprising a string of 10-inverters followed by 3-INV1s (Figure 34).

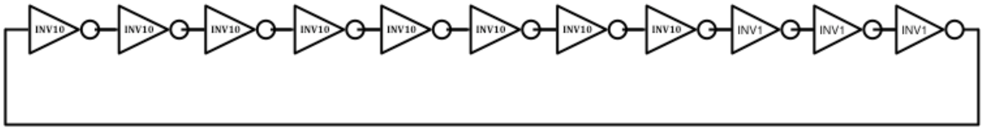


Figure 34. 90-nm inverter ring oscillator

The rise and fall times for the 90-nm inverter design are τ_r = 23.5 ps and τ_f = 20.7 ps for approximately a 45-ps SET pulse-width minimum that should propagate through the 90-nm inverter. Ring oscillator designs are implemented for the 90-nm NAND gate and NOR gate to determine the rise and fall times of these specific combinational cells. They result in an approximate minimum pulse width of 76 ps (i.e., τ_r = 33.1 ps and τ_f = 42.8 ps) that should propagate through the 90-nm NAND gate, and an approximate minimum pulse width of 105 ps (i.e., τ_r = 67.1 ps and τ_f = 38.1 ps) that should propagate through the 90-nm NOR gate. Using these minimum pulse widths reduces the simulation space.

CHAPTER VI

MRED2SPICE ANALYSIS

This chapter details the integration of the tools for radiation transport (MRED) and circuit-level simulation (SPICE). This flow is called MRED2SPICE. The details are used for the development of the Python scripting for integration and automation of MRED2SPICE for SET generation and propagation. Descriptions of the different processes are provided. Results from the MRED2SPICE portion of the multi-scale simulation approach are compared to experimental data for three cells.

Connecting MRED to SPICE for SET analysis

In [10, 48, 49], the authors suggest that dynamic errors can be the dominant contributor to the overall system soft error response. In order to predict dynamic errors, a calibrated model that generates and propagates SETs from particles in a radiation environment to circuit response is required. Multiple organizations have developed circuits to characterize SETs. Narasimham et al. [40, 50], Baze et al. [50], and Cannon et al. [16] use an on-chip asynchronous approach to measure SET pulse widths. We use data from [11] to compare to simulation results from MRED2SPICE segment of the multi-scale simulation approach.

MRED2SPICE Framework

Figure 35 shows the building blocks of the portion of the multi-scale simulation that uses MRED and SPICE. The amount of charge associated with each radiation event generated by MRED resulting in a sufficient charge collection to produce an SET is passed to the SPICE process for generation and propagation of an SET. Applying the charge collection efficiencies and converting the charge to independent current sources is accomplished in the SPICE Python script. SPICE simulations categorize the effect of the SET on the circuit. The results are compared to the experimental SET data provided in Cannon et al. [16]. This paper uses the same 90-nm technology for the library cells, and the comparisons are made for a lightly ionizing environment.

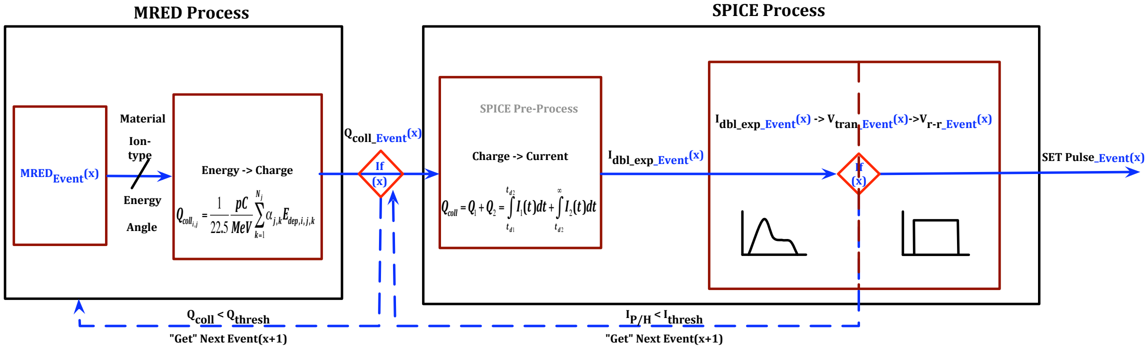


Figure 35. MRED to SPICE (MRED2SPICE) framework block diagram.

MRED2SPICE Python Script Implementation

An MRED output data file that consists of a list of the characteristics of each radiation event, including the struck node and the charge collected for each nested sensitive volume, is precomputed for the specific circuit. Each event is then analyzed to determine if Q_{coll} is greater than Q_{thresh}. If this is true, then charge is converted to current for input into SPICE; this current pulse is uniquely identified with the specific energy deposition computed by MRED.

The current source for SPICE is computed by determining the values for I_{Prompt} and I_{Hold} from Q_{coll}, and the corresponding pulse-width (t_{d2}) resulting from the excess charge, which is greater than the Q_{prompt} charge. The initial values for the SPICE dual double-exponential current sources (I_{Prompt}, I_{Hold}) were taken from [42]. The script creates a circuit netlist and populates the dual current sources with these values. These circuits were used to evaluate this portion of the multi-scale simulation: (1) a target chain of 65 inverters (INV1s), or NAND gates (NAND2s), or NOR gates (NOR2s), (2) a guard gate, and (3) an asynchronous latch. This mimics the designs by Cannon et al. [16], and the inverter version is seen in Figure 36:

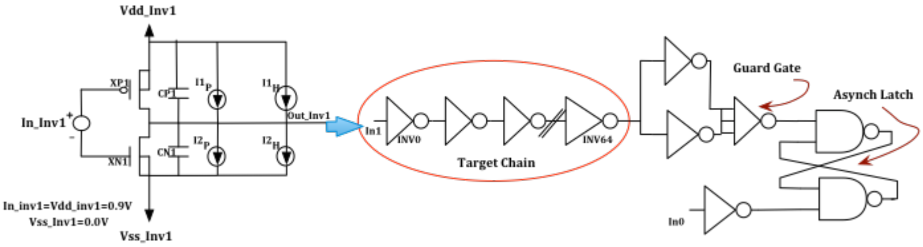


Figure 36. Sample test design using INV1s for MRED2SPICE development

SPICE is then run on the netlist producing two output data files. The first file records if an SET is generated from a particular MRED event (identified with an MRED event number) and the corresponding SET pulse-width. The second file records if the SET produces an upset at the output of the asynchronous latch. The output files are in data array fashion, and the first column of the array is the MRED event number that is parsed in the first step. This script is done in parallel for both the PMOSFETs and NMOSFETs designed in the combinational cell. Figure 37 shows a flowchart for the MRED2SPICE Python script, and an example of the Python script is in APPENDIX D. Examples of the MRED source data file and the SPICE SET latch results data file are seen side-by-side in Figure 38.

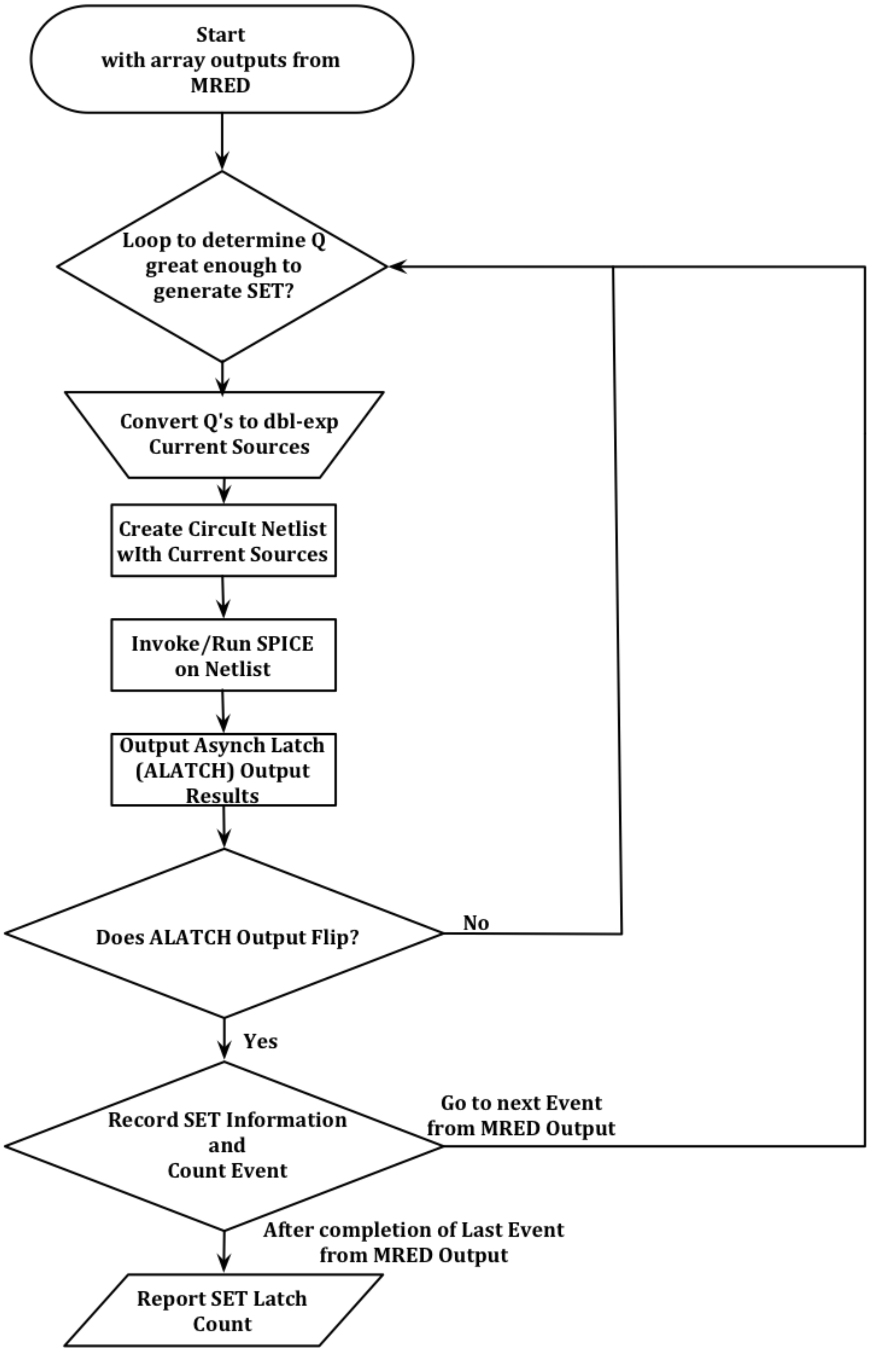
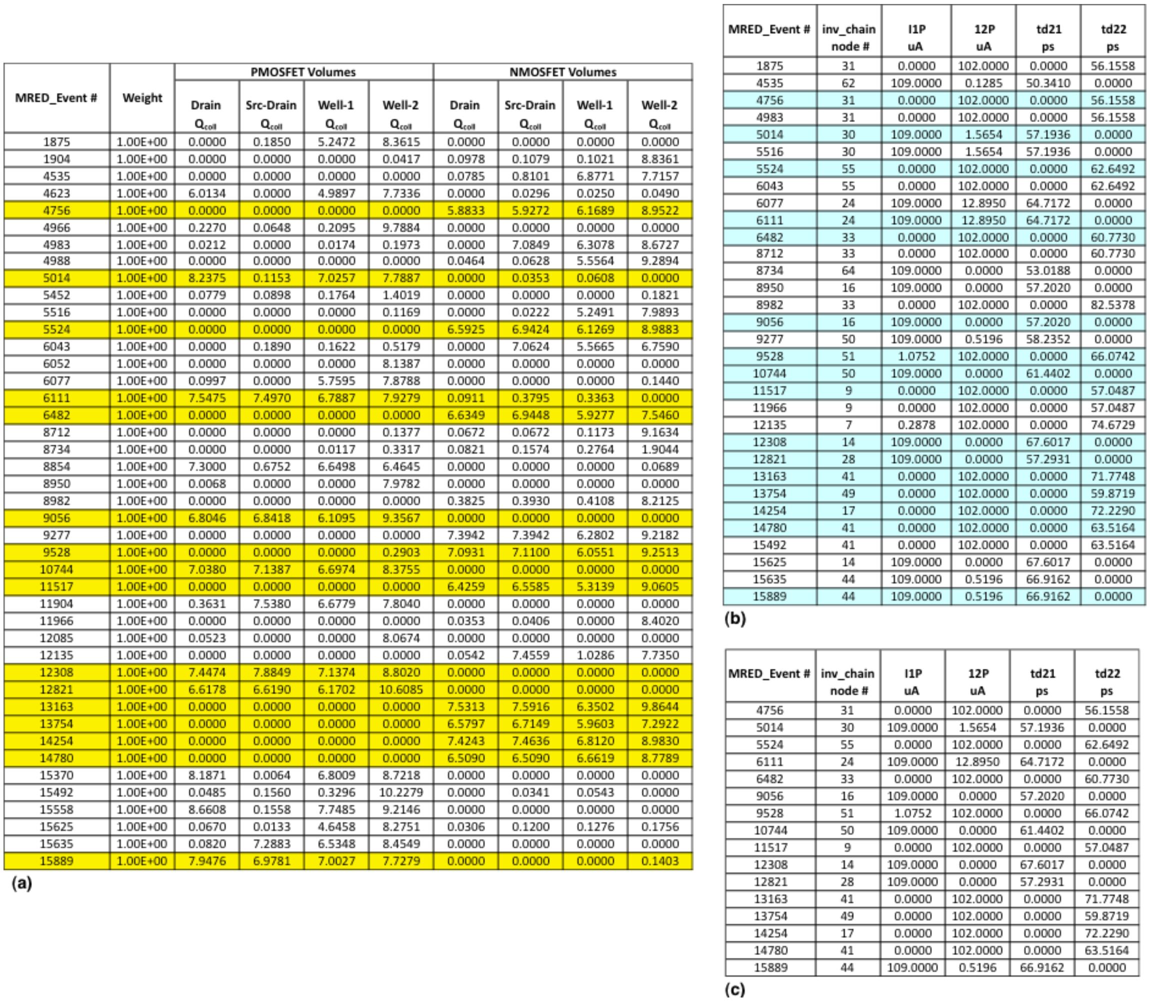


Figure 37. MRED2SPICE process flowchart.



MRED_Event #	Weight	PMOSFET Volumes				NMOSFET Volumes			
		Drain Q_{coll}	Src-Drain Q_{coll}	Well-1 Q_{coll}	Well-2 Q_{coll}	Drain Q_{coll}	Src-Drain Q_{coll}	Well-1 Q_{coll}	Well-2 Q_{coll}
1875	1.00E+00	0.0000	0.1850	5.2472	8.3615	0.0000	0.0000	0.0000	0.0000
1904	1.00E+00	0.0000	0.0000	0.0000	0.0417	0.0978	0.1079	0.1021	8.8361
4535	1.00E+00	0.0000	0.0000	0.0000	0.0000	0.0785	0.8101	6.8771	7.7157
4623	1.00E+00	6.0134	0.0000	4.9897	7.7336	0.0000	0.0296	0.0250	0.0490
4756	1.00E+00	0.0000	0.0000	0.0000	0.0000	5.8833	5.9272	6.1689	8.9522
4966	1.00E+00	0.2270	0.0648	0.2095	9.7884	0.0000	0.0000	0.0000	0.0000
4983	1.00E+00	0.0212	0.0000	0.0174	0.1973	0.0000	7.0849	6.3078	8.6727
4988	1.00E+00	0.0000	0.0000	0.0000	0.0000	0.0464	0.0628	5.5564	9.2894
5014	1.00E+00	8.2375	0.1153	7.0257	7.7887	0.0000	0.0353	0.0608	0.0000
5452	1.00E+00	0.0779	0.0898	0.1764	1.4019	0.0000	0.0000	0.0000	0.1821
5516	1.00E+00	0.0000	0.0000	0.0000	0.1169	0.0000	0.0222	5.2491	7.9893
5524	1.00E+00	0.0000	0.0000	0.0000	0.0000	6.5925	6.9424	6.1269	8.9883
6043	1.00E+00	0.0000	0.1890	0.1622	0.5179	0.0000	7.0624	5.5665	6.7590
6052	1.00E+00	0.0000	0.0000	0.0000	8.1387	0.0000	0.0000	0.0000	0.0000
6077	1.00E+00	0.0997	0.0000	5.7595	7.8788	0.0000	0.0000	0.0000	0.1440
6111	1.00E+00	7.5475	7.4970	6.7887	7.9279	0.0911	0.3795	0.3363	0.0000
6482	1.00E+00	0.0000	0.0000	0.0000	0.0000	6.6349	6.9448	5.9277	7.5460
8712	1.00E+00	0.0000	0.0000	0.0000	0.1377	0.0672	0.0672	0.1173	9.1634
8734	1.00E+00	0.0000	0.0000	0.0117	0.3317	0.0821	0.1574	0.2764	1.9044
8854	1.00E+00	7.3000	0.6752	6.6498	6.4645	0.0000	0.0000	0.0000	0.0689
8950	1.00E+00	0.0068	0.0000	0.0000	7.9782	0.0000	0.0000	0.0000	0.0000
8982	1.00E+00	0.0000	0.0000	0.0000	0.0000	0.3825	0.3930	0.4108	8.2125
9056	1.00E+00	6.8046	6.8418	6.1095	9.3567	0.0000	0.0000	0.0000	0.0000
9277	1.00E+00	0.0000	0.0000	0.0000	0.0000	7.3942	7.3942	6.2802	9.2182
9528	1.00E+00	0.0000	0.0000	0.0000	0.2903	7.0931	7.1100	6.0551	9.2513
10744	1.00E+00	7.0380	7.1387	6.6974	8.3755	0.0000	0.0000	0.0000	0.0000
11517	1.00E+00	0.0000	0.0000	0.0000	0.0000	6.4259	6.5585	5.3139	9.0605
11904	1.00E+00	0.3631	7.5380	6.6779	7.8040	0.0000	0.0000	0.0000	0.0000
11966	1.00E+00	0.0000	0.0000	0.0000	0.0000	0.0353	0.0406	0.0000	8.4020
12085	1.00E+00	0.0523	0.0000	0.0000	8.0674	0.0000	0.0000	0.0000	0.0000
12135	1.00E+00	0.0000	0.0000	0.0000	0.0000	0.0542	7.4559	1.0286	7.7350
12308	1.00E+00	7.4474	7.8849	7.1374	8.8020	0.0000	0.0000	0.0000	0.0000
12821	1.00E+00	6.6178	6.6190	6.1702	10.6085	0.0000	0.0000	0.0000	0.0000
13163	1.00E+00	0.0000	0.0000	0.0000	0.0000	7.5313	7.5916	6.3502	9.8644
13754	1.00E+00	0.0000	0.0000	0.0000	0.0000	6.5797	6.7149	5.9603	7.2922
14254	1.00E+00	0.0000	0.0000	0.0000	0.0000	7.4243	7.4636	6.8120	8.9830
14780	1.00E+00	0.0000	0.0000	0.0000	0.0000	6.5090	6.5090	6.6619	8.7789
15370	1.00E+00	8.1871	0.0064	6.8009	8.7218	0.0000	0.0000	0.0000	0.0000
15492	1.00E+00	0.0485	0.1560	0.3296	10.2279	0.0000	0.0341	0.0543	0.0000
15558	1.00E+00	8.6608	0.1558	7.7485	9.2146	0.0000	0.0000	0.0000	0.0000
15625	1.00E+00	0.0670	0.0133	4.6458	8.2751	0.0306	0.1200	0.1276	0.1756
15635	1.00E+00	0.0820	7.2883	6.5348	8.4549	0.0000	0.0000	0.0000	0.0000
15889	1.00E+00	7.9476	6.9781	7.0027	7.7279	0.0000	0.0000	0.0000	0.1403

(a)

MRED_Event #	inv_chain node #	I1P uA	12P uA	td21 ps	td22 ps
1875	31	0.0000	102.0000	0.0000	56.1558
4535	62	109.0000	0.1285	50.3410	0.0000
4756	31	0.0000	102.0000	0.0000	56.1558
4983	31	0.0000	102.0000	0.0000	56.1558
5014	30	109.0000	1.5654	57.1936	0.0000
5516	30	109.0000	1.5654	57.1936	0.0000
5524	55	0.0000	102.0000	0.0000	62.6492
6043	55	0.0000	102.0000	0.0000	62.6492
6077	24	109.0000	12.8950	64.7172	0.0000
6111	24	109.0000	12.8950	64.7172	0.0000
6482	33	0.0000	102.0000	0.0000	60.7730
8712	33	0.0000	102.0000	0.0000	60.7730
8734	64	109.0000	0.0000	53.0188	0.0000
8950	16	109.0000	0.0000	57.2020	0.0000
8982	33	0.0000	102.0000	0.0000	82.5378
9056	16	109.0000	0.0000	57.2020	0.0000
9277	50	109.0000	0.5196	58.2352	0.0000
9528	51	1.0752	102.0000	0.0000	66.0742
10744	50	109.0000	0.0000	61.4402	0.0000
11517	9	0.0000	102.0000	0.0000	57.0487
11966	9	0.0000	102.0000	0.0000	57.0487
12135	7	0.2878	102.0000	0.0000	74.6729
12308	14	109.0000	0.0000	67.6017	0.0000
12821	28	109.0000	0.0000	57.2931	0.0000
13163	41	0.0000	102.0000	0.0000	71.7748
13754	49	0.0000	102.0000	0.0000	59.8719
14254	17	0.0000	102.0000	0.0000	72.2290
14780	41	0.0000	102.0000	0.0000	63.5164
15492	41	0.0000	102.0000	0.0000	63.5164
15625	14	109.0000	0.0000	67.6017	0.0000
15635	44	109.0000	0.5196	66.9162	0.0000
15889	44	109.0000	0.5196	66.9162	0.0000

(b)

MRED_Event #	inv_chain node #	I1P uA	12P uA	td21 ps	td22 ps
4756	31	0.0000	102.0000	0.0000	56.1558
5014	30	109.0000	1.5654	57.1936	0.0000
5524	55	0.0000	102.0000	0.0000	62.6492
6111	24	109.0000	12.8950	64.7172	0.0000
6482	33	0.0000	102.0000	0.0000	60.7730
9056	16	109.0000	0.0000	57.2020	0.0000
9528	51	1.0752	102.0000	0.0000	66.0742
10744	50	109.0000	0.0000	61.4402	0.0000
11517	9	0.0000	102.0000	0.0000	57.0487
12308	14	109.0000	0.0000	67.6017	0.0000
12821	28	109.0000	0.0000	57.2931	0.0000
13163	41	0.0000	102.0000	0.0000	71.7748
13754	49	0.0000	102.0000	0.0000	59.8719
14254	17	0.0000	102.0000	0.0000	72.2290
14780	41	0.0000	102.0000	0.0000	63.5164
15889	44	109.0000	0.5196	66.9162	0.0000

(c)

Figure 38. Output samples of data files for: (a) MRED Q_{coll} source (b) SPICE-generated SET results, and (c) SPICE latch SET results. The yellow highlighted events from (a) result in a generated output highlighted in blue in (b), and the final SET latched errors are in shown in (c).

MRED2SPICE – Comparison of Experimental Data for CMOS Combinational Cells

MRED Inputs for IBM 90-nm Technology

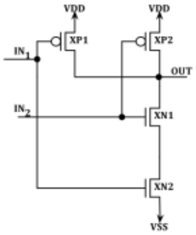
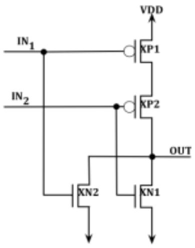
The MRED nested sensitive volumes used in this study for the 90-nm IBM technologies are given in Chapter IV. The sensitive volumes are defined identically for the three specific combinational cells (INVx1, NAND2x1, and NOR2x1) from this library. Experimental data published in Cannon et al. [16] is used for calibration of the sensitive volumes and their efficiencies. Each PMOSFET and NMOSFET transistor (or set of transistors in the same active area) contains the following four sensitive volumes: (1) Drain: cross-section defined by the drain layout, depth of 0.35 μm, (2) Src_drain: cross-section defined by the source_drain region, depth of 0.35 μm, (3) Well-1: cross-section defined by the drain plus consistent extensions in x and y, depth of 0.3 μm starting at the bottom of the active area, and (4) Well-2: cross-section defined by the active area plus consistent extensions in x and y, depth of 0.4 μm starting at the bottom of the active area.

The NAND2x1 has two PMOSFETs in parallel that have a shared drain, so that is handled as a single drain for charge deposition purposes. The NAND2x1 also has two NMOSFETs in series, so there is an output drain and an intermediate drain. The intermediate drain can be included in the sensitive volume depending on the NAND2x1 input configuration. The same holds for the NOR2x1 though the PMOSFET and NMOSFET situations are reversed. In total, there were 4 charge collection efficiencies in each type of transistor (NMOSFET/PMOSFET) for a total of eight efficiencies to modify for comparison to the single event SET data. This is consistent with the methodology in Chapter IV.

MRED2SPICE Simulation

The charge deposited in each sensitive volume is summed by specific charge collection efficiencies to determine the total amount of charge collected in each transistor type. This charge is then used to create independent current sources for simulation in SPICE. For low-level charge injection, two independent current sources in parallel are applied. DasGupta et al. [42] and Kauppila et al. [41] show current pulses with a high prompt component and a sustained current shelf. The prompt component is the charge necessary to raise the node from V_{SS} to V_{DD}, or vice versa, and is a factor of the restoring current and the node capacitance. The current shelf is the current necessary to maintain the voltage at the opposite potential and is a factor of only the restoring current. Two values, Q_{thresh} (charge necessary to flip the node potential) and I_{Hold} (current to hold the voltage at the opposite potential), are determined with SPICE simulations. The results of these simulations are given in Table 8. There are three different configurations given for the NAND2x1 and NOR2x1 circuits: (1) a chain of circuits connecting the output of one cell to the first input of the following cell with other input tied high or low, (2) a chain of circuits connecting the output of one cell to the second input of the following cell with the other input tied high or low, and (3) a chain of circuits connecting the output of one cell to both inputs of the following cell.

Table 8. Simulated threshold charges (Q_{thresh}) and hold currents (I_{Hold}) for the combinational cells for comparison to experimental data given different test configurations

Cell	Schematic	Chain Configuration	Vulnerable Device	Q_{thresh} (fC)	I_{Hold} (μA)
INVx1		Standard Inverter Chain	PMOSFET	4.8	121
			NMOSFET	4.6	118
NAND2x1	 VDD VDD XP1 XP2 IN1 OUT IN2 XN1 XN2 VSS	IN_1 = "Chain", IN_2 = V_{dd}	Either PMOSFET	3.3	67
			Either NMOSFET	4.1	69
		IN_1 = V_{dd}, IN_2 = "Chain"	Either PMOSFET	3.4	67
			NMOSFET #1	3.4	69
		IN_1 = IN_2 = "Chain"	Either PMOSFET	4.1	69
			Either NMOSFET	5.6	137
NOR2x1		**(v1)** IN_1 = "Chain", IN_2 = V_{ss}	Either PMOSFET	8.8	122
			Either NMOSFET	7.4	113
		(v2) IN_1 = V_{ss}, IN_2 = "Chain"	PMOSFET #2	6.5	121
			Either NMOSFET	6.5	113
		(v3) IN_1 = IN_2 = "Chain"	Either PMOSFET	10.7	242
			Either NMOSFET	8.8	114

The circuits simulated in SPICE are duplicates of the circuits tested in [16]. These consist of 65 combinational cells followed by delay elements, a guard gate, and an asynchronous latch. An example of the INVx1 circuit is shown in Figure 39.

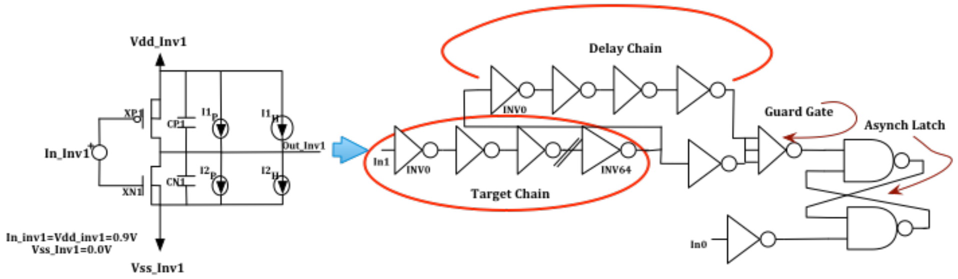


Figure 39. Example core circuit for SET characterization test structure.

MRED2SPICE randomly selects a node to apply the single event and checks to determine if the latch has changed states. If so, then it records the event. The simulation flow allows the MRED2SPICE method to recreate the single event experiment.

Comparison of MRED2SPICE Model to Experimental Data

Nested sensitive volumes are used to determine the charge collected for the inverter, NAND gate, and the NOR gate for a variety of ion species and energies. Figure 40, Figure 41, and Figure 42 compare the MRED2SPICE predictions of SET cross section to the experimental data presented by Cannon et al. [16]. These data show that, for the most part, the MRED2SPICE predictions are in closer agreement with the experimental data at lower LETs than at higher LETs. The lack of agreement at the highest LETs is most likely do the limited applicability of the simple double-exponential current source, e.g., it does not contain appropriate terms to model multi-node charge collection. These figures demonstrate that the model shows an increase in SET pulse-width vs. LET as well as high variations in the pulse-width under specific ion test conditions.

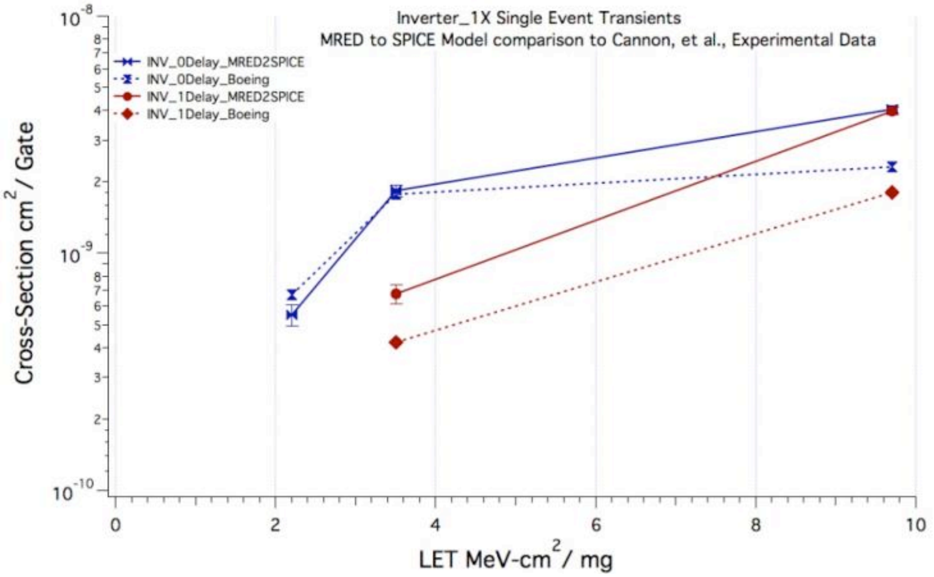


Figure 40. SET cross-section of 1X drive strength for inverter as a function of LET. MRED2SPICE predictions are drawn with solid lines and SEE data is drawn with dashed lines.

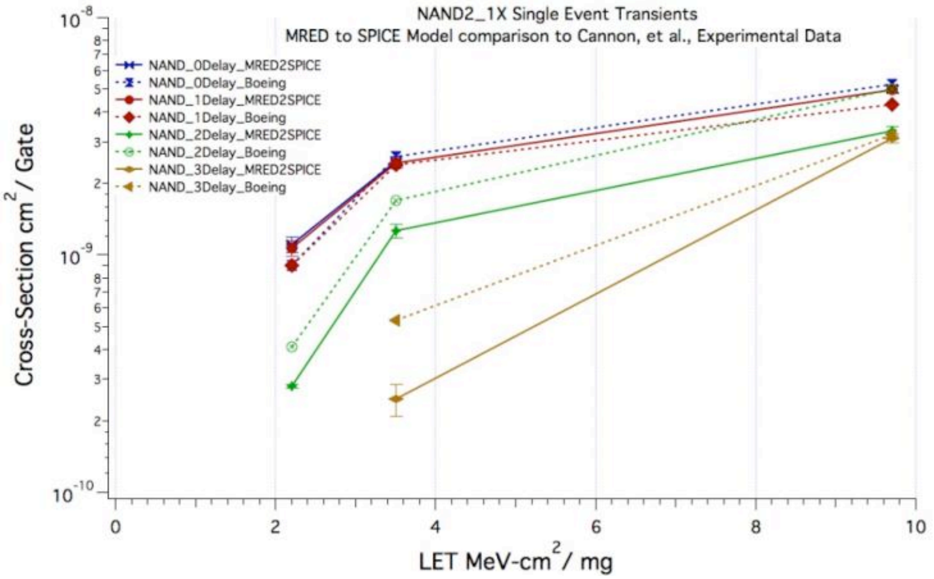


Figure 41. SET cross-section of 1X drive strength of 2-input NAND gate (1st input chained, 2nd input tied to V_{dd}) as a function of LET. MRED2SPICE predictions are drawn with solid lines and SEE data are drawn with dashed lines.

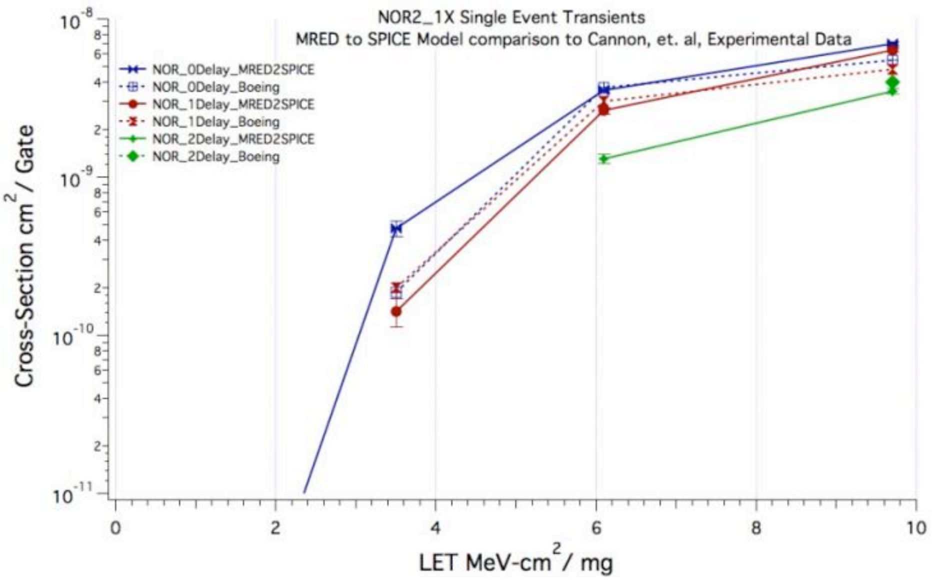
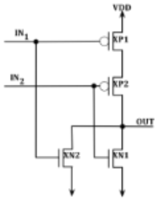


Figure 42. SET cross-section of 1X drive strength for 2-input NOR gate (1st input chained, 2nd input tied to V_{ss}) as function of LET. MRED2SPICE predictions are drawn with solid lines and SEE data are drawn with dashed lines.

Applications of MRED2SPICE Model

One application for this model is the ability to predict the SET response for a logic cell and its different input configurations when the gate accepts multiple inputs. An inverter chain only has one option, while 2-input gates (i.e., IN_1, IN_2) can receive one or both inputs from the previous gate. Figure 43 illustrates the three different configurations listed in Table 9.

Table 9. Input Configurations for 2-Input NOR gate

Cell	Schematic	Chain Configuration	Vulnerable Device	Q_{thresh} (fC)	I_{Hold} (μA)
NOR2x1		**(v1)** IN_1 = "Chain", IN_2 = V_{ss}	Either PMOSFET	8.8	122
			Either NMOSFET	7.4	113
		(v2) IN_1 = V_{ss}, IN_2 = "Chain"	PMOSFET #2	6.5	121
			Either NMOSFET	6.5	113
		(v3) IN_1 = IN_2 = "Chain"	Either PMOSFET	10.7	242
			Either NMOSFET	8.8	114

The first chain configuration in Table 9, v1, is designed with IN_1 receiving its signal from the logic chain, while IN_2 is tied to V_{ss}. Another configuration, **v3**, is designed with IN_1 tied to IN_2 and also tied to the logic chain. These two configurations have the output drain electrically connected to the intermediate drain. Therefore, these two configurations have the highest drain cross-section. The last configuration, v2, is designed with IN_1 tied to V_{ss} while IN_2 is tied to the logic chain. This configuration has the intermediate drain electrically connected to the source, so it has the smallest drain cross-section. The intermediate drain layout is designed three times as large as the output drain. In Table 9, Q_{thresh} is the smallest value for Configuration v2. Therefore, it has the highest cross-section at the lower LETs. Finally, all of these chains attenuate the SET pulse as it propagates down the chain with **v3** having the largest attenuation. Pulse attenuation gives an apparent decrease in cross-section and that is why the **v3** configuration is lower than the **v1** configuration.

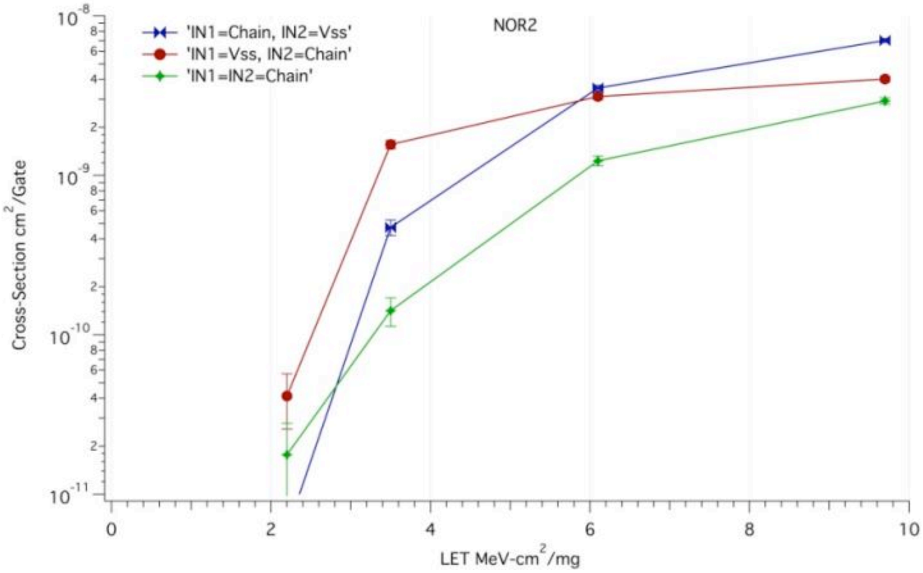


Figure 43. SET cross-section of 1X drive strength NOR2 MRED2SPICE results with different input chain configurations from Table 9.

MRED2SPICE Method Summary

The MRED2SPICE simulations allow analysis of SET experiments on combinational logic chains when the incident particles produce low levels of charge deposition. The models can predict the SET response of the combinational logic given different input configurations. The MRED2SPICE multi-scale simulation can also predict the SET response of other combinational logic cells in the same technology.

CHAPTER VII

TRANSIENT FAULT ANALYSIS FOR SEQUENTIAL CAPTURE IN DUAL-COMPLEMENTARY FLIP-FLOPS

The multi-scale simulation approach not only must consider transients within the combinational logic, but also must evaluate the effects of sequential logic. This chapter discusses the impact of SETs on a dual-complementary D-type Flip-Flop (DC-DFF). The internal structure is based upon a standard two-input NAND function discussed in Chapter V. Circuit-level modeling indicates that the DC-DFF is resistant to single event transient (SET) capture of errant signals on the data lines while increasing the operating speed to gigahertz frequencies. However, the simulations also predict that the DC-DFF is susceptible to internal single events during data transitions. Heavy ion testing verified the simulations of the internal single-event mechanism in the DC-DFF design.

BACKGROUND

Flip flops are suspectable to two clock-rate-dependent single event upsets (SEUs). The first mechanism is a circuit response to an SET that propagates to the data input of the DFF [49, 51]. The SET in this case must arrive during the window of vulnerability, i.e., the time when the data input must be stable before (i.e., the setup time) or after (i.e., the hold time) the active clock edge. SETs outside of this window will not be captured by the DC-DFF and will not show up as an SEU on the output. The second mechanism is a circuit response to a single event on: (1) the clock input, (2) internal clock buffering, or (3) other controls such as set, reset, preset, and clear [50]. Ion strikes affecting the clock can sample data at the wrong time and thus capture the input data before it settles to the correct value.

DUAL COMPLEMENTARY DFF (DC-DFF)

The DC-DFF is a modification of a 90-nm IBM Dual Interlocked Cell (DICE) design [52], implemented to increase the speed of its operations due to faster switching between the complementary logic. It also makes use of its four internal nodes to transmit data. The internal structure is based upon a standard two-input NAND function (i.e., NAND2), however some input connections are changed to enable redundancy within the internal nodes.

Cell Description

Typical DFFs are designed with one data input, d, and two complementary outputs, q and nq. A shift register can be constructed from the DC-FFs. The distinguishing characteristic of the DC-DFF shift register configuration is that the four storage nodes (qa, qb, nqa, and nqb) are output from one DC-DFF to the four input ports (da, db, nda, and ndb) of the next DC-DFF (similar to Figure 44). All connections internal to an individual DC-DFF are also dual complementary. The DC-DFF is designed as a master-slave DFF and is created by connecting two D-latches in series. It is called master-slave because the second latch in the series only changes in response to a change in the first (master) using a non-overlapping clock. Figure 45 shows the input circuit for either the master or slave of the DC-DFF and is followed by the memory circuit (Figure 46). Both of the figures combined (32 transistors in all) form a single D-latch and are one-half of the total DC-DFF cell.

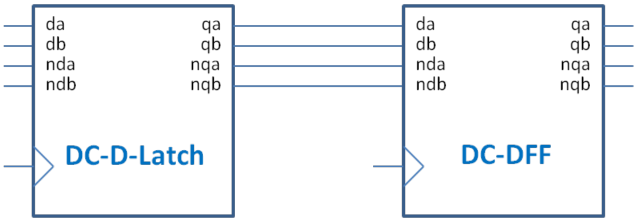


Figure 44. Block diagram of circuits for single event simulation of DC-D-Latch.

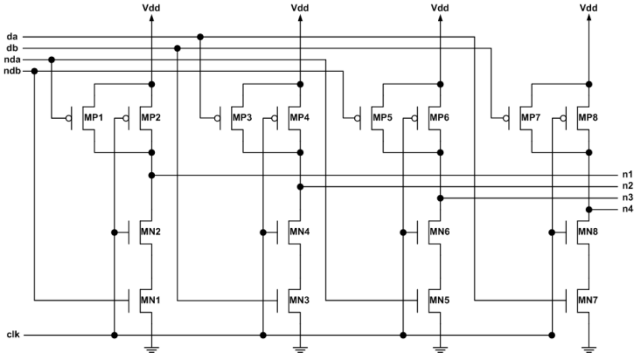


Figure 45. Dual-Complementary DFF input circuit showing internal connections.

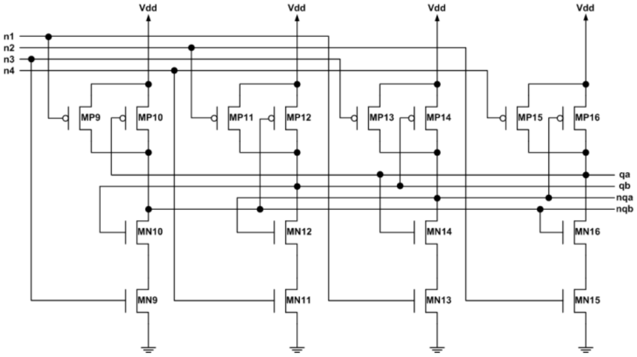


Figure 46. Dual-Complementary DFF memory circuit showing internal connections.

Cell Write Operation

The nominal operation of the cell can be understood by examining the procedure for overwriting the state of the memory portion of the DC-DFF. Assume that the memory circuit in Figure 46 (illustrated by simple switches in Figure 47) is holding qa and qb high (logic state 1) and nqa and nqb low (logic state 0). In the hold state, the clock input into the input circuit (Figure 45) is low, driving the internal connections (n1, n2, n3, and n4) high or logic state 1; this configuration is demonstrated with closed switches in Figure 47. This makes the even-numbered transistors in Figure 46 (MP10/MN10, MP12/MN12, MP14/MN14 and MP16/MN16) or their equivalent switches in Figure 47 (qa!/qb, nqb!/nqa, qb!/qa and nqa!/nqb) in the memory circuit operate in traditional DICE storage operation. The odd-numbered transistors in Figure 46 (MP9/MN9, MP11/MN11, MP13/MN13 and MP15/MN15) are otherwise known as the access transistor pairs of the traditional DICE storage, with their equivalent switches in Figure 47 (n1!/n3, n2!/n4, n3!/n1 and n4!/n2). These switches provide the control to determine if the memory circuit should hold its existing state or sample a new state through the input circuit of the DC-DFF.

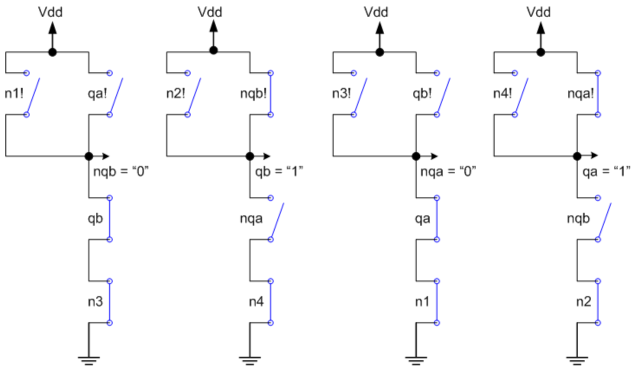
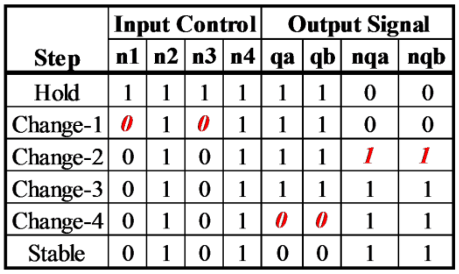


Figure 47. Memory Circuit - start state qa/qb holds logic state 1.

Table 10 and Figure 48 show the input controls, output signals, and the results of the transistor conditions to overwrite the stored data value. The first row (Hold) is the condition just before the clock input goes from logic state 0 to logic state 1. The next five rows show the progression of how the transistors change. The red items in each row are the changes from the previous row. The final row (Stable) means that the clock could return to low, and the new state will remain in the memory circuit. The table shows that changing the stored value from logic state 1 to logic state 0 begins via closing the switches of n1! and n3! (i.e., pulling up the internal nodes nqa and nqb) and conversely opening the switches n1 and n3, thereby changing the output signals. Similarly, changing the memory state from logic state 0 to logic state 1 is accomplished in reverse.

Table 10. Input controls, output signals, and transistor conditions required to change DC-DFF memory circuit from logic state 1 to logic state 0.



Step	Input Control				Output Signal			
	n1	n2	n3	n4	qa	qb	nqa	nqb
Hold	1	1	1	1	1	1	0	0
Change-1	*0*	1	*0*	1	1	1	0	0
Change-2	0	1	0	1	1	1	*1*	*1*
Change-3	0	1	0	1	1	1	1	1
Change-4	0	1	0	1	*0*	*0*	1	1
Stable	0	1	0	1	0	0	1	1

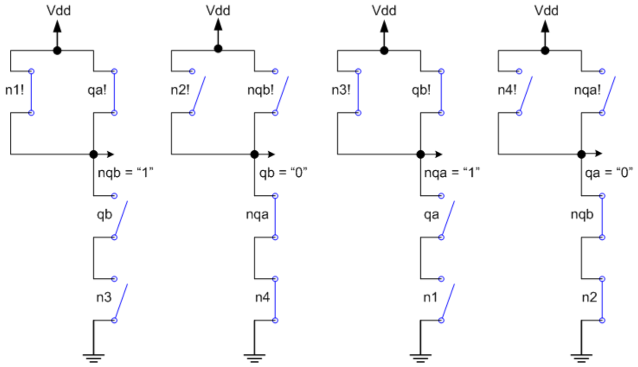


Figure 48. Results from memory circuit normal operation - circuit will settle to logic state qa/qb = 0, (i.e., no error).

SINGLE EVENT TRANSIENT CIRCUIT SIMULATION

Cadence Spectre was used to simulate the input and memory circuits (Figure 45 and Figure 46) with various stimuli on the input ports (clk, da, db, nda, and ndb). The simulation includes a full DC-DFF connected to the output ports, as shown in Figure 49. This simulation describes the behavior of a shift register. In this model, all transistors in the input and memory circuits can be simulated with a current source between the drain and body, which represents a heavy ion strike to the transistor. The circuit simulation determines: (a) if an SET generated in either the input or memory circuit would result in an SEU in the following DC-DFF, (b) if an SET on the clock input resulted in an SEU, or (c) if an SET produced by an ion strike in either the input or memory circuit could prohibit the propagation of the correct data down a shift register.

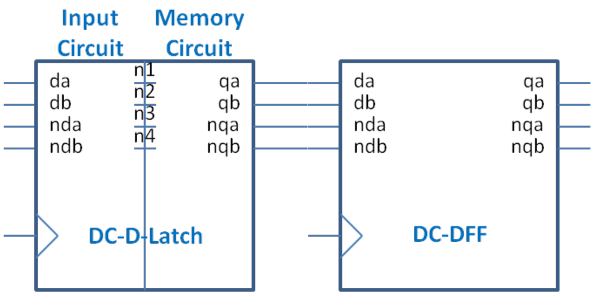


Figure 49: Block diagram of circuits for single event simulation of Dual-Complementary D-Latch. The current sources for single event modeling are placed in the input and memory circuits.

A double exponential current source is used to represent an SET in all of the simulations presented in this chapter. The damping factor for the rise of the current pulse is 50 ps. The length of the pulse is sufficent to cause errors between clock edges. The amplitude of the current is 1 mA, so that the effect of the current is saturated in the circuit level simulation. The tail current damping factor is 500 ps, see Figure 50.c (IO_sink).

Data Line SET

SETs on the data line were simulated with the input and memory circuits in static mode. The SET was generated at internal nodes. Simulations were performed to determine if the SET was latched by the next DC-DFF. Several conditions were evaluated including: (1) an initial logic state 0 in the DC-DFF with a next logic state 0, (2) an initial logic state 0 with a next logic state 1, (3) an initial logic state 1 with a next logic state 0, and (4) an initial logic state 1 with a next logic state 1. The first condition propagates a constant logic 0 through the shift register, while the last condition propagates a constant logic 1. The middle two conditions represent alternating logic states that propagate through the shift register. No soft errors were observed for any of these conditions on the output of the DC-DFF.

An example of the alternating data set of simulations results is shown in Figure 50. Figure 50.a is the input clock – clk, Figure 50.b is one of four alternating input data signals – da, Figure 50.d output signal – nqb, Figure 50.e output signal – qb, Figure 50.f output signal – nqa, Figure 50.g output signal – qa, and Figure 50.h is the shifted output data signal – shift_qa. This ordering is used in similar figures later in the chapter.

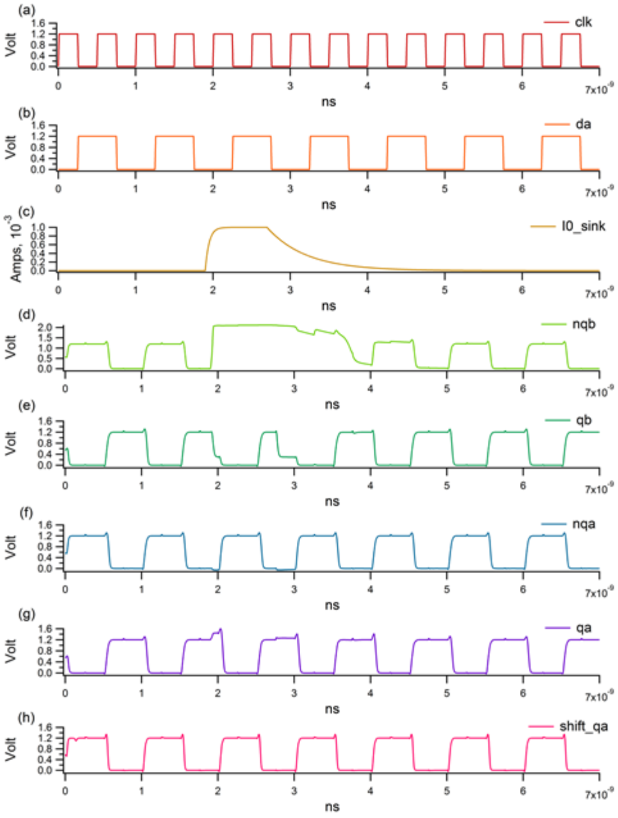


Figure 50. Complementary data - no error (b) matches (h), change of logic state 0 to 1, or logic state 1 to 0; (a) input clock pulse, (b) input state, (c) ion strike, (d) through (g) internal storage nodes, (h) output data.

Clock Line SET

SETs on the clock line were analyzed for this circuit (the clock line was not hardened to SETs). Clock-line SET-induced upsets will not occur if the input data signal is held constant (logic state 1 or 0). However, when alternating logic state from 1 to 0 back to 1 (or vice versa), a clock-line SET occurring at the incorrect time may change the stored state early or cause failure to sample the logic state correctly. In general, SETs on the clock lines could show up when the input data value is alternated, but not when the input is constant. A comparison of bit errors that occur during switched input versus constant input can determine errors induced on the clock distribution circuit. If more upsets are observed when using alternating data as opposed to constant data, then clock-line SETs are contributing to this increased upset rate.

Internal DC-DFF Single Event

SETs internal to the DC-DFF that prohibit the memory circuit from loading the proper state were simulated similarly to the SETs on the data lines. The circuit in Figure 49 was operated in a shift register fashion. As previously stated, there were no errors observed in the SET on the data line simulations (Figure 50); errors observed here result from ion strikes in the DC-DFF, causing the DC-DFF input circuit to be unable to write the correct data into the memory circuit. All four conditions defined above were simulated: a constant input of either logic state 0 or 1 and two alternating data patterns (1 → 0 and 0 → 1). There were no soft errors for the constant data condition, therefore the design of the DC-DFF is hardened against SETs when the data are constant.

There are many sources of soft errors for the alternating data patterns. If more upsets are observed for alternating data as opposed to constant data, then internal DC-DFF single events will contribute to this increase in upset sensitivity. These internal DC-DFF single events represent a new clock dependent mechanism, and will be discussed in more detail later.

Input Circuit Single Events

In the alternating data case, every single event on every transistor in the input circuit (Figure 45) could block the change of state in the memory circuit. To illustrate one of these errors, consider an SET affecting input control n1. A single event on the transistors driving n1 could prohibit the node from pulling down and remain at logic state 1. Table 11, Figure 51, and Figure 52 show what occurs in the memory circuit under this condition. The items highlighted in blue (or bold/italic font) show the differences from the non-single-event case shown in Table 10. The easiest way to understand this mechanism is to examine the input controls to the memory circuit. Only one control signal changes (i.e., n3) on the DC-DFF. DICE-like cells are resistant to change from perturbations on a single node by design [52]. The net result is one internal node being in contention (C) and one node floating (F).

Contention is defined as a short from power (Vdd) to ground (GND) caused by both PMOSFETs and NMOSFETs being turned on at the same time. In this example, the DC-DFF circuit will return to its original state. Thus, there is no error if the access transistors are not attempting to change the state, while an error occurs (Figure 52) if the intent is to change the storage state.

Table 11. Input control, output signal, and transistor conditions required to change DC-DFF memory circuit from logic state 1 to logic state 0. A single event to the input circuit keeps n1 at logic state 1.

	Input Control				Output Signal			
Step	n1	n2	n3	n4	qa	qb	naq	nqb
Hold	1	1	1	1	1	1	0	0
Change-1	*1*	1	0	1	1	1	0	0
Change-2	*1*	1	0	1	1	1	*C*	*F*
Change-3	*1*	1	0	1	1	1	*C*	*F*
Change-4	*1*	1	0	1	*1*	*1*	*C*	*F*
Unstable	*1*	1	0	1	*1*	*1*	*C*	*F*

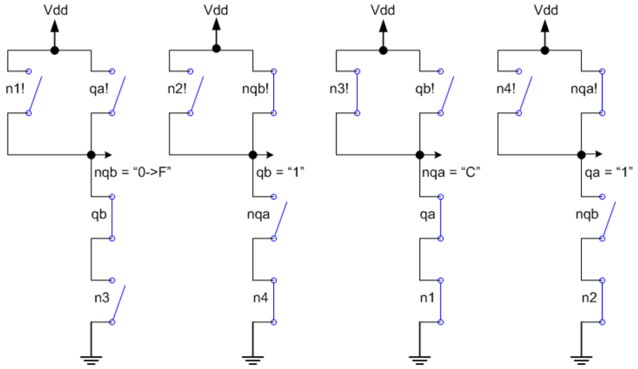


Figure 51. Results from input circuit single event - circuit will settle to logic state qa/qb =1, which is an error.

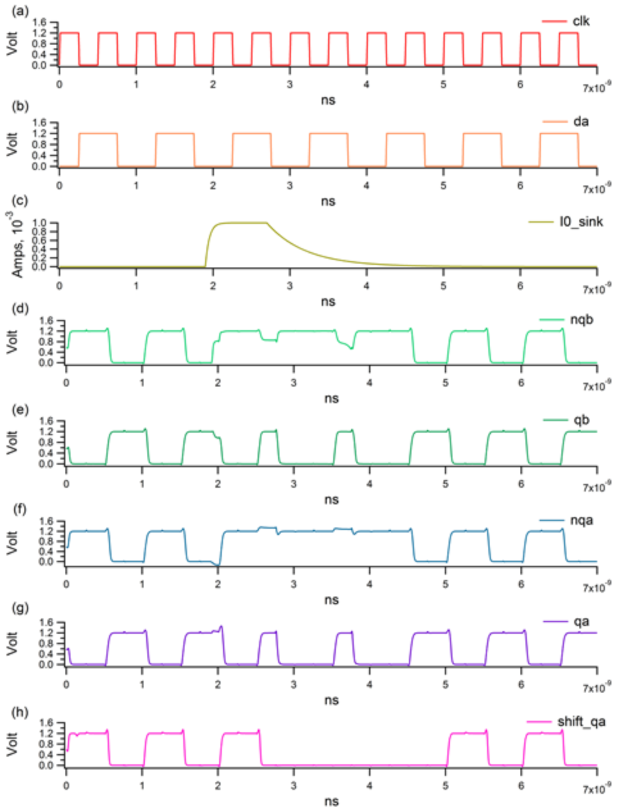


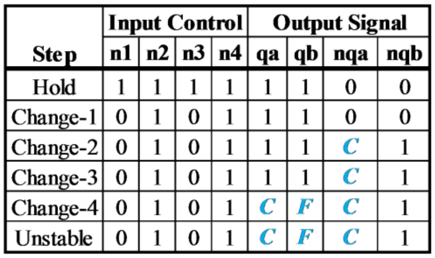
Figure 52. Complementary input data with error located between 3 ns and 5 ns - single event on input circuit.

The simulation results show that single events on both PMOSFETs and NMOSFETs in the input circuit prevent the input data from being written to the DC-DFF. The simulation results also show that this would only occur in one direction, either on the data transition from logic state 0 to logic state 1 or from logic state 1 to logic state 0. This is dependent on the location of the transistor in the input circuit design. In Figure 52, the input data transition from logic state 0 to logic state 1 at the time of the single event (at 2.5 ns) is output correctly. However, on the next data input transition from logic state 1 to logic state 0, the output data is blocked and therefore, incorrectly output.

Memory Circuit Single Events

The mechanism in the memory circuit is not the same as the input circuit. The constant input conditions do not cause an soft error. For switched data, single events on the PMOSFETs in the memory circuit do not cause any soft errors, and single events on the NMOSFETs in the memory circuit all caused soft errors. Table 12, Figure 53, and Figure 54 show the simulation results for a single event to an NMOSFET in the memory circuit, i.e., MN13 (Figure 46). In the example, the transistor is struck while it is on. When the two input controls, n1 and n3, pull down, the internal node nqa goes into contention (C). As the transition continues, another internal node goes into contention (C), qa, and one floats (F), qb. Therefore, only one storage node is correct. This state leads to a very unstable condition for the cell, which will recover over time. The recovery mechanism can be complicated when various nodes are in contention.

Table 12. Input control, output signal and transistor conditions required to change DC-DFF memory circuit from logic state 1 to logic state 0. A single event to the memory circuit keeps transistor MN13 on, thereby keeping qa closed.



Step	Input Control				Output Signal			
	n1	n2	n3	n4	qa	qb	nqa	nqb
Hold	1	1	1	1	1	1	0	0
Change-1	0	1	0	1	1	1	0	0
Change-2	0	1	0	1	1	1	*C*	1
Change-3	0	1	0	1	1	1	*C*	1
Change-4	0	1	0	1	*C*	*F*	*C*	1
Unstable	0	1	0	1	*C*	*F*	*C*	1

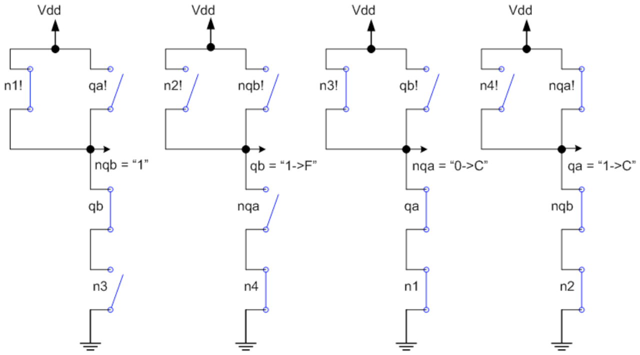


Figure 53. Results from memory circuit single event – circuit will settle to logic state qa/qb = 1, which is an error.

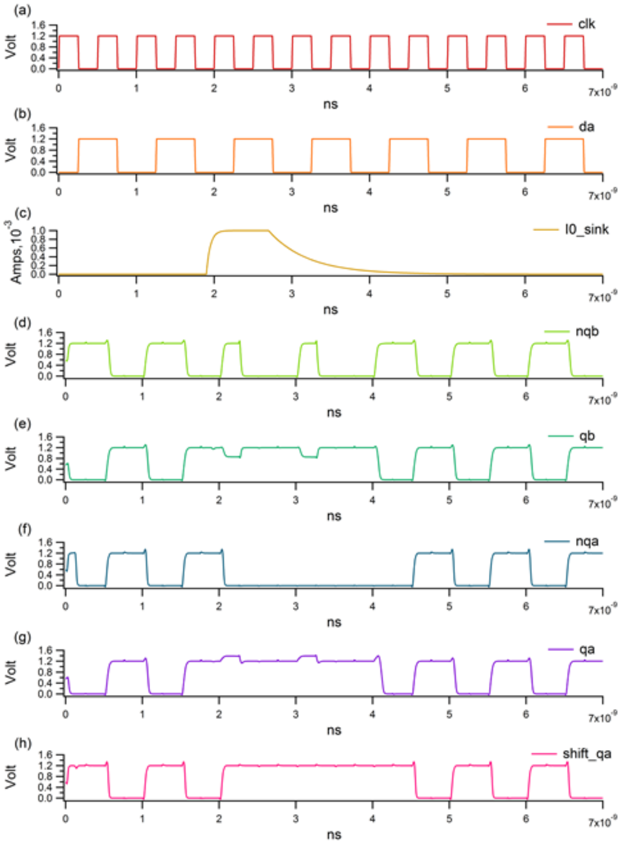


Figure 54. Complementary input data with error between 2.5 ns and 4.5 ns – single event on memory circuit.

Simulation of the memory circuit show a difference in the PMOSFET and NMOSFET single event response. No errors are observed for single events occurring on PMOSFETs in the memory circuit. However, if a single event occurred on the NMOSFETs, an error resulted in all the simulations. Changing the value stored in the DC-DFF begins by pulling up the internal nodes. Errors result because ion strikes to NMOSFETs block the pull-up action, while ion strikes to PMOSFETs will not block that action.

HEAVY ION TESTING OF DUAL-COMPLEMENTARY DFFS

The DC-DFF cells were used to design a shift register string using IBM's CMOS9SF process. The circuit includes built-in self-test (BIST) to evaluate the DC-DFF at high frequencies [48, 53]. Two circuit layouts (standard and guard-band) are evaluated during single event effects testing.

V-CREST Test Chip Design

The shift register string design was based on the Circuit for Radiation Effects Self Test (CREST) as originally described by Marshall et al. [53]. This original concept implements built-in self-test (BIST) of SiGe flip-flops (FFs) to measure single event upset in shift register stings. It enables SEU experiments at high operating speeds, reduced challenges with test chip input/output pad design and packaging, and removed requirements for high frequency cabling at the test facilities. It accomplishes these features by integrating the clock and the data pattern from a 127-bit pseudo-random number generation onto the test chip. It also moves the error/upset detection circuit onto the test chip. This integration significantly reduces the switching frequency needed for the test chip input/output.

The original CREST design was expanded to accommodate bulk CMOS FFs; this will be referred to as V-CREST [48]. A block diagram of V-CREST is shown in Figure 55. Since the SEU sensitive area of bulk CMOS FFs is smaller than that of heterojunction bipolar transistor (HBT) SiGe FFs, the shift register chain needed to include a large number of FFs. The shift registers were implemented using 584 FFs, which were designed in groups of four, where all four types of shift registers were powered and clocked together.

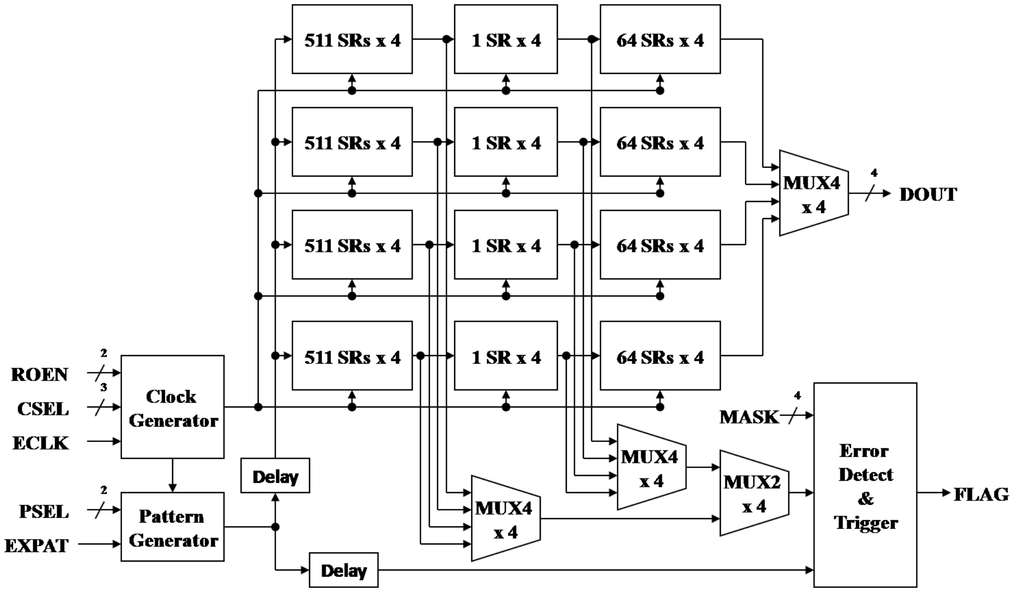


Figure 55. V-CREST block diagram.

The V-CREST circuit can distinguish between errors caused by SETs in the clock path versus those in the data path [48, 54]. The clock distribution network uses its last buffer to drive four FFs. SETs on the clock distribution network would affect all four FFs, thereby distinguishing SETs in the clock distribution network from other errors. If all four FF shift registers showed an error, then the error flag is not triggered. Finally, the clock distribution design includes both the clock and its inverse so that clock signals are not generated within a FF.

Two different DC-DFF layouts were evaluated, one utilized guard-bands and the other did not [54, 55]. The transistor placement was the same for both designs. The size of each layout was also exactly the same.

Heavy Ion Test Results

The DC-DFFs were tested at the 88-inch cyclotron at Lawrence Berkeley National Laboratories. Electrical measurements were made before and after each exposure to verify the samples were functional and within electrical parameter limits. The de-lidded test devices were exposed to a range of beam conditions (energy and species). Test routines exercised various functions of the shift register. During each exposure the device outputs and supply current were monitored for erroneous conditions.

The fluence for each test was selected so that at least 30 errors were observed for each exposure. In test runs where no errors were observed and the accumulation of total dose permitted, the minimum fluence was 10^7 particles/cm^2. The ions used in heavy ion testing of the DC-DFF are summarized in Table 13.

The SEU cross-section per bit versus LET is shown in Figure 56. Only one data point is shown for a constant input (0000) with a 150 MHz operating frequency. The other data points represent alternating data (1010) either without or with a guard band (GB).

Table 13. Heavy ions, LETs and ion energies used to test DC-DFFS.

Ion	LET (MeV-cm^2/mg)	Energy (MeV)
Ne	5.76	90
Ar	14.33	180
Cu	30.04	284
Kr	38.25	387
Xe	68.50	612

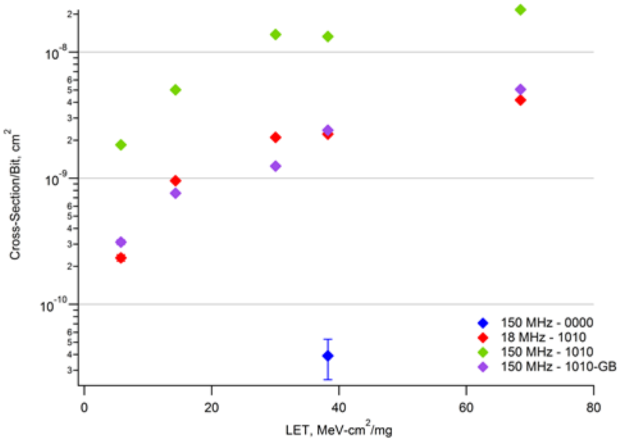


Figure 56. Upset cross-section versus LET for DC-DFF layouts at two different clock frequencies.

CLOCK DEPENDENT MECHANISMS

The susceptibility of the input and memory circuits was analyzed by simulating ion strikes to all circuit nodes. The simulations show the DC-DFF is not susceptible to data line SETs but is susceptible to clock line SETs and internal DC-DFF single events. The heavy ion testing of the DC-DFF shows a much higher cross-section when propagating the alternating input data versus constant input data when clocked. Therefore, the higher cross-section can only be explained by clock line SETs and internal DC-DFF single events.

The SEU cross-section, Figure 56, for the guard-band design (150 MHz – 1010-GB) is lower than that for the non-guard-band design (150 MHz – 1010). The guard band was not applied to the transistors in clock distribution circuitry. Therefore, for the standard design, the internal DC-DFF single event upsets are the largest contributor to the cross-section. If the clock line SETs were the larger contributor, then the cross-sections would be approximately equal because the clock distribution network on the chip was exactly the same. Therefore, internal DC-DFF single event upsets are clock-dependent and are a significant contributor to the error cross-section.

The simulation results reveal that all 32 of the NMOSFETs and PMOSFETs in the two input circuits and all 16 of the NMOSFETs in the two memory circuits were susceptible to this mechanism. Therefore, 48 potential transistors factor into the upset cross-section for this mechanism. Also, SETs that prevent a state change can be longer than SETs that change the state. This factor leads to an increased probability of occurrence.

NAND2 logic cells were simulated as the fundamenal building block of the DC-DFF to determine the generated SET pulse widths as a function of collected charge. Dasgupta et al. [42] describe SET currents with a prompt and a shelf component, and that model was implemented with dual double-exponential current sources, one for the current shelf and one for the prompt component. The shelf level was a function of the node's restoring current and was independent of the node's capacitance, but the prompt component was dependent upon both the restoring current and the node capacitance. If the load on a circuit node is increased, then only the prompt current source will change. When an SET attempts to prevent a state change, it does not have to overcome the node's capacitance, so there is no prompt current source.

Figure 57 shows simulated SET pulse widths versus collected charge for the NAND2 circuit. The ON line represents the prevention of a state change and is independent of loading. The single event simulation of the ON line occurs at the moment when the state was at the front edge of the transition and produces a linear relationship of SET delay from collected charge. The OFF lines represent the change of a state with various loads on the node. In the DC-DFF case, the load on each NAND2 is approximately three logic gates (3x). At a collected charge of 10 fC, the ON SET pulse width is 142 ps, and the OFF with 3x load SET pulse width is 84 ps. Therefore, legitimate state changes are potentially blocked when the ON node is struck. The maximum pulse width for the ON case can only occur when the ion strike is exactly at the time when the state is about to change.

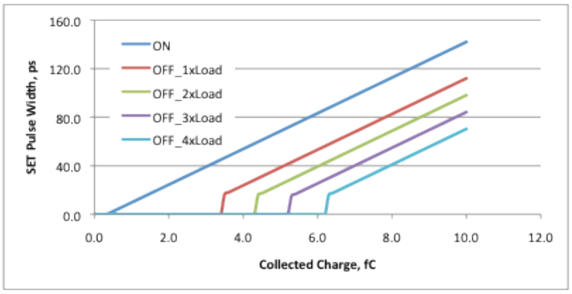


Figure 57. Maximum SET pulse widths versus collected charge for various NAND2 transistor conditions

CHAPTER VIII

SPICE CIRCUIT ANALYSIS FOR LOGICAL AND TIMING SIMULATIONS

This chapter discusses the generation and propagation of SETs within a complex digital IC. A background discussion motivates the use charge collection at the cell level to calculate soft errors of the IC. Because a transient must propagate to a storage element (e.g., a flip-flop or register) for visibility as an error, the logic depth between registers must be considered. The duration of the transient (i.e., the SET pulse-width) affects the probability of latching the transient. Instead of using a fixed SET pulse-width, the analysis is based upon the distribution of pulse widths as predicted from charge collection from MRED.

SET Pulse-Width Characterization for Radiation-Induced Faults

This section discusses in detail how an SET is generated at the cell level and some of the factors that contribute to SET pulse-width variability. The tools used to generate the SET are also described. Next, details regarding the propagation up the hierarchy of an IC [56] are described, as well as how the SET is further shaped as it propagates through combinational logic cells. The discussion also includes the role of timing vulnerability in the propagation of an SET and the descriptions of the tools used.

Background

Simulation results show that SET pulse-width variability is due to various factors including cell layout and the input state. In [40], Narasimham et al. use an on-chip asynchronous circuit approach to capture and analyze SET pulses generated from an inverter chain. The on-chip detection circuitry measures the width of each SET by determining the number of latches affected by the SET. SET pulse-width distributions and the SET cross-section of the individual inverters were evaluated. Simulations show that the distance to the body contact affects the SET pulse-width (Figure 58).

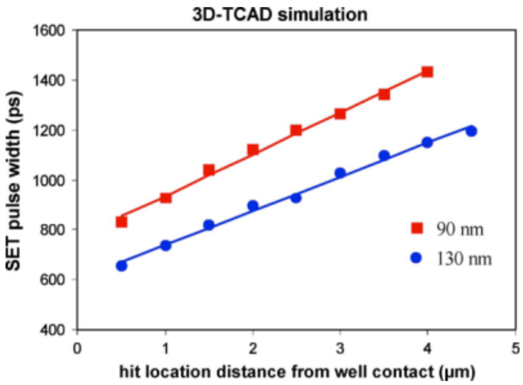


Figure 58. Variation of SET pulse-width relative to strike location distance to well contact[40]

Several research efforts [42, 57, 58] indicate that well contacts, specifically for the N-well, can significantly affect single event response. In [59], the minimum-size inverter in the IBM 90-nm cell library was implemented in TCAD, and a 40 MeV-cm^2/mg ion strike on the center of the PMOSFET drain is simulated for n-well contact areas of 0.2 μm^2 (same area for p-well) and 4 μm^2 (2 μm^2 for p-well). Figure 59 shows an example of the change in the full-width, half-maximum V_{dd} pulse-width from 1.7 ns to 620 ps when the n-well contact size is increased. (Note that the location and size of the well contact is not a significant factor for lightly ionizing particle charge collection. If this research is extended beyond lightly ionizing particles, then this factor will need to be included in the new model.)

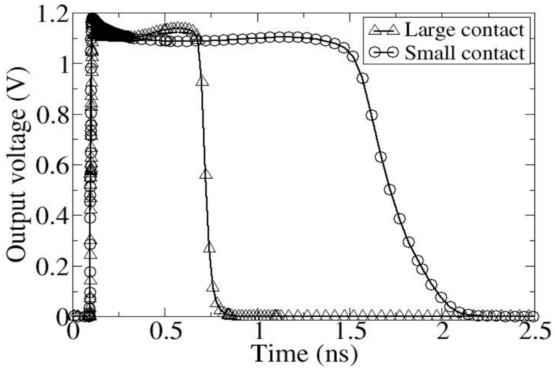


Figure 59. PMOSFET drain ion strike voltage pulses for 0.2 μm^2 and 4 μm^2 n-well contacts.[60]

The input state of a cell can determine which devices will collect charge, or whether the charge collected will cause the output to toggle. The input state can also factor into the resistance of the restoring current to terminate the effects of the single event, which was confirmed in Chapter VI. In [59], the authors show the effect of input states on SET pulse-width by using TCAD to model LET 40 MeV-cm^2/mg ion events in a NAND gate. The SETs obtained for various input states are shown in Figure 60. This factor is taken into account in the multi-scale simulation approach discussed in this dissertation. Figure 43 shows model results from MRED2SPICE that demonstrate the same effect for a NOR2x1 gate.

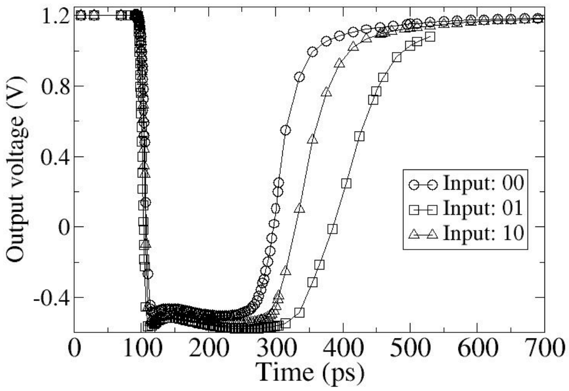


Figure 60. Effect of input state on single event response of NAND gate[60].

SPICE SET Pulse-Width Characterization

SETs become identical to a typical digital transitioning signal, i.e., logic 0 to logic 1 and vice versa after traversing a certain number of stages. As an SET propagates through logic cells, its shape will be altered; Dodd et al. [51] show this effect for particles with fairly low LETs. Figure 61 shows how pulse shaping changes the SET into a typical square voltage signal after propagating through a few logic cells.

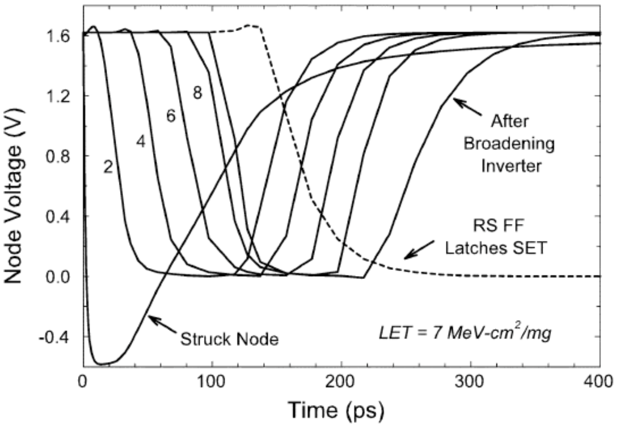


Figure 61. SET propagation in 10-inverter delay chains [51]

SPICE and the appropriate SET propagation techniques from Chapter V were used to develop a simulation solution to quantify an equivalent rail-to-rail voltage (V_{r-r}). The simulation results show the minimum number of follow-on cells beyond the struck cell that continues to shape the SET, and eventually forms a digital logic 1 to logic 0 signal, or vice versa. Logic depth is classified in a digital circuit as the maximum number of basic combinational gates, e.g. inverter, NAND gate, or NOR gate, that a signal is required to travel from source memory element to a destination memory element. The logic depth of combinational cells between storage elements can affect SET propagation. This is the key element necessary to couple MRED2SPICE to an IC modeling tool for SER prediction of a complex digital IC.

Logic Depth Consideration

The most aggressive logic depth for practical implementations is less than 10 fanout-4 (F04) gates per cycle to maintain sufficient on-chip performance [61]. The IBM PowerPC with integrated Sony cells falls into this range [44, 62-64]. The first XScale® ARM processor had a worse-case logic depth of approximately 27 gates across the whole chip, and the ALU was identified as the most problematic component in this processor due to the series shifter [65-67]. The ALU's critical path is the adder, which is used for most of the mathematical operations and is duplicated throughout the design. The adder needs to complete an operation within a clock cycle to receive inputs from the register file (RF) and the address to the cache. The only pertinent information is the logic depth for completing the cycle. The cycle consists of: (1) reading the RF at the active edge of the clock (less than a phase of the clock), (2) multiplexing (less than a phase of the clock), (3) performing an ALU operation (about a phase), and (4) multiplexing to get the address out to the cache (less than a phase) to finish out the clock cycle. Therefore, the critical path involves the ALU operations, which typically is designed with a logic depth of ~ 8 [64, 65]. Based on these results, a logic depth of 10 is the upper bound that will be used for this analysis.

Logic Depth for SPICE Simulation

The methodology of the MRED2SPICE tool flow and the associated Python scripts were extended to determine the impact of logic depth on the SET. Three logic depths were considered to determine if the SET *squares up* within a typical design: 3 (minimum simulation solution), 7, and 10 (upper bound). The SETs evaluated are from the original MRED simulation runs specified for the 90-nm IBM inverter, NAND gate, and NOR gate combinational cells discussed previously in this dissertation. The block diagram illustrating this process is seen in Figure 62 and the target design used for the logic depth SET analysis is in Figure 63.

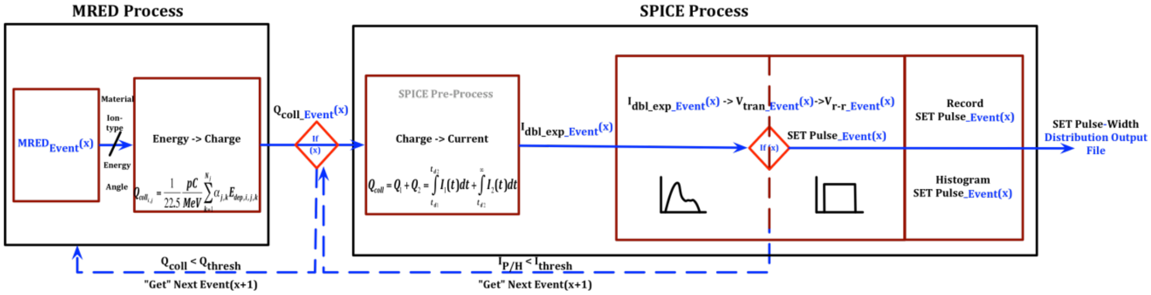

MRED Process
$\mathrm{MRED}_{\mathrm{Event}}(x)$
Material
Ion-type
Energy
Angle
Energy -> Charge
$Q_{coll_{i,j}} = \frac{1}{22.5}\frac{pC}{MeV}\sum_{k=1}^{N_j}\alpha_{j,k}E_{dep,i,j,k}$
$Q_{\mathrm{coll\_Event}}(x)$
If (x)
SPICE Process
SPICE Pre-Process
Charge -> Current
$Q_{coll} = Q_1 + Q_2 = \int_{t_{d1}}^{t_{d2}} I_1(t)dt + \int_{t_{d2}}^{\infty} I_2(t)dt$
$I_{\mathrm{dbl\_exp\_Event}}(x)$
$I_{\mathrm{dbl\_exp\_Event}}(x)$ -> $V_{\mathrm{tran\_Event}}(x)$->$V_{\mathrm{r\text{-}r\_Event}}(x)$
If (x)
SET Pulse_Event(x)
Record
SET Pulse_Event(x)
Histogram
SET Pulse_Event(x)
SET Pulse-Width
Distribution Output
File
$Q_{coll} < Q_{thresh}$
"Get" Next Event(x+1)
$I_{P/H} < I_{thresh}$
"Get" Next Event(x+1)

Figure 62. MRED to LOGIC depth (MRED2LOGIC) block diagram

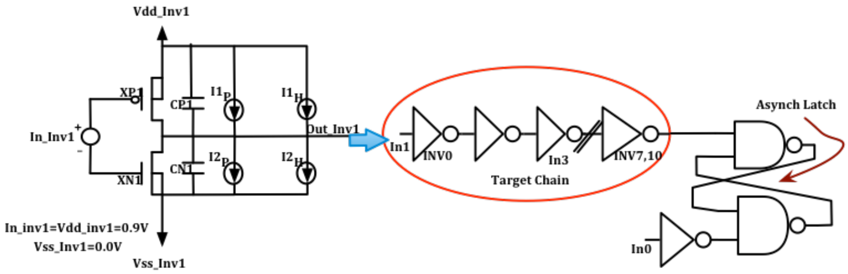


Figure 63. Target design for logic depth SET pulse-width analysis. Logic depths of 3, 7, and 10 were used.

<u>Procedure for MRED2LOGIC within the Multi-Scale Simulation</u>

MRED can be linked to the IC-level simulation via the circuit-level simulation to form the MRED2LOGIC approach. The procedural steps for MRED2LOGIC begin by using the MRED output data array file listing each MRED event and the charge collection, Q_{coll}, for each nested sensitive volume as an input to SPICE. This output data array file is parsed for Q_{coll} to extract the relevant cases where Q_{coll} is greater than Q_{thresh}, and the results are stored while maintaining the MRED event number. Using Q_{coll}, the calculations are completed for I_{Prompt} and I_{Hold} and their corresponding SET pulse-widths. The script then creates a circuit netlist for the 3, 7, or, 10 logic depth (Figure 63) and populates the dual double-exponential current sources with these values. SPICE is run on the circuit netlist. The SPICE simulation strikes a random node within the chosen logic depth and evaluates the results for a full-width half-maximum crossing of the resulting output rail-to-rail voltage ($V_{r\text{-}r}$). The results are then recorded into two different files. One records the MRED event number, the node that was struck, and the resulting SET pulse-width. The second output file is the histogram of the resulting SET pulse-widths binned into 10-ps increments. This increment was determined to be the optimal bin width for no loss of data and minimal error in SET pulse-width during development. Using the two output files forms gives the user the flexibility to make a decision on the preferred IC modeling tool. The research in this dissertation uses ModelSim®, so the pulse-width distribution is formatted for this tool. The script is executed in parallel for both the PMOSFETs and the NMOSFETs used in the combinational cell. A flowchart for the MRED2LOGIC Python script is shown in Figure 64, and an example of the Python script is found in APPENDIX E.

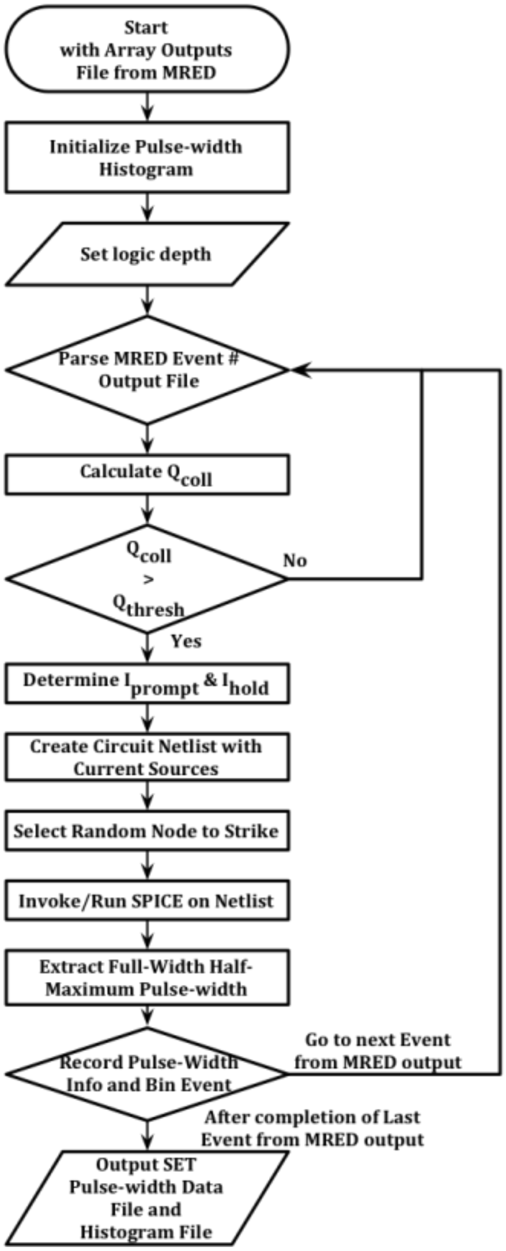
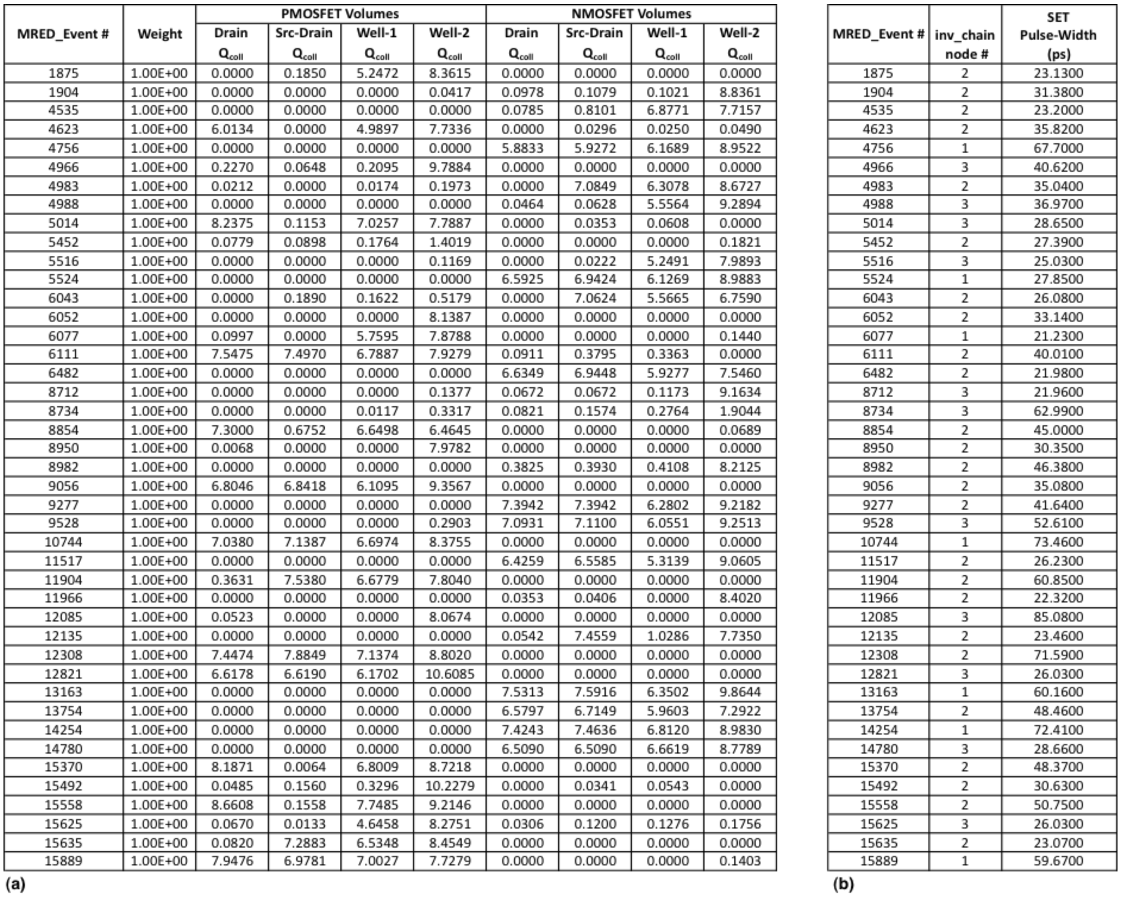


Figure 64. MRED2LOGIC process flowchart

An example of the SET pulse-width output file for a design containing the IBM 90-nm inverter design and a logic depth of 3 is seen in Figure 65.



MRED_Event #	Weight	PMOSFET Volumes				NMOSFET Volumes			
		Drain	Src-Drain	Well-1	Well-2	Drain	Src-Drain	Well-1	Well-2
		Q_{coll}	Q_{coll}	Q_{coll}	Q_{coll}	Q_{coll}	Q_{coll}	Q_{coll}	Q_{coll}
1875	1.00E+00	0.0000	0.1850	5.2472	8.3615	0.0000	0.0000	0.0000	0.0000
1904	1.00E+00	0.0000	0.0000	0.0000	0.0417	0.0978	0.1079	0.1021	8.8361
4535	1.00E+00	0.0000	0.0000	0.0000	0.0000	0.0785	0.8101	6.8771	7.7157
4623	1.00E+00	6.0134	0.0000	4.9897	7.7336	0.0000	0.0296	0.0250	0.0490
4756	1.00E+00	0.0000	0.0000	0.0000	0.0000	5.8833	5.9272	6.1689	8.9522
4966	1.00E+00	0.2270	0.0648	0.2095	9.7884	0.0000	0.0000	0.0000	0.0000
4983	1.00E+00	0.0212	0.0000	0.0174	0.1973	0.0000	7.0849	6.3078	8.6727
4988	1.00E+00	0.0000	0.0000	0.0000	0.0000	0.0464	0.0628	5.5564	9.2894
5014	1.00E+00	8.2375	0.1153	7.0257	7.7887	0.0000	0.0353	0.0608	0.0000
5452	1.00E+00	0.0779	0.0898	0.1764	1.4019	0.0000	0.0000	0.0000	0.1821
5516	1.00E+00	0.0000	0.0000	0.0000	0.1169	0.0000	0.0222	5.2491	7.9893
5524	1.00E+00	0.0000	0.0000	0.0000	0.0000	6.5925	6.9424	6.1269	8.9883
6043	1.00E+00	0.0000	0.1890	0.1622	0.5179	0.0000	7.0624	5.5665	6.7590
6052	1.00E+00	0.0000	0.0000	0.0000	8.1387	0.0000	0.0000	0.0000	0.0000
6077	1.00E+00	0.0997	0.0000	5.7595	7.8788	0.0000	0.0000	0.0000	0.1440
6111	1.00E+00	7.5475	7.4970	6.7887	7.9279	0.0911	0.3795	0.3363	0.0000
6482	1.00E+00	0.0000	0.0000	0.0000	0.0000	6.6349	6.9448	5.9277	7.5460
8712	1.00E+00	0.0000	0.0000	0.0000	0.1377	0.0672	0.0672	0.1173	9.1634
8734	1.00E+00	0.0000	0.0000	0.0117	0.3317	0.0821	0.1574	0.2764	1.9044
8854	1.00E+00	7.3000	0.6752	6.6498	6.4645	0.0000	0.0000	0.0000	0.0689
8950	1.00E+00	0.0068	0.0000	0.0000	7.9782	0.0000	0.0000	0.0000	0.0000
8982	1.00E+00	0.0000	0.0000	0.0000	0.0000	0.3825	0.3930	0.4108	8.2125
9056	1.00E+00	6.8046	6.8418	6.1095	9.3567	0.0000	0.0000	0.0000	0.0000
9277	1.00E+00	0.0000	0.0000	0.0000	0.0000	7.3942	7.3942	6.2802	9.2182
9528	1.00E+00	0.0000	0.0000	0.0000	0.2903	7.0931	7.1100	6.0551	9.2513
10744	1.00E+00	7.0380	7.1387	6.6974	8.3755	0.0000	0.0000	0.0000	0.0000
11517	1.00E+00	0.0000	0.0000	0.0000	0.0000	6.4259	6.5585	5.3139	9.0605
11904	1.00E+00	0.3631	7.5380	6.6779	7.8040	0.0000	0.0000	0.0000	0.0000
11966	1.00E+00	0.0000	0.0000	0.0000	0.0000	0.0353	0.0406	0.0000	8.4020
12085	1.00E+00	0.0523	0.0000	0.0000	8.0674	0.0000	0.0000	0.0000	0.0000
12135	1.00E+00	0.0000	0.0000	0.0000	0.0000	0.0542	7.4559	1.0286	7.7350
12308	1.00E+00	7.4474	7.8849	7.1374	8.8020	0.0000	0.0000	0.0000	0.0000
12821	1.00E+00	6.6178	6.6190	6.1702	10.6085	0.0000	0.0000	0.0000	0.0000
13163	1.00E+00	0.0000	0.0000	0.0000	0.0000	7.5313	7.5916	6.3502	9.8644
13754	1.00E+00	0.0000	0.0000	0.0000	0.0000	6.5797	6.7149	5.9603	7.2922
14254	1.00E+00	0.0000	0.0000	0.0000	0.0000	7.4243	7.4636	6.8120	8.9830
14780	1.00E+00	0.0000	0.0000	0.0000	0.0000	6.5090	6.5090	6.6619	8.7789
15370	1.00E+00	8.1871	0.0064	6.8009	8.7218	0.0000	0.0000	0.0000	0.0000
15492	1.00E+00	0.0485	0.1560	0.3296	10.2279	0.0000	0.0341	0.0543	0.0000
15558	1.00E+00	8.6608	0.1558	7.7485	9.2146	0.0000	0.0000	0.0000	0.0000
15625	1.00E+00	0.0670	0.0133	4.6458	8.2751	0.0306	0.1200	0.1276	0.1756
15635	1.00E+00	0.0820	7.2883	6.5348	8.4549	0.0000	0.0000	0.0000	0.0000
15889	1.00E+00	7.9476	6.9781	7.0027	7.7279	0.0000	0.0000	0.0000	0.1403

(a)

MRED_Event #	inv_chain node #	SET Pulse-Width (ps)
1875	2	23.1300
1904	2	31.3800
4535	2	23.2000
4623	2	35.8200
4756	1	67.7000
4966	3	40.6200
4983	2	35.0400
4988	3	36.9700
5014	3	28.6500
5452	2	27.3900
5516	3	25.0300
5524	1	27.8500
6043	2	26.0800
6052	2	33.1400
6077	1	21.2300
6111	2	40.0100
6482	2	21.9800
8712	3	21.9600
8734	3	62.9900
8854	2	45.0000
8950	2	30.3500
8982	2	46.3800
9056	2	35.0800
9277	2	41.6400
9528	3	52.6100
10744	1	73.4600
11517	2	26.2300
11904	2	60.8500
11966	2	22.3200
12085	3	85.0800
12135	2	23.4600
12308	2	71.5900
12821	3	26.0300
13163	1	60.1600
13754	2	48.4600
14254	1	72.4100
14780	3	28.6600
15370	2	48.3700
15492	2	30.6300
15558	2	50.7500
15625	3	26.0300
15635	2	23.0700
15889	1	59.6700

(b)

Figure 65. MRED2LOGIC SET pulse-width output file samples for inverter cell with logic depth 3, (a) Original MRED input file for Q_{coll}, (b) MRED2LOGIC SET pulse-width output file results with corresponding MRED simulation event number after being processed through the multi-scale simulation with no loss of information.

MRED2LOGIC Histogram Results

The three IBM 90-nm combinational cells were analyzed for SETs at an LET of 2.1 MeV-cm^2/mg for three logic depths (3, 7, and 10) using the same radiation environment as Cannon et al. [16] and Atkinson et al. [60]. Examples of the resulting histogram files are shown for the INVx1 (Figure 66), the NAND2x1 (Figure 67), and the NOR2x1 (Figure 68).

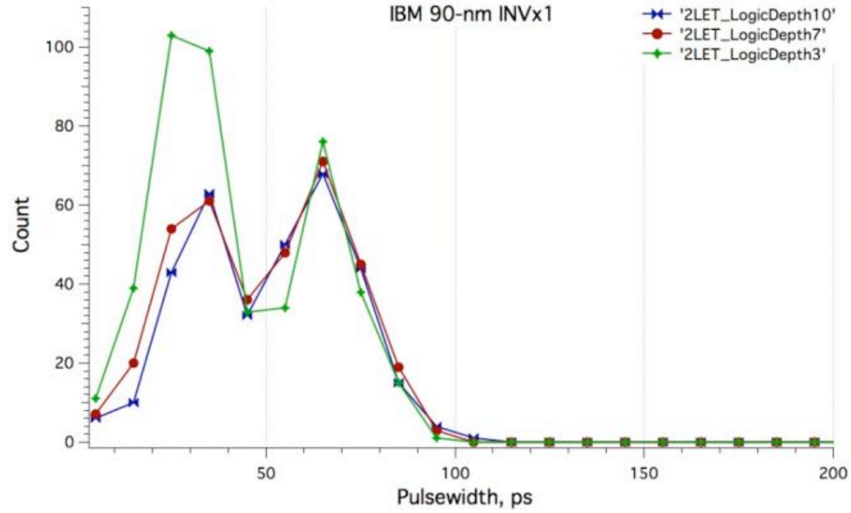


Figure 66. Bin counts of SET pulse-widths for IBM 90-nm INVx1 for three different logic depths and particles of 2.1 MeV-cm^2/mg

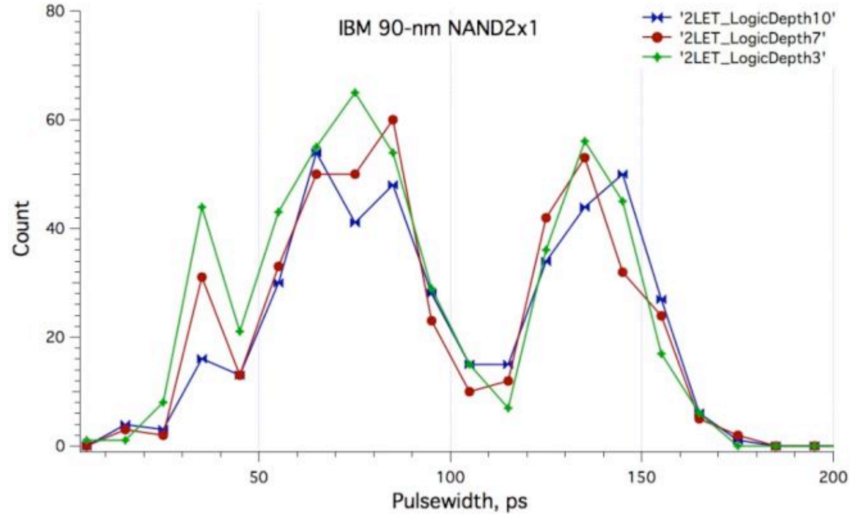


Figure 67. Bin counts of SET pulse-widths for IBM 90-nm NAND2x1 for three different logic depths and particles of 2.1 MeV-cm^2/mg

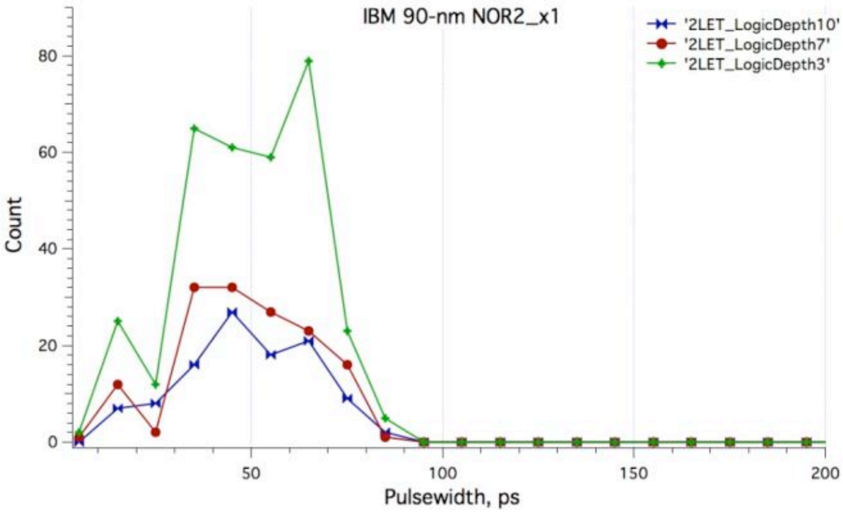


Figure 68. Bin counts of SET pulse-widths for IBM 90-nm NOR2x1 for three different logic depths and particles of 2.1 MeV-cm^2/mg

The simulations included all transients that are generated from the NMOSFETS and PMOSFETS; the distribution shows the frequency of durations that have been generated from MRED events. However, the figures indicate that a simple, fixed pulse-width does not capture the true behavior from SETs generated within the circuit. A logic depth of 3 produces the largest number of SETs. Shorter pulses (e.g., less than 50 ps) are not observed as frequently for the INVx1 and the NOR2x1 because the pulses are shorter than the rise time of the circuit [47]. The remainder of the dissertation will use a logic depth of 3.

Summary

This chapter describes the development of pulse-width distributions for three library cells. Charge collection from MRED events was translated into SETs for circuits with logic depths of 3, 7, and 10. The framework includes traceability to actual particle strikes. The distribution enables the analysis of more complex designs, such as an ALU, by using the individual cells as building blocks. This topic is described in the next chapter.

CHAPTER IX

IC LOGIC SIMULATION FOR SOFT ERROR PREDICTION

This chapter discusses the use of simulation techniques to predict soft errors that propagate up the hierarchy from a single combinational cell to the full complex digital circuit. This analysis can be performed at design time. It also gives a brief discussion of the testbench used to test an example IC (an ALU) and inputs required to verify this design and ultimately produce a soft error prediction.

Basic Testing Approach

Once an equivalent rail-to-rail voltage pulse width, $V_{r\text{-}r}$, has been determined, it can be used in conjunction with an IC simulation tool, such as ModelSim®. The ModelSim® tool is a unified debug environment for full simulation of an IC that has been modeled with Verilog, VHDL, or SystemC [68]. The benefit of integrating ModelSim® with MRED and SPICE is the ability to have a full circuit simulation that enables the demonstration of SET capture via an operational circuit such as the ALU (Figure 69). $V_{r\text{-}r}$ is identical to a typical digital signal – logic 0 to logic 1 and vice versa. Compatibility with a circuit simulator requires the translation of SETs into logic 0 or logic 1.

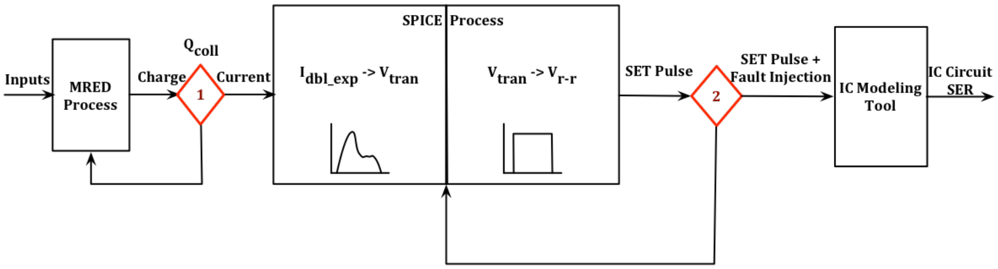


Figure 69. SET multi-scale simulation MRED to SPICE to ModelSim® for complex digital ICs

Functional verification is performed on hardware designs during the design phase of the digital circuit development process to check its behavior. The verification environment is composed of a testbench surrounding the Device Under Test (DUT) (Figure 70) [48, 69-71]. Functional verification is done by comparing simulated design responses from incoming data (stimuli) against expected values.

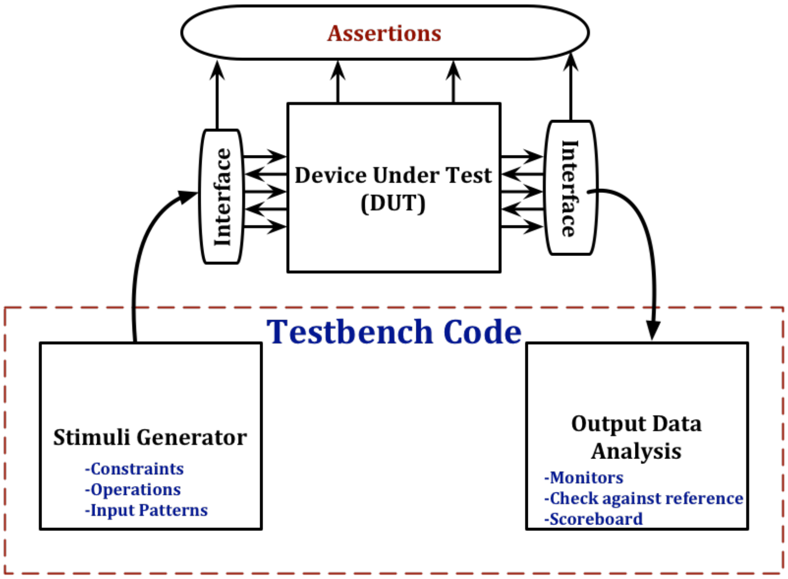


Figure 70. Basic block diagram for a testbench with a design under test (DUT)

The testbench may include a behavioral model of the design and test vectors. These test vectors can be provided as a file of inputs and expected outputs (I/O) if pre-calculated responses are available from an external reference model. The testbench includes additional constructs, such as stimuli generation, output analysis, or reporting. DUT responses (or actions) to stimuli are compared with the expected results to validate the behavior. In terms of languages, modern verification techniques, such as directed and constrained-random verification, coverage driven verification, and assertion based verification [72], make use of hardware description languages like VHDL, Verilog, or SystemVerilog [73]. These languages enable development of more efficient functional verification environments and facilitate the reuse of testbench components.

During the verification process, the design must be checked to have correct functionality in normal operation. It may be a challenge for the designer to simulate complex designs correctly and to check that the verification process covers all of the functional features. The challenge is greater when time is short for creating the verification environment, defining the test cases of interest, and simulating them. Assertion statements can be used within the verification process to help a designer know the current status of the testbench, as well as the states of the I/O. Execution of a test can terminate early if a fault should occur. This methodology was used for the multi-scale simulation development.

Multi-scale simulation for Soft Error Rate Prediction

Testbench Framework

The previous chapters describe the basic testing method used for the IBM 90-nm combinational cells that have been characterized for SET pulse-widths.

A fault injection library [17], which was developed at the Vanderbilt University Institute for Space and Defense Electronics, was used with the SET pulse-width distributions for the INVx1, NAND2x1, and NOR2x1 to enable soft error rate prediction for those components within the ALU. The library simulates single event upsets and single event transients in an IC simulation tool based upon a Register-Transfer Level (RTL) description. RTL is a level of abstraction used in describing the operation of a synchronous digital circuit. In an RTL representation, a circuit's behavior is defined in terms of the flow of signals, or the transfer of data between hardware registers, and the logical operations performed on those signals. The implementation is intended to be independent of the user's choice of simulators and hardware description language. The fault injection library has been tested with Icarus Verilog, Cadence NC-Verilog, Synopsys VCS, and Mentor ModelSim®. The version used for the multi-scale simulation only contained the functionality related to event generation. Using an ALU as the DUT, the testbench defined the effects that constituted an error in the system as well as the appropriate checking and logging for fault generation and error detection. A block diagram of this advanced technique with fault injection is shown in Figure 71.

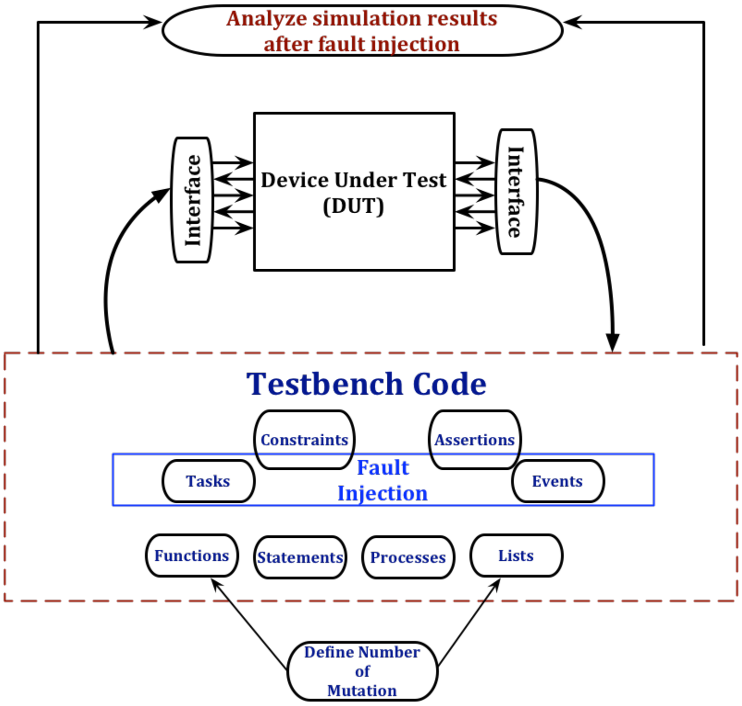


Figure 71. Block diagram for injecting faults into the testbench

Soft Error Rate Simulation Process

The fault injection library implements a randomization of the SETs. The testbench performs several mutations to strike a different combinational cell with each invocation. The module provides a *$pseudoRandom(MAX)* function that returns an integer from 0 to *MAX*. Unless, *pseudoRandomSeed (SEED)* has previously been given a random seed as a parameter, the *$pseudoRandom* number stream is different for each process. Next, a random SET pulse-width is generated from the SET pulse-width distribution file produced from the MRED2LOGIC process in CHAPTER VIII. The fault injection library required that the input SET distribution file be converted to a raw count divided by the irradiation fluence multiplied by the bin-width (i.e., fluence × bin-width). This conversion makes the cross-section calculation for soft error rate prediction traceable from the originating MRED simulations, since the distributions are now in (cm²/ps) vs. (bin-width in ps). An example of the originating SET pulse-width distribution and the converted distribution is shown in Figure 72.

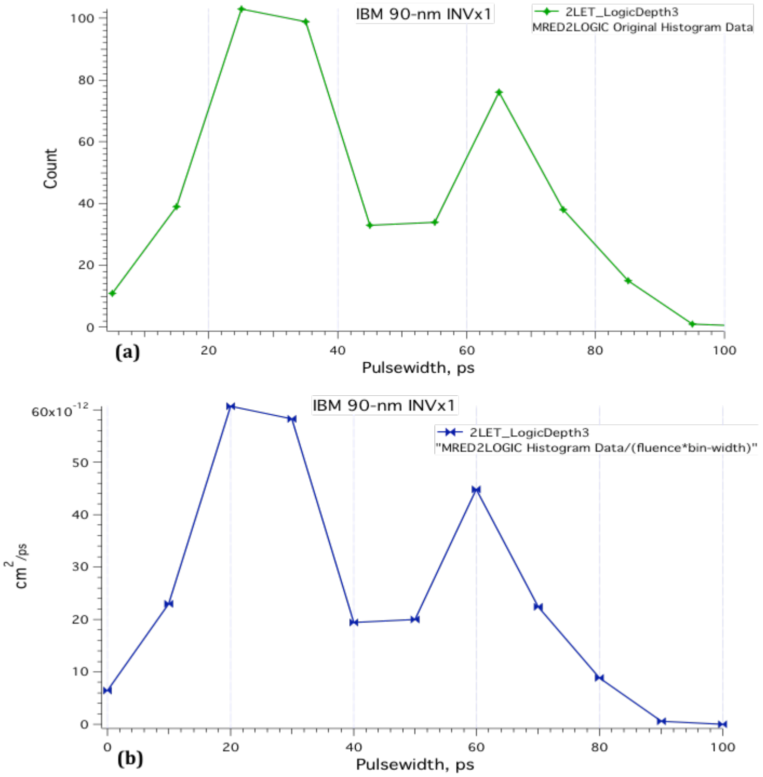


Figure 72. MRED2LOGIC SET pulse-width distribution for 90-nm INVx1 for LET = 2.1 MeV-cm²/mg (a) Original histogram data distribution (b) Original histogram data converted as required for fault injection.

A random SET is selected based upon the distribution. Next, the netlist (e.g., an ALU for this dissertation) is parsed to identify the usage of each library cell (e.g., INVx1, NAND2x1, and NOR2x1 for this dissertation). At the end of the simulation, the calculated results include the integral cross-sections for: (1) each of the individual cells within the ALU and (2) the total instantiated cell count for the ALU.

The testbench chooses a random cell in the ALU for fault injection and strikes it with a random SET pulse-width within the distribution associated with the cell. Then, the testbench runs all test vectors for all functions specified by the ALU. In this case, the ALU has a datapath width of 8 bits and uses three control bits; the total number of test vectors is 131,075. The testbench monitors the outputs for errors. If an error occurs, then the testbench: (1) stops the simulation, (2) reports all the erroneous outputs, (3) reports the expected outputs, (4) reports the SET pulse-width, and (5) reports the time the error occurred. If no error occurs, then another mutation is invoked to continue the simulation. A flowchart for this process is shown in Figure 73. Once all the mutations are complete, then the probability of a soft error for the DUT can be calculated by dividing the number of errors recorded by the number of simulation mutations (i.e., the number of SETs generated). An example of the testbench code is found in APPENDIX F.

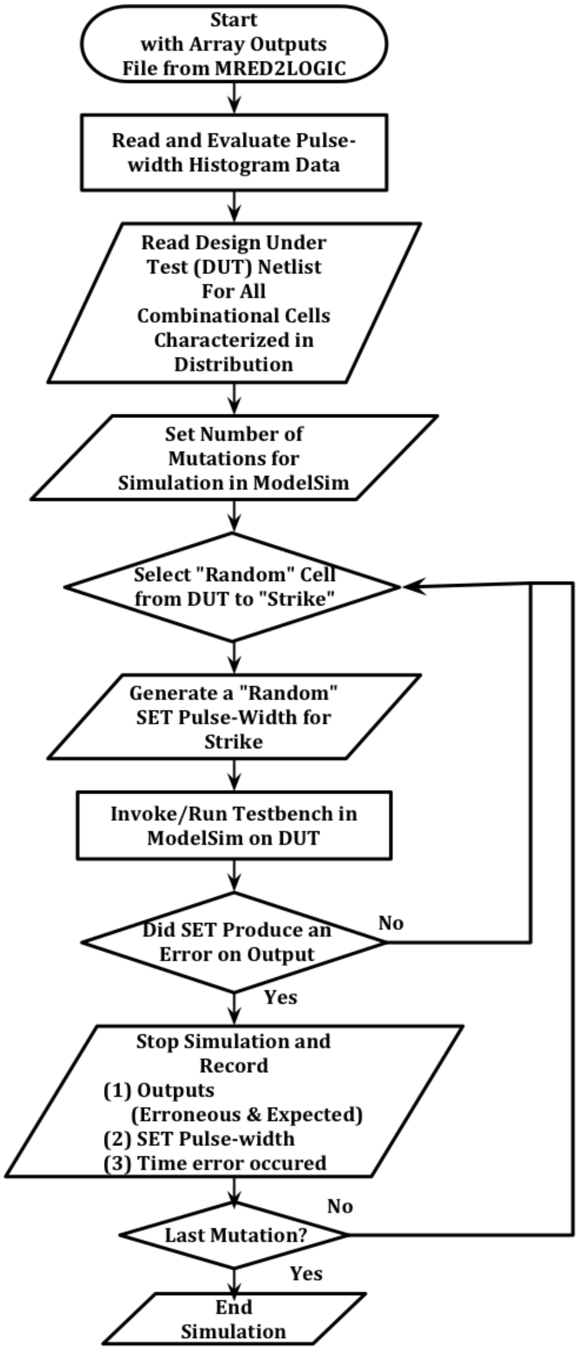
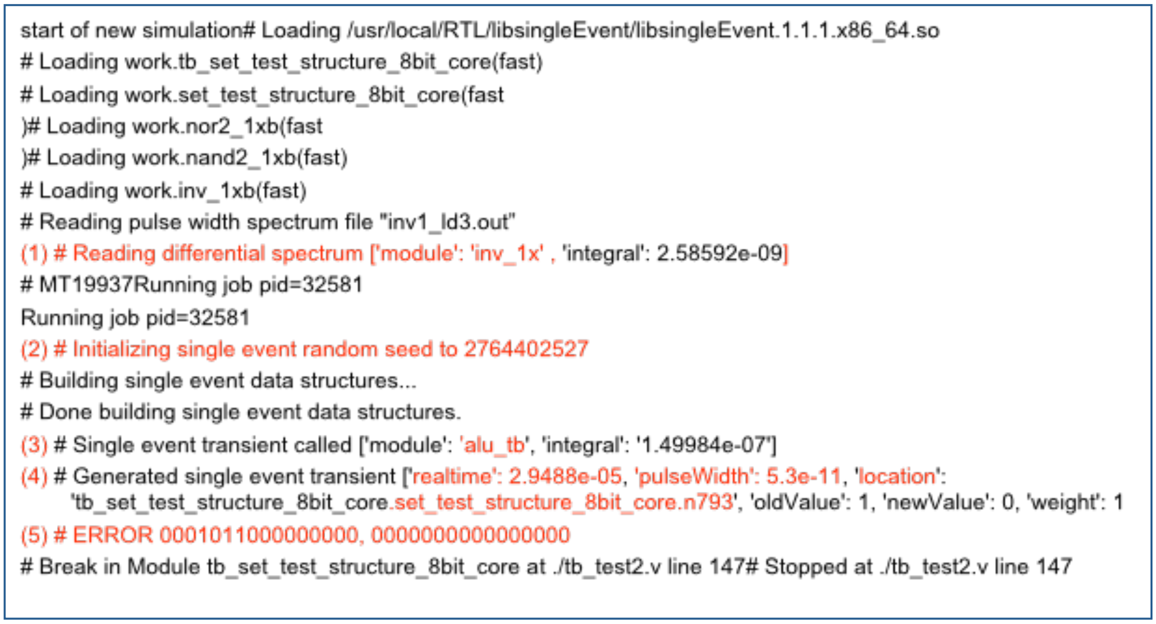

Start
with Array Outputs
File from MRED2LOGIC
Read and Evaluate Pulse-width Histogram Data
Read Design Under Test (DUT) Netlist For All Combinational Cells Characterized in Distribution
Set Number of Mutations for Simulation in ModelSim
Select "Random" Cell from DUT to "Strike"
Generate a "Random" SET Pulse-Width for Strike
Invoke/Run Testbench in ModelSim on DUT
Did SET Produce an Error on Output
No
Yes
Stop Simulation and Record
(1) Outputs
(Erroneous & Expected)
(2) SET Pulse-width
(3) Time error occured
Last Mutation?
No
Yes
End Simulation

Figure 73. Flowchart for ModelSim® simulation to determine soft error rate

Python scripting was used to produce the queue of simulation mutations. The transcript for each simulation was stored into an output file for analysis when all simulations were complete. An example of the transcript window shows the steps from the flowchart as they are executed via the testbench (Figure 74).



```
start of new simulation# Loading /usr/local/RTL/libsingleEvent/libsingleEvent.1.1.1.x86_64.so
# Loading work.tb_set_test_structure_8bit_core(fast)
# Loading work.set_test_structure_8bit_core(fast
)# Loading work.nor2_1xb(fast
)# Loading work.nand2_1xb(fast)
# Loading work.inv_1xb(fast)
# Reading pulse width spectrum file "inv1_ld3.out"
(1) # Reading differential spectrum ['module': 'inv_1x' , 'integral': 2.58592e-09]
# MT19937Running job pid=32581
Running job pid=32581
(2) # Initializing single event random seed to 2764402527
# Building single event data structures...
# Done building single event data structures.
(3) # Single event transient called ['module': 'alu_tb', 'integral': '1.49984e-07']
(4) # Generated single event transient ['realtime': 2.9488e-05, 'pulseWidth': 5.3e-11, 'location':
      'tb_set_test_structure_8bit_core.set_test_structure_8bit_core.n793', 'oldValue': 1, 'newValue': 0, 'weight': 1
(5) # ERROR 0001011000000000, 0000000000000000
# Break in Module tb_set_test_structure_8bit_core at ./tb_test2.v line 147# Stopped at ./tb_test2.v line 147
```

Figure 74. Sample of the ModelSim® simulation transcript. (1) INVx1 SET pulse-width distribution file is read, (2) Random seed for cell to strike, (3) ALU testbench stimuli and monitor are invoked, (4) Random SET pulse-width strike length, time of strike, and specific cell (5) Erroneous outputs and expected outputs at time of error.

Soft Error Rate Predictions for Individual Library Cells

Multiple ModelSim® simulations were executed using the flow identified in the previous section. The logical-masking error rate was examined by using errors that lasted the full clock width. This method enabled comparison to previous work [59, 60] to verify the accuracy. However, the previous method used by Black et al. [59] used multiple tools: (1) CRÈME96 [74] (Cosmic Ray Effects on Micro-Electronics Code) for calculations of both incident and shielded cosmic heavy ion fluxes, (2) Technology-Computer Aided Design (TCAD) for process and design simulation of the PMOSFETs and NMOSFETs, (3) SPICE for timing and SET pulse-width evaluation, and finally (4) ModelSim® for circuit level (ALU) error probability prediction. This approach required separate tools and analyses, which required over a year for the evaluation and the final results, mainly due to TCAD simulations that remained incomplete when the report was submitted.

The results for the multi-scale simulation for the IBM 90-nm INVx1, NAND2x1, and NOR2x1 agreed with the simulation results from Black et al. [59] and Atkinson et al. [60]. Table 14 reports the results, including: (1) the simulation method, (2) the combinational cell simulated with the number of instantiations for the ALU design, and lastly (3) the number of mutations required to simulate for the reported results.

Table 14. Simulation results for percent susceptible to SETs for logical-masking.

Simulation Method	**INVx1** **(cell count = 58)** **Susceptibility**	**NAND2x1** **(cell count =114)** **Susceptibility**	**NOR2x1** **(cell count = 51)** **Susceptibility**	**Simulation Mutations**
Black et al. [59]	9 %	14.3%	8.3%	Cell Count x 131075
MRED2LOGIC	10%	14.3%	8.5%	3000 per cell

The multi-scale simulation required 12,000 total mutations to simulate (3000 for each combinational cell, 3000 for all three cells evaluated together, or *Multi*), while Black et al. took 29,229,725 [(58 + 114 + 51) x 131075] mutations in ModelSim®. The MRED2LOGIC multi-scale simulation with fault injection achieved comparable logical-masking results in a greatly reduced amount of simulation and computing time.

However, the goal of the multi-scale simulation approach is to provide a means to achieve a soft error analysis for a complex digital IC not limited to logical-masking only. Therefore, additional simulations were run at the circuit's operating frequency with SETs from the distribution applied randomly during the clock period. This takes into account timing-masking because the pulse-widths vary in length and may or may not appear during the window of vulnerability. This approach mimics IC radiation testing. The logical-masking percentages for susceptibility are the upper bound for the SER for each combinational cell evaluated during simulation.

For the ALU, the multi-scale simulation SER cross-sections per cell (i.e., INVx1, NAND2x1, and NOR2x1) were calculated via fault injection as the testbench computed the integral cross-sections per cell (e.g., the total contribution to the cross section for all INVx1 cells in the ALU design). The verification process parsed the ALU design for the number of instantiations of each cell (i.e., cell count). It then: (1) calculates the integral cross-section per cell as well as for the total contribution of all instances of that cell for the ALU, (2) applies a randomly selected SET during the cycle time, and then (3) determines if a generated fault results in an output error. Given this process, the SER is then a calculation of the number of errors observed divided by the number of faults generated (simulation mutations) multiplied by the total contribution integral cross-section (6).

$$SER = \left(\frac{Total \;\; Errors \;\; Observed}{Total \;\; Faults \;\; Generated} \right) x \int_{i=1}^{n} (Cell \;\; Cross - Section)_i \tag{6}$$

These results do not distinguish between logical-masked or timing-masked errors, similar to the results obtained during a radiation test. The data were computed in 3 days, a fraction of the simulation and computing time (approximately 3 months) reported by Black et al. The results for the INVx1, NAND2x1, and NOR2x1, and all three cells combined (i.e., Multi) are presented in Table 15. A full analysis of the ALU SER would require that all cells from the library are characterized for the full space environment.

Table 15. SER Error cross-section => (Errors Observed / Faults Generated) x Integral Cross-Section for LET = 2.1 MeV-cm^2/mg

Cell	Errors Observed	Faults Generated	Integral Cross-Section (cm^2)	SER cross-section (cm^2)
INVx1	107	3000	1.50 x 10^{-7}	5.35 x 10^{-9}
NAND2x1	128	3000	3.37 x 10^{-7}	14.40 x 10^{-9}
NOR2x1	89	3000	9.91 x 10^{-8}	2.94 x 10^{-9}
Multi (All 3)	114	3000	5.87 x 10^{-7}	2.23 x 10^{-8}

The sum of the individual SER cross-sections for the ALU is 2.27 x 10^{-8} cm^2. Comparing that result to the Multi (All 3) cross-section, 2.23 x 10^{-8} cm^2 shows a difference of 3.99 x 10^{-10} cm^2, which is less than 1 percent error. The multi-scale simulation produces results for either: (1) an individual cell's contribution to SER cross-section, or (2) the overall SER cross-section for the full IC. To obtain the SER for an entire IC, the same procedure would need to be performed for all the library cells within the ALU design (see Table 16). This method provides a cross-layer solution to identify vulnerabilities using (1) fundamental device physics, (2) cell library design, and (3) transient capture. Previous methods were unable to connect transients within the circuit to specific ionizing events. Also, a single method for sensitive volumes is applied for each library cell to reduce the overhead of TCAD simulations.

Table 16. Methodology for SER Error cross-section of the entire ALU design. The table would need to be completed for all the cell types. The three cells from this dissertation are included within the table.

Library Cell Type	# Instances in Design	Individual Cross-Section (10^{-8}cm^2)	Integral Cross-Section (10^{-8} cm^2)	SE Probability cross-section (10^{-8} cm^2)	Running Total
add_1x1x	22				
add_2x2x	9				
add_6x6x	1				
addh_1x1x	2				
addh_3x3x	1				
and2_1x	17				
and2_3x	6				
and2_4x	4				
and3_1x	3				
ao21_1x	1				
ao21_3x	2				
aoai211_1x	7				
aoi21_1x	45				
aoi21_2x	11				
aoi21_3x	5				
aoi21_8x	1				
aoi21_b1_1x	6				
aoi21_b1_2x	1				
aoi21_b2_1x	8				
aoi21_b2_2x	1				
aoi22_1x	5				
aoi22_3x	1				

buf_10x	6				
buf_16x	2				
buf_1x	2				
buf_3x	3				
buf_8x	1				
delay1_4x	1				
inv_1x	58	0.30	15.0	0.54	0.54
inv_2x	35				
inv_30x	1				
inv_3x	17				
inv_4x	3				
inv_6x	1				
nand2_1x	114	0.30	33.7	1.44	1.98
nand2_2x	43				
nand2_3x	6				
nand2_4x	4				
nand2_5x	4				
nand2_6x	1				
nand2_8x	1				
nand2b0_1x	27				
nand2b0_2x	1				
nand2b0_4x	14				
nand3_1x	3				
nand3_2x	1				
nand3b0_1x	47				
nand3b0_2x	3				
nand4_1x	1				
nor2_1x	51	0.19	9.91	0.29	2.26
nor2_2x	1				
nor2_3x	6				
nor2b0_1x	5				
nor2b0_4x	1				
nor3_1x	1				
nor3b0_1x	1				
nor4_1x	1				
oa21_1x	2				
oai21_1x	50				
oai21_2x	2				
oai21_3x	3				
oai21_4x	8				
oai21_6x	2				
oai211_1x	1				
oai21b0b1_1x	4				
oai21b0b1_4x	1				

oai21b1_1x	21				
oai21b2_2x	1				
oai21b2_4x	2				
oai22_1x	36				
oai22_4x	3				
oai22b0b1_1x	1				
oai22b1_1x	14				
oaoi211_1x	1				
or2_1x	10				
or2_2x	1				
scandice_cpq_1x	15				
scandice_cpq_2x	8				
scandice_cpq_4x	12				
xnor2_1x	42				
xnor2_3x	1				
xnor3_1x	7				
xnor3_2x	2				
xor2_1x	14				
xor3_1x	1				

Summary

The addition of ModelSim® into the multi-scale simulation approach showed the link between MRED2SPICE and MRED2LOGIC for cell characterization. It takes the process from cell level characterization up a level of abstraction to IC circuit level analysis.

MRED2SPICE and MRED2LOGIC processes achieved a good match to existing experimental data. The SET pulse-width distribution was used to analyze an ALU. Within this design, the SER cross section was predicted for individual library cells. For different IC designs, fault injection can be performed without having to redo the analysis with MRED or SPICE if the full cell library has been characterized previously. The only change would be the IC design itself. Once a full library is characterized, then fault injection provides a full system SER prediction for the design under test.

CHAPTER X

CONCLUSIONS

The key result of this research is a multi-scale simulation approach for modeling SETs in ICs. It represents an end-to-end process from detailed simulation of energy deposition by each radiation event within a semiconductor to a one-to-one mapping of how these events produce a fault at the IC level. The process begins with energy deposition within a set of transistors, proceeds up the hierarchy to single event transient effects on a combinational cell, and then up one more level of abstraction to a single event error on a digital IC where the effect of the SET is observed and counted as an error. The process can be run in steps, as demonstrated throughout this dissertation, or executed end-to-end. The multi-scale approach allows for an ensemble of radiation events to be simulated so that the aggregated affect from a complex radiation environment can be computed.

This work demonstrates that using TCAD results to define the inputs, MRED coupled with SPICE can be used to compute the cross section for producing an SET for three circuits fabricated in a 90-nm bulk CMOS technology. The TCAD results obtained from a single transistor were used to define a multi-volume structure in order to make a first estimate of the charge collection efficiencies of these volumes. Then the efficiencies were refined by comparing the simulation result to experimental cross section results using ions with various LETs less than ~10 MeV-cm2/mg. Only one efficiency was changed for one of the volumes, all others remained identical to that estimated by TCAD. With this single small refinement, the tool was able to predict the measured SET cross section for an inverter, a NAND gate, and a NOR gate fabricated in the same technology. Characterization of basic library cells would enable the virtual irradiation of more complex logic designs.

A multi-scale simulation approach for SET response is demonstrated that eliminates or significantly reduces the over-estimation of the soft error response caused by assuming that every fault included in the error prediction equation translates to an error. Using the predicted cross-sections of the inverter, NAND gate, and the NOR gate, a distribution of transient pulses was generated for each of those basic cells to enable analysis at the logic level for transient capture. Typically, a worst-case duration (i.e., pulse width) is used for simulations at the IC level. However, a Monte Carlo method has been used in this dissertation to show the probability of error based upon incident particles in lightly ionizing environments. Since all digital functions can be synthesized from basic gates, a distribution of pulse widths from those gates can be used to analyze larger integrated circuits. This dissertation leaves the analysis of an entire cell library as future work, but a pathway is established that shows the feasibility of determining the error rate for an IC.

The multi-scale approach improved the efficiency of computing the effects from SETs over other techniques. Previous digital IC predictions of single-event vulnerability used TCAD simulations of all library cells to obtain a small subset of SET responses [75]. The multi-scale approach applied a number of the TCAD simulations on NMOSFETs and PMOSFETs to define the sensitive volume (SV) sizes. With a few iterations the MRED2SPICE charge collection efficiencies were able to replicate the experimental SET data presented by Cannon et al. [17]. With just a relatively small number of TCAD simulations and the MRED2SPICE iterations, a full set of SET responses for an entire library can be predicted. This dissertation describes the process for selected library cells. This method has the impact of both improving computation efficiency and model accuracy at the same time. Another computational efficiency improvement is seen with simulating smaller pieces of the digital IC in SPICE and then simulating the whole IC in ModelSim®. SPICE is an ideal tool to model the propagation of an analog transient signal in circuits [59], but once it becomes a digital transient signal, ModelSim® is an effective tool for computational efficiency.

The multi-scale simulation enables flexibility with respect to the design to be analyzed. For example, the ALU had been fully analyzed for logical-masking for all input conditions and all combinational logic cells in the design [59, 60]. Only logical-masking analysis for the ALU could be accomplished because each ALU operation was implemented in a single clock cycle. The outputs were solely a function of the ALU inputs and not a function of the previous state of the ALU. If the ALU was pipelined to improve its operating frequency, then it is not clear how this full logical-masking analysis could be performed in Black et al. Also, if any redundancy were added to the ALU, it would further complicate the analysis. There were no provisions in the testbench from Black et al. to handle this situation. The multi-scale simulation, however, applied Monte Carlo fault injection and demonstrated equivalency to the results seen in Black et al. [59]. With a Monte Carlo approach, redundancy and pipelining does not impact the complexity of the testing for the ALU, therefore it is robust for simulating these types of designs.

Another technical contribution of this research is the addition of a dual double-exponential current source for SET generation in SPICE. Other research has shown the limitations of the double-exponential current source [11], but this research developed a simple way to overcome these limitations that had reasonable accuracy. Three important parameters for a circuit being affected by a double-exponential current were described: (1) threshold charge, (2) prompt current, and (3) hold current. Simulation approaches were also described to determine these parameters were also described.

An additional technical contribution is the notion of $V_{r\text{-}r}$. IC modeling tools only accept digital signals as inputs. This research examined how to transition between SPICE (used to model the generation and propagation of voltage transients) and the IC modeling tool. $V_{r\text{-}r}$ provides an equivalent rail-to-rail pulse for this transition to preserve the important information from the voltage transient signal, V_{trans}, with respect to SET fault capture.

Future Work

For this dissertation, lightly ionizing particles were simulated in SPICE via a dual double-exponential current source. To expand beyond lightly ionizing particles and include all particles, better models must be developed. First, the quantity and size of the SVs implemented in MRED would grow. Second, research would have to be conducted on how to handle overlapping SVs or much larger SVs. Warren implemented overlapping SVs because his research was not limited to lightly ionizing particles or single small combinational cells. These large SVs would significantly extend beyond the boundary of a single small combinational cell, and would be affected by the SVs of the adjacent logic cells, for example for an ASIC or field programmable gate array. So, it would be a difficult task to construct SVs and their charge collection efficiencies without specific knowledge of the adjacent cells.

If the environment were to be expanded beyond the lightly ionizing particle environment, then more complex charge collection mechanisms and an increase in the number and shape of SVs would have to be modeled. Research involving the effects of well modulation is ongoing. To keep step with this research, these effects would need to be studied and modeled if they are to be added into the tool flow.

The next natural step leads to multiple node charge collection. In the current version of the multi-scale simulation, all charge collected at any transistor results in a current source being implemented in SPICE, so multiple node charge collection is handled in the individual cell. But, it is not considered for adjacent cells, and additional research will be needed to thoroughly and efficiently implement multiple cell charge collection.

Finally, logic synthesis from a cell library is typically used for modern IC design and implementation. In this dissertation, several library cells of the ALU were analyzed to determine their contribution to the full SER of the design. These cells were characterized for particles of normal incidence. An extension of this work would enable the SER prediction for an entire IC when the entire cell library is fully characterized for omnidirectional particle strikes. With a full profile of the incident particles, the framework would predict the distribution of transient pulses observed within circuits synthesized from the cell library. The appropriate testbench would observe the dynamic behavior of the IC to predict the SER.

REFERENCES

1. Moore, G.E., *Cramming More Components Onto Integrated Circuits.* Proceedings of the IEEE, 1998. **86**(1): p. 82-85.

2. Benini, L., Bogliolo, A. and De Micheli, G., *A survey of design techniques for system-level dynamic power management.* Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 2000. **8**(3): p. 299-316.

3. Shin, Y., Seomun, J., Choi, K.-M. and Sakurai, T., *Power gating: Circuits, design methodologies, and best practice for standard-cell VLSI designs.* ACM Trans. Des. Autom. Electron. Syst., 2010. **15**(4): p. 1-37.

4. Venkatachalam, V. and Franz, M., *Power reduction techniques for microprocessor systems.* ACM Comput. Surv., 2005. **37**(3): p. 195-237.

5. Moshovos, A. and Sohi, G.S., *Microarchitectural innovations: boosting microprocessor performance beyond semiconductor technology scaling.* Proceedings of the IEEE, 2001. **89**(11): p. 1560-1575.

6. Baumann, R.C., *Soft errors in advanced semiconductor devices-part I: the three radiation sources.* Device and Materials Reliability, IEEE Transactions on, 2001. **1**(1): p. 17-22.

7. Sierawski, B.D., Mendenhall, M.H., Reed, R.A., Clemens, M.A., Weller, R.A., Schrimpf, R.D., Blackmore, E.W., Trinczek, M., Hitti, B., Pellish, J.A. and Baumann, R.C., *Muon-induced single event upsets in deep-submicron technology.* Nuclear Science, IEEE Transactions on, 2010. **57**(6).

8. Sierawski, B.D., Pellish, J.A., Reed, R.A., Schrimpf, R.D., Warren, K.M., Weller, R.A., Mendenhall, M.H., Black, J.D., Tipton, A.D., Xapsos, M.A., Baumann, R.C., Xiaowei, D., Campola, M.J., Friendlich, M.R., Kim, H.S., Phan, A.M. and Seidleck, C.M., *Impact of Low-Energy Proton Induced Upsets on Test Methods and Rate Predictions.* Nuclear Science, IEEE Transactions on, 2009. **56**(6): p. 3085-3092.

9. Massengill, L.W., Baranski, A.E., Van Nort, D.O., Meng, J. and Bhuva, B.L., *Analysis of single-event effects in combinational logic-simulation of the AM2901 bitslice processor.* Nuclear Science, IEEE Transactions on, 2000. **47**(6): p. 2609-2615.

10. Benedetto, J.M., Eaton, P.H., Mavis, D.G., Gadlage, M. and Turflinger, T., *Variation of digital SET pulse widths and the implications for single event hardening of advanced CMOS processes.* Nuclear Science, IEEE Transactions on, 2005. **52**(6): p. 2114-2119.

11. Warren, K.M., *Sensitive Volume Models For Single Event Upset Analysis and Rate Prediction for Space, Atmospheric, and Terrestrial Radiation*

Environments, in *Electrical Engineering*. 2010, Vanderbilt University: Nashville, TN.

12. Warren, K.M., Sternberg, A.L., Black, J.D., Weller, R.A., Reed, R.A., Mendenhall, M.H., Schrimpf, R.D. and Massengill, L.W., *Heavy Ion Testing and Single Event Upset Rate Prediction Considerations for a DICE Flip-Flop.* Nuclear Science, IEEE Transactions on, 2009. **56**(6): p. 3130-3137.

13. Weller, R.A., Reed, R.A., Warren, K.M., Mendenhall, M.H., Sierawski, B.D., Schrimpf, R.D. and Massengill, L.W., *General Framework for Single Event Effects Rate Prediction in Microelectronics.* Nuclear Science, IEEE Transactions on, 2009. **56**(6): p. 3098-3108.

14. Tipton, A.D., Pellish, J.A., Reed, R.A., Schrimpf, R.D., Weller, R.A., Mendenhall, M.H., Sierawski, B., Sutton, A.K., Diestelhorst, R.M., Espinel, G., Cressler, J.D., Marshall, P.W. and Vizkelethy, G., *Multiple-Bit Upset in 130 nm CMOS Technology.* Nuclear Science, IEEE Transactions on, 2006. **53**(6): p. 3259-3264.

15. Weller, R.A., Schrimpf, R.D., Reed, R.A., Mendenhall, M.H., Warren, K.M., Sierawski, B.D. and Massengill, L.W., *Monte Carlo Simulation of Single Event Effects.* RADECS 2009 Short Course, 2009.

16. Cannon, E.H. and Cabanas-Holmen, M., *Heavy Ion and High Energy Proton-Induced Single Event Transients in 90 nm Inverter, NAND and NOR Gates.* Nuclear Science, IEEE Transactions on, 2009. **56**(6): p. 3511-3518.

17. Sierawski, B., *libsingleEvent Library Module*. 2010, Vanderbilt University, Institute for Space and Defence Electronics: Nashville, TN. p. 1-43.

18. Dodd, P.E. and Massengill, L.W., *Basic mechanisms and modeling of single-event upset in digital microelectronics.* Nuclear Science, IEEE Transactions on, 2003. **50**(3): p. 583-602.

19. Black, J.D., Reed, R.A., Hafer, C., Peterson, E., Benedetto, J. and Wilkinson, J., *Soft Errors: From the Ground Up.* NSREC Short Course, 2008.

20. Poivey, C., Barth, J.A., Reed, R., Stassinopoulos, E.G., LaBel, K.A. and Xapsos, M. *Implications of advanced microelectronics technologies for heavy ion single event effect (SEE) testing*. in *Radiation and Its Effects on Components and Systems, 2001. 6th European Conference on*. 2001.

21. Messenger, G.C., *Collection of Charge on Junction Nodes from Ion Tracks.* Nuclear Science, IEEE Transactions on, 1982. **29**(6): p. 2024-2031.

22. Baumann, R., *Soft errors in advanced computer systems.* Design & Test of Computers, IEEE, 2005. **22**(3): p. 258-266.

23. Baumann, R.C., *Radiation-induced soft errors in advanced semiconductor technologies.* Device and Materials Reliability, IEEE Transactions on, 2005. **5**(3): p. 305-316.

24. Mukherjee, S., *Architecture Design for Soft Errors*. 2008, Burlington, MA: Morgan Kaufmann.

25. Dicello, J.F., McCabe, C.W., Doss, J.D. and Paciotti, M., *The Relative Efficiency of Soft-Error Induction in 4K Static RAMS by Muons and Pions.* Nuclear Science, IEEE Transactions on, 1983. **30**(6): p. 4613-4615.

26. May, T.C. and Woods, M.H., *Alpha-particle-induced soft errors in dynamic memories.* Electron Devices, IEEE Transactions on, 1979. **26**(1): p. 2-9.

27. Rodbell, K.P., Heidel, D.F., Tang, H.H.K., Gordon, M.S., Oldiges, P. and Murray, C.E., *Low-Energy Proton-Induced Single-Event-Upsets in 65 nm Node, Silicon-on-Insulator, Latches and Memory Cells.* Nuclear Science, IEEE Transactions on, 2007. **54**(6): p. 2474-2479.

28. Binder, D., Smith, E.C. and Holman, A.B., *Satellite Anomalies from Galactic Cosmic Rays.* Nuclear Science, IEEE Transactions on, 1975. **22**(6): p. 2675-2680.

29. Constantinescu, C. *Impact of deep submicron technology on dependability of VLSI circuits*. in *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*. 2002.

30. Reed, R.A., Xapsos, M., Santini, G., Law, M., Black, J.D. and Holman, T., *Modeling the Space Radiation Environment and Effects on Microelectronic Devices and Circuits.* IEEE Short Course, 2006.

31. Seifert, N. and Tam, N., *Timing vulnerability factors of sequentials.* Device and Materials Reliability, IEEE Transactions on, 2004. **4**(3): p. 516-522.

32. Mukherjee, S.S., Weaver, C., Emer, J., Reinhardt, S.K. and Austin, T. *A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor*. in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*. 2003.

33. Dhillon, Y.S., Diril, A.U. and Chatterjee, A. *Soft-error tolerance analysis and optimization of nanometer circuits*. in *Design, Automation and Test in Europe, 2005. Proceedings*. 2005.

34. Ming, Z. and Shanbhag, N.R., *Soft-Error-Rate-Analysis (SERA) Methodology.* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2006. **25**(10): p. 2140-2155.

35. Rajaraman, R., Kim, J.S., Vijaykrishnan, N., Xie, Y. and Irwin, M.J. *SEAT-LA: a soft error analysis tool for combinational logic*. in *VLSI Design, 2006. Held jointly with 5th International Conference on Embedded Systems and Design., 19th International Conference on*. 2006.

36. Natasa, M.-Z. and Diana, M., *Circuit Reliability Analysis Using Symbolic Techniques.* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2006. **25**(12): p. 2638-2649.

37. Miskov-Zivanov, N. and Marculescu, D., *Modeling and Optimization for Soft-Error Reliability of Sequential Circuits.* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2008. **27**(5): p. 803-816.

38. Petersen, E.L., Pickel, J.C., Adams, J.H., Jr. and Smith, E.C., *Rate prediction for single event effects-a critique.* Nuclear Science, IEEE Transactions on, 1992. **39**(6): p. 1577-1599.

39. Bergeron, J., *Writing Testbenches - Functional Verification of HDL Models*. 2002, Norwell, MA: Kluwer Academic Publishers.

40. Narasimham, B., Bhuva, B.L., Schrimpf, R.D., Massengill, L.W., Gadlage, M.J., Amusan, O.A., Holman, W.T., Witulski, A.F., Robinson, W.H., Black, J.D., Benedetto, J.M. and Eaton, P.H., *Characterization of Digital Single Event Transient Pulse-Widths in 130-nm and 90-nm CMOS Technologies.* Nuclear Science, IEEE Transactions on, 2007. **54**(6): p. 2506-2511.

41. Kauppila, J.S., Sternberg, A.L., Alles, M.L., Francis, A.M., Holmes, J., Amusan, O.A. and Massengill, L.W., *A Bias-Dependent Single-Event Compact Model Implemented Into BSIM4 and a 90 nm CMOS Process Design Kit.* Nuclear Science, IEEE Transactions on, 2009. **56**(6): p. 3152-3157.

42. DasGupta, S., Witulski, A.F., Bhuva, B.L., Alles, M.L., Reed, R.A., Amusan, O.A., Ahlbin, J.R., Schrimpf, R.D. and Massengill, L.W., *Effect of Well and Substrate Potential Modulation on Single Event Pulse Shape in Deep Submicron CMOS.* Nuclear Science, IEEE Transactions on, 2007. **54**(6): p. 2407-2412.

43. Fan, W. and Agrawal, V.D. *Single Event Upset: An Embedded Tutorial*. in *VLSI Design, 2008. VLSID 2008. 21st International Conference on*. 2008.

44. Bose, P. *Tutorial: Power-aware, reliable microprocessor design*. in *VLSI Design, 2005. 18th International Conference on*. 2005.

45. Cavrois, V.F., Pouget, V., McMorrow, D., Schwank, J.R., Fel, N., Essely, F., Flores, R.S., Paillet, P., Gaillardin, M., Kobayashi, D., Melinger, J.S., Duhamel, O., Dodd, P.E. and Shaneyfelt, M.R., *Investigation of the Propagation Induced Pulse Broadening (PIPB) Effect on Single Event Transients in SOI and Bulk Inverter Chains.* Nuclear Science, IEEE Transactions on, 2008. **55**(6): p. 2842-2853.

46. Gouker, P., Brandt, J., Wyatt, P., Tyrrell, B., Soares, A., Knecht, J., Keast, C., McMorrow, D., Narasimham, B., Gadlage, M. and Bhuva, B., *Generation and Propagation of Single Event Transients in 0.18-um Fully Depleted SOI.* Nuclear Science, IEEE Transactions on, 2008. **55**(6): p. 2854-2860.

47. Massengill, L.W. and Tuinenga, P.W., *Single-Event Transient Pulse Propagation in Digital CMOS.* Nuclear Science, IEEE Transactions on, 2008. **55**(6): p. 2861-2871.

48. Ahlbin, J.R., Black, J.D., Massengill, L.W., Amusan, O.A., Balasubramanian, A., Casey, M.C., Black, D.A., McCurdy, M.W., Reed, R.A. and Bhuva, B.L., *C-CREST Technique for Combinational Logic SET Testing.* Nuclear Science, IEEE Transactions on, 2008. **55**(6): p. 3347-3351.

49. Benedetto, J., Eaton, P., Avery, K., Mavis, D., Gadlage, M., Turflinger, T., Dodd, P.E. and Vizkelethyd, G., *Heavy ion-induced digital single-event transients in deep submicron Processes.* Nuclear Science, IEEE Transactions on, 2004. **51**(6): p. 3480-3485.

50. Baze, M.P., Wert, J., Clement, J.W., Hubert, M.G., Witulski, A., Amusan, O.A., Massengill, L. and McMorrow, D., *Propagating SET Characterization Technique for Digital CMOS Libraries.* Nuclear Science, IEEE Transactions on, 2006. **53**(6): p. 3472-3478.

51. Dodd, P.E., Shaneyfelt, M.R., Felix, J.A. and Schwank, J.R., *Production and propagation of single-event transients in high-speed digital logic ICs.* Nuclear Science, IEEE Transactions on, 2004. **51**(6): p. 3278-3284.

52. Calin, T., Nicolaidis, M. and Velazco, R., *Upset hardened memory design for submicron CMOS technology.* Nuclear Science, IEEE Transactions on, 1996. **43**(6): p. 2874-2878.

53. Marshall, P., Carts, M., Currie, S., Reed, R., Randall, B., Fritz, K., Kennedy, K., Berg, M., Krithivasan, R., Siedleck, C., Ladbury, R., Marshall, C., Cressler, J., Guofu, N., LaBel, K. and Gilbert, B., *Autonomous bit error rate testing at multi-gbit/s rates implemented in a 5AM SiGe circuit for radiation effects self test (CREST).* Nuclear Science, IEEE Transactions on, 2005. **52**(6): p. 2446-2454.

54. Black, J.D., Sternberg, A.L., Alles, M.L., Witulski, A.F., Bhuva, B.L., Massengill, L.W., Benedetto, J.M., Baze, M.P., Wert, J.L. and Hubert, M.G., *HBD layout isolation techniques for multiple node charge collection mitigation.* Nuclear Science, IEEE Transactions on, 2005. **52**(6): p. 2536-2541.

55. Narasimham, B., Bhuva, B.L., Schrimpf, R.D., Massengill, L.W., Gadlage, M.J., Holman, T.W., Witulski, A.F., Robinson, W.H., Black, J.D., Benedetto, J.M. and Eaton, P.H., *Effects of Guard Bands and Well Contacts in Mitigating Long SETs in Advanced CMOS Processes.* Nuclear Science, IEEE Transactions on, 2008. **55**(3): p. 1708-1713.

56. Robinson, W.H., Alles, M.L., Bapty, T.A., Bhuva, B.L., Black, J.D., Bonds, A.B., Massengill, L.W., Neema, S.K., Schrimpf, R.D. and Scott, J.M. *Soft Error Considerations for Multicore Microprocessor Design*. in *Integrated Circuit Design and Technology, 2007. ICICDT '07. IEEE International Conference on*. 2007.

57. Amusan, O.A., Massengill, L.W., Bhuva, B.L., DasGupta, S., Witulski, A.F. and Ahlbin, J.R., *Design Techniques to Reduce SET Pulse Widths in Deep-Submicron Combinational Logic.* Nuclear Science, IEEE Transactions on, 2007. **54**(6): p. 2060-2064.

58. Olson, B.D., Amusan, O.A., Dasgupta, S., Massengill, L.W., Witulski, A.F., Bhuva, B.L., Alles, M.L., Warren, K.M. and Ball, D.R., *Analysis of Parasitic PNP Bipolar Transistor Mitigation Using Well Contacts in 130 nm and 90 nm CMOS Technology.* Nuclear Science, IEEE Transactions on, 2007. **54**(4): p. 894-897.

59. Black, J.D., Massengill, L., Bhuva, B., Holman, W.T. and Warren, K.M., *Final Report to Boeing Phantom Works*. 2008, Institute for Space and Defense Electronics,Vanderbilt University: Nashville, TN.

60. Atkinson, N.M., Witulski, A.F., Holman, W.T., Bhuva, B.L., Black, J.D. and Massengill, L.W., *Single Event Characterization of a 90 nm Bulk CMOS Digital Cell Library.* Proceedings of the Government Microcircuit Applications & Critical Technology Conference, 2010.

61. Hrishikesh, M.S., Jouppi, N.P., Farkas, K.I., Burger, D., Keckler, S.W. and Shivakumar, P. *The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays*. in *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*. 2002.

62. *The PowerPC 440 Core, A high-performance, superscalar processor core for embedded applicaitons*. 1999, IBM Microelectronics Division: Research Triangle Park, NC. p. 1-18.

63. Hartstein, A. and Puzak, T.R. *The optimum pipeline depth for a microprocessor*. in *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*. 2002.

64. Zyuban, V., Brooks, D., Viji, S., Gschwind, M., Pradip, B., Strenski, P.N. and Emma, P.G., *Integrated analysis of power and performance for pipelined microprocessors.* Computers, IEEE Transactions on, 2004. **53**(8): p. 1004-1016.

65. Clark, L., *Professor of Electrical Engineering*, Arizona State University, Technical Discussions with D.A. Black, 2011: Nashville.

66. *XScale Microarchitecture Technical Datasheet*. 2000, Intel Corporation.

67. *Intel XScale Core Developer's Manual*. 2004, Intel Corporation.

68. *ModelSim - Advanced Simulation and Debugging*. [cited 2010; 10.0d:[High Performance and Capacity Mixed HDL Simulation]. Available from: http://model.com/.

69. Amusan, O.A., Massengill, L.W., Baze, M.P., Bhuva, B.L., Witulski, A.F., Black, J.D., Balasubramanian, A., Casey, M.C., Black, D.A., Ahlbin, J.R., Reed, R.A. and McCurdy, M.W. *Mitigation techniques for single event induced charge sharing in a 90 nm bulk CMOS process*. in *Reliability Physics Symposium, 2008. IRPS 2008. IEEE International*. 2008.

70. Amusan, O.A., Massengill, L.W., Baze, M.P., Bhuva, B.L., Witulski, A.F., Black, J.D., Balasubramanian, A., Casey, M.C., Black, D.A., Ahlbin, J.R., Reed, R.A. and McCurdy, M.W., *Mitigation Techniques for Single-Event-Induced Charge Sharing in a 90-nm Bulk CMOS Process.* Device and Materials Reliability, IEEE Transactions on, 2009. **9**(2): p. 311-317.

71. Black, J.D., Ball, D.R., Robinson, W.H., Fleetwood, D.M., Schrimpf, R.D., Reed, R.A., Black, D.A., Warren, K.M., Tipton, A.D., Dodd, P.E., Haddad, N.F., Xapsos,

M.A., Kim, H.S. and Friendlich, M., *Characterizing SRAM Single Event Upset in Terms of Single and Multiple Node Charge Collection.* Nuclear Science, IEEE Transactions on, 2008. **55**(6): p. 2943-2947.

72. Benjamin, M., Geist, D., Hartman, A., Wolfsthal, Y., Mas, G. and Smeets, R. *A study in coverage-driven test generation*. in *Design Automation Conference, 1999. Proceedings. 36th*. 1999.

73. *SystemVerilog 3.1, Accellera's Extensions to Verilog*. 2003, Accellera.

74. Tylka, A.J., Adams, J.H., Jr., Boberg, P.R., Brownstein, B., Dietrich, W.F., Flueckiger, E.O., Petersen, E.L., Shea, M.A., Smart, D.F. and Smith, E.C., *CREME96: A Revision of the Cosmic Ray Effects on Micro-Electronics Code.* Nuclear Science, IEEE Transactions on, 1997. **44**(6): p. 2150-2160.

75. Boulghassoul, Y., Rowe, J.D. and Massengill, L.W., *Applicability of circuit macromodeling to analog single-event transient analysis.* Nuclear Science, IEEE Transactions on, 2003. **50**(6): p. 2119-2125.

APPENDIX A

MRED INPUT PYTHON SCRIPT

```
#####################################################################
# Dolores Black 5/08/2011
# INV1 MRED single layer, 5 nested sensitive volumes,
# per transistor  -Boeing data 2.2 LET
#####################################################################

from PyG4Core import G4Colour as G4Color
import sys, os, base64, cPickle

#---------------------------------------------------------------------------
# Set up variable to use for output array to be used
#   for analysis of charge to current for SET
#    FDEL      = Output File Delimeter
#    FNAME     = Output File Name
#    MAXBUFFER = Max length of results between rights (don't swamp file system)
#---------------------------------------------------------------------------

FDEL = ',' # Output File Delimeter
FNAME = 'q0_q9boeinginvnsv2LET.out' # Output File Name
MAXBUFFER = 50 # Max length of results between rights (don't swamp file system)

## Set this to "False" if you don't want to deal with mred's viewer and just want the text output
#I just put this switch in because the dx viewer is a little clunky. You can play around with it
# and try to fade-out the various parts and see if you can see some events and the sv. I could
# do it just to verify that things were going, but it's not that informative. The viewers
# are usually just good to make sure your SV placement isn't crazy

doViewer = False

# Turn on the basic materials (silicon, tungsten, sio2, etc.)
# Mred has to know what materials it is going to need. This can be as rich as we want it but it's better
# to be a minimalist because it has to build a lot of tables re physics.

mred.materials.enableBasicElectronicMaterials()

#Select electronic stopping power physics model
#Electronic stopping is the basic energy loss due to electronic stopping. Nuclear physics (reactions, etc.) are
#not taken into account. You do not need to worry about nuclear physics yet. It's not more complicated, per se, just
#takes longer to run. If you need nukes, then we can turn them on
mred.physics.addModule('StandardScreened')
mred.physics.module_dict['Decay'].SetIncludeRadioactiveDecay(False)

#Establish what kind of device we need (basic RPP layer, TCAD, etc.)
# Create an instance of a RPP device
d = mred.setDevice('rpp')

#Build a list of layers with x,y,z dimensions. These stack in the list as shown top to bottom (physically)
d.setLayers([
#((5.0*um, 5.*um, 1.*um), 'tungsten'),
#((5.*um, 5.*um, 5.*um), 'SiO2'),
((5.*um, 5.*um, 5.*um), 'silicon','sd1')])


#Embed the device in a silicon wafer - the silicon should contain the silicon layer above
# This basically fills in all possible voids left by the setLayers command with silicon
d.wafer_material = 'silicon'
```

```
#Register the device
d.register()

# Initialize the simulator
mred.init()

# Define sensitive volume inside sensitive detector

svList = [] # create a new empty list (array)
sd1 = mred.sd_vector[0] #get the zeroth entry of the table contained in mred
sd1.coincidence_order=1 #1 will turn on min_valid_energy filter, otherwise, the events all go to singleEventCallback

# Let's add four sensitive volumes (these are physically separate, but you can move them around and nest them too)
# PMOS transistor SVs

sv=sd1.addSensitiveVolume("rpp")
sv.size=(0.26*um,0.84*um,0.35*um)
sv.center=(0.59*um,2.17*um,0.175*um)
sv.min_valid_total_energy = 1*keV # This is the trigger point, set it low, but high enough to drop zeros and the stray d ray
svList.append(sv)

sv=sd1.addSensitiveVolume("rpp")
sv.size=(0.26*um,0.84*um,0.35*um)
sv.center=(0.42*um,2.17*um,0.175*um)
sv.min_valid_total_energy = 1*keV
svList.append(sv)

sv=sd1.addSensitiveVolume("rpp")
sv.size=(0.52*um,1.14*um,0.30*um)
sv.center=(0.42*um,2.17*um,0.175*um)
sv.min_valid_total_energy = 1*keV
svList.append(sv)

sv=sd1.addSensitiveVolume("rpp")
sv.size=(0.88*um,1.24*um,0.40*um)
sv.center=(0.59*um,2.17*um,0.55*um)
sv.min_valid_total_energy = 1*keV
svList.append(sv)

# Let's add four sensitive volumes (these are physically separate, but you can move them around and nest them too)
# NMOS transistor SVs

sv=sd1.addSensitiveVolume("rpp")
sv.size=(0.26*um,0.28*um,0.35*um)
sv.center=(0.59*um,1.08*um,0.175*um)
sv.min_valid_total_energy = 1*keV # This is the trigger point, set it low, but high enough to drop zeros and the stray d ray
svList.append(sv)

sv=sd1.addSensitiveVolume("rpp")
sv.size=(0.62*um,0.28*um,0.35*um)
sv.center=(0.42*um,1.08*um,0.175*um)
sv.min_valid_total_energy = 1*keV
svList.append(sv)

sv=sd1.addSensitiveVolume("rpp")
sv.size=(0.52*um,0.48*um,0.30*um)
sv.center=(0.42*um,1.08*um,0.175*um)
sv.min_valid_total_energy = 1*keV
svList.append(sv)

sv=sd1.addSensitiveVolume("rpp")
sv.size=(0.88*um,0.58*um,0.40*um)
sv.center=(0.59*um,1.08*um,0.55*um)
sv.min_valid_total_energy = 1*keV
svList.append(sv)


# Set up the gun
#-----------------------------------------------------------------------------
```

```
# Loops to set-up, evaluate and populate output buffer array
#----------------------------------------------------------------------------

OutputBuffer = []

# Method to dump results buffer to file
def clearBuffer(out_buff):
    global FNAME,FDEL
    f=open(FNAME,'a')
    for buff in out_buff:
        f.write("%d%s%.3e%s" % (buff[0],FDEL,buff[1],FDEL))
        for i in range(len(buff[2])):
            f.write("%.4f%s" % (buff[2][i],FDEL))
        f.write("%.4f\n" % buff[2][-1])
    f.close()

def singleEventCallback(evt):
    global svList,doViewer,OutputBuffer,MAXBUFFER
    # Compact notation

    Qs = [svList[i].total_energy/0.0225 for i in range(len(svList))]

    #### Or, the long way
    #Qs = []
    #for i in range(len(svList)):
    #       Qs.append(svList[i].total_energy/22.5)

    if max(Qs) < 1.0: return # if there's nothing more than 0.1 fC, return)

    eventNumber = mred.runMgr.GetEventCount()-1
    weight = mred.evtAct.ComputeEventWeight(evt)
    variance = weight**2

    print "Event #%d:" % (eventNumber),

    for i in range(len(svList)):
        print "Q%d=%.4f(fC)" % (i,Qs[i]),
    print "Statistical Weight=%.3e" % (weight)

    OutputBuffer.append([eventNumber,weight,Qs])

    if len(OutputBuffer) > MAXBUFFER:
        clearBuffer(OutputBuffer)
        OutputBuffer = []

    sys.stdout.flush()

    if doViewer:
        mred.dx.displayMredEvent(evt)

# set Particle takes atomic number and atomic mass

mred.gun.setParticle('ion', 8, 18) #Oxygen, 2.2 LET
mred.gun.energy=183*MeV

#Spatial sampling "directionalFlux" is like a broadbeam experiment. Randomizes over a plane in the direction
# of the gun direction vector.
mred.gun.random_spatial_sampling = "directionalFlux"
mred.gun.random_use_device_radius=True
mred.gun.direction=vector3d(0,0,-1)

# There is no need to do this in our case, at least not right now

# Accumlate histograms

#mred.accumulate_histograms = True
#mred.hdf5.file_path=("/home/blackda1/hdf5_output")
#mred.hdf5.file_name="stack_Random40MeV10Runs.hdf5"
#mred.hdf5.write_output_files=True
```

```
#mred.hdf5.include_energies=False
#mred.hdf5.include_tracks=False
#mred.hdf5.include_hits=False
#mred.hdf5.include_histograms=True

# Visualize with Open DX

#mred.dx.captureGeometry([mred.detCon.GetPhysicalWorld()]+sd1.g4PV())
# Get sensitive volume's visual attributes
if doViewer:

        dVis = d.g4PV().GetLogicalVolume().GetVisAttributes()
        dVis.SetColor(G4Color(0,0,1,1))
        mred.dx.captureGeometry([mred.detCon.GetPhysicalWorld(), d.g4PV()],
                                                  opacity_multiplier=1,custom_color_map=('silicon' : G4Color(0,1,0,.5),))

# Now, let's run mred for x ions using singleEventCallback as our interrupt
# interrupts will be invoked based on the condition that the defined amount of min_valid_energy for each sv
# is deposited in that sv. In other words, at least one has to be triggered

mred.runSingleEventMode(1000000,function=singleEventCallback)

if len(OutputBuffer) > 0:
        clearBuffer(OutputBuffer)

## Note that the 'statistical weight' is always one in this case. That is because there is no fancy nuclear event biasing
## Since you are not yet looking at nukes.

## The true statistical weight for each event is actually not 1, but it is pi*r^^2 (see my dissertation appendix).
## Note that the units work out for the weight as area (e.g., cross section in cm^2 or um^2, you get to figure it all out)
## you can get this value from mred directly:

eventWeightFactor = 1.0/mred.gun.fluence_unit

print eventWeightFactor

## So, each event from singleEventCallback says it has a weight of "1" but in reality is has a weight of eventWeightFactor.
## The reason we don't propagate it f
## or each event and just get it after the fact is that it is a constant. It is a function of the geometry of the simulation
## so it doesn't change. It's just a matter of convenience, or inconvenience, to have to get it to do a cross section calculation

## A total cross section is then eventWeightFactor*sum(weights of events from singleEventCallback that meet your
## criteria)/(Number of total events run)

## If you think about this calculation, it is pretty intuitive. If every event you run is a valid event, then the
## total cross section is the eventWeightFactor, which is just the total surface area of the plane from
## which the particles are fired. If half of the plane is sensitive and half isn't,  then you'll
## have half of your events be counted relative to the total number run and your cross section
## will be 1/2 of the eventWeightFactor. You're just using Monte-Carlo estimation methods to estimate
## that area here, and that's it
```

APPENDIX B

4-INVERTER CHAIN BASIC CIRCUIT SPICE NETLIST

```
* Test circuit for Python

.subckt INV1 in_inv1 out_inv1 vdd_inv1 vss_inv1
XPINV1 vdd_inv1 in_inv1 out_inv1 vdd_inv1 pfet w=840n l=80n ad=218f pd=2.2u
XNINV1 out_inv1 in_inv1 vss_inv1 vss_inv1 nfet w=280n l=80n ad=73f pd=1.1u
.ends INV1

I1P nr1 n1 exp (0u 109.0u 100p 2p 115p 4p)
I1H nr1 n1 exp (0u 121.0u 100p 2p 500p 10p)

XP1 vdd in n1 vdd pfet w=840n l=80n ad=218f pd=2.2u
XN1 n1 in vss vss nfet w=280n l=80n ad=73f pd=1.1u
XINV1 n1 n2 vdd vss INV1
XINV2 n2 n3 vdd vss INV1
XINV3 n3 out vdd vss INV1
XINV4 out n4 vdd vss INV1


V1 vdd 0 0.9
V2 in 0 0.9
V3 vss 0 0.0
.tran 1ps 1ns
.printfile tran v(n1) v(n2) v(n3) v(out) file=alltransientI1.out
.printfile tran v(n1) v(out) file=transientI1.out
```

APPENDIX C

PYTHON SCRIPT FOR I_{thresh}, I_{prompt}, I_{hold}

```
# ---------------------------------------------------------------------------
# Filename:    findthreshandholdcurrents_inv1_i1.py
# Version:     v1
# Description: Python Code for determining
#                        Threshold/Prompt/Hold Currents
# ---------------------------------------------------------------------------
# Author:      Dolores Black, Vanderbilt University
# History:
#    DAB  4-25-2011      -- First Version
# ---------------------------------------------------------------------------
# Naming Conventions:
#
# ---------------------------------------------------------------------------
# Full Description:
# ---------------------------------------------------------------------------
#    Python script to search for the threshold current (total current needed)
#    to drive to the rail. This script also determines the prompt and hold
#    currents (IP + IH) = Ithresh.
#
# ---------------------------------------------------------------------------
# Boeing INV1 I1 current determination
# ---------------------------------------------------------------------------
#    I1 is charge collection on the output of the PMOSFET drain. Threshold
#    current is defined as the prompt current needed to drive the transient
#    voltage to the rail, but may not be necessarily the minimum to propagate
#    a transient signal. Hold current is defined as the current needed to
#    maintain the voltage at the rail once it's at the rail.
# ---------------------------------------------------------------------------

# ---------------------------------------------------------------------------
# import os is a command that allows the running of command line functions
# in Python
# ---------------------------------------------------------------------------
import os

# ---------------------------------------------------------------------------
# Set default/initial values to search for the threshold voltage
# - foundthresh => loop variable
#                 0 = threshold not found
#                 1 = threshold found or too many tries "executed"
# - ithresh     => I1 current (uA) for the exponential current source in ELDO
# - iincre      => Incremental current (uA) or step-size for the search
# ---------------------------------------------------------------------------
foundthresh = 0
ithresh = 100.0
iincre = 100.0

# ---------------------------------------------------------------------------
# Main (While) loop for searching for the threshold current
# ---------------------------------------------------------------------------

while (foundthresh < 1):

# ---------------------------------------------------------------------------
# Sanity check, to print values through the loop
# ---------------------------------------------------------------------------
   print ithresh

# ---------------------------------------------------------------------------
# First create a file -circuittest.cir- in tmp directory to -write to-
#    It is referred to as -eldodeck-
# ----------------------------------------------------------------------------
  eldodeck = open('tmp/circuittest.cir', 'w')

# ----------------------------------------------------------------------------
# The following lines in -eldodeck- populate the lines -circuittest.cir- file
#    ' and " are interchangeable.
#    If ' is needed in the line, then use " to enclose the statement, vice verse
#    \n performs a carriage return

# ----------------------------------------------------------------------------
#    First create/write the subcircuit
# ----------------------------------------------------------------------------
  eldodeck = open('tmp/circuittest.cir', 'w')
  eldodeck.write('* Test circuit for Python\n\n')
  eldodeck.write(".lib '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./skew.file' stats\n")
  eldodeck.write(".inc '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./fixed_corner'\n")
  eldodeck.write(".inc '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./hspice.param'\n")
  eldodeck.write(".inc '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./nfet.inc'\n")
  eldodeck.write(".inc '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./pfet.inc'\n\n\n")
  eldodeck.write(".subckt INV1 in_inv1 out_inv1 vdd_inv1 vss_inv1\n")
  eldodeck.write("XPINV1 vdd_inv1 in_inv1 out_inv1 vdd_inv1 pfet w=840n l=80n ad=218f pd=2.2u\n")
  eldodeck.write("XNINV1 out_inv1 in_inv1 vss_inv1 vss_inv1 nfet w=280n l=80n ad=73f pd=1.1u\n")
  eldodeck.write(".ends INV1\n\n\n")
# ----------------------------------------------------------------------------
#    Create the string to write the current source using the ithresh variable
# ----------------------------------------------------------------------------
  cl = 'I1P vdd n1 exp (0u ' + str(ithresh) + 'u 100p 2p 115p 4p)\n\n'

# ----------------------------------------------------------------------------
#    Finish creating/writing the remainder of the netlist
# ----------------------------------------------------------------------------
  eldodeck.write(cl)
  eldodeck.write("XP1 vdd in n1 vdd pfet w=840n l=80n ad=218f pd=2.2u\n")
  eldodeck.write("XN1 n1 in vss vss nfet w=280n l=80n ad=73f pd=1.1u\n")
  eldodeck.write("XINV1 n1 n2 vdd vss INV1\n")
  eldodeck.write("XINV2 n2 n3 vdd vss INV1\n")
  eldodeck.write("XINV3 n3 out vdd vss INV1\n")
  eldodeck.write("XINV4 out n4 vdd vss INV1\n\n\n")
  eldodeck.write("V1 vdd 0 0.9\n")
  eldodeck.write("V2 in 0 0.9\n")
  eldodeck.write("V3 vss 0 0.0\n")
  eldodeck.write(".tran 1ps 1ns\n")
  eldodeck.write(".printfile tran v(n1) file=transient.out\n")

# ----------------------------------------------------------------------------
#    Close the file when done, as required by Python
# ----------------------------------------------------------------------------
  eldodeck.close()

# ----------------------------------------------------------------------------
# Run/Invoke circuittest.cir netlist in Eldo, use same OS command
# ----------------------------------------------------------------------------
  os.system('eldo -compat -i tmp/circuittest.cir')

# ----------------------------------------------------------------------------
# Open the newly created output file -transient.out- from resulting simulation
#    in /tmp directory
# Using variables:
#    eldoresult - pointer for transient.out file
#    erlines - Read all the lines of -transient.out-int an array of strings
#            erlines(1) is the first line, erlines(2) is the second and so on
#            Dynamically assigned the array based upon the length of the file
# Close the file
# ----------------------------------------------------------------------------
  eldoresult = open('tmp/transient.out', 'r')
  erlines = eldoresult.readlines()
  eldoresult.close()
```

```
# -----------------------------------------------------------------------------
# Determine if the threshold current guess is too low or too high
#   Begin by assuming the guess is too low and NOT too high
#
#   Variables:
#       toolow => default =1 (too low)
#       toohigh=> default =0 (Not too high)
# -----------------------------------------------------------------------------
  toolow = 1
  toohigh = 0

# -----------------------------------------------------------------------------
#   Loop through each line in the result file
#       find ('#')     - Ignores any line with a '#' (comments, not results)
#       strip()        - Remove carriage returns /n from the output lines
#       split()        - Split the line by a space between 2 numbers into
#                      strings a & b
#       float()        - Convert b into a real number and call it volts
# -----------------------------------------------------------------------------
  for line in erlines:
    if line.find('#')<0:
      line=line.strip()
      a,b=line.split(' ')
      volts=float(b)

# -----------------------------------------------------------------------------
#   Determine the range for the threshold
#       If voltage exceeds 0.89 volts, then ithresh is NOT too low
#       If voltage exceeds 0.91 volts, then it is too high
# -----------------------------------------------------------------------------
      if volts > 0.89:
        print volts
        toolow = 0
      # End if
      if volts > 0.91:
        print volts
        toohigh = 1
      # End if
    # End if
  #End for

# -----------------------------------------------------------------------------
#   Once all the voltages in the output file are evaluated
#   Determine whether or not ithresh was too low or high
#       If too low, add iincre to ithresh and reloop
#       If too high, reduce ithresh by iincre and cut iincre by factor of 10
#           then add the new iincre to ithresh =>
#               It is assumed that the previous ithresh was too low
#   If the result is neither too low or too high, exit loop
# -----------------------------------------------------------------------------
  if toolow > 0:
    ithresh = ithresh + iincre
  # End if
  elif toohigh > 0:
    ithresh = ithresh - iincre
    iincre = iincre / 10.0
    ithresh = ithresh + iincre
  # End elif
  else:
    foundthresh = 1
  # End else
  print toolow, toohigh, foundthresh

# -----------------------------------------------------------------------------
# End Main (While) loop
# -----------------------------------------------------------------------------

# -----------------------------------------------------------------------------
# Print to screen final ithresh value
# ----------------------------------------------------------------------------
print ithresh

# ----------------------------------------------------------------------------
# Begin searching for prompt/hold currents
#    Variables:
#        foundhold => loop variable
#        trials    => Number of attempts to loop to determine ithresh/ihold
#        ihold     => Evaluated Hold current variable
# ----------------------------------------------------------------------------
foundhold = 0
trials = 0

# ----------------------------------------------------------------------------
#    Determine/Guess if the threshold current is greater than 110u set
#        ihold default = 100u
#        iincre default = 100u
#    Otherwise set
#        ihold  = 10u
#        iincre = 10u
#
# ----------------------------------------------------------------------------
if ithresh > 110:
   ihold = 100
   iincre = 100
# End if
else:
   ihold = 10
   iincre = 10
# End else

# ----------------------------------------------------------------------------
# 2nd Main (While) loop for searching for the prompt/hold currents
# ----------------------------------------------------------------------------
while (foundhold < 1):

# ----------------------------------------------------------------------------
# Sanity check, to print hold current values through the loop
# ----------------------------------------------------------------------------
   print ihold

# ----------------------------------------------------------------------------
#    Start counter for number of trials to determine if a prompt/hold current
#    can be determined
# ----------------------------------------------------------------------------
   trials = trials + 1

# ----------------------------------------------------------------------------
# The following lines in -eldodeck- populate the lines -circuittest.cir- file
#    ' and " are interchangeable.
#    If ' is needed in the line, then use " to enclose the statement, vice verse
#    \n performs a carriage return

# ----------------------------------------------------------------------------
#    First create/write the subcircuit
# ----------------------------------------------------------------------------
   eldodeck = open('tmp/circuittestI1.cir', 'w')
   eldodeck.write('* Test circuit for Python\n\n')
   eldodeck.write(".lib '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./skew.file' stats\n")
   eldodeck.write(".inc '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./fixed_corner'\n")
   eldodeck.write(".inc '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./hspice.param'\n")
   eldodeck.write(".inc '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./nfet.inc'\n")
   eldodeck.write(".inc '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./pfet.inc'\n\n\n")
   eldodeck.write(".subckt INV1 in_inv1 out_inv1 vdd_inv1 vss_inv1\n")
   eldodeck.write("XPINV1 vdd_inv1 in_inv1 out_inv1 vdd_inv1 pfet w=840n l=80n ad=218f pd=2.2u\n")
   eldodeck.write("XNINV1 out_inv1 in_inv1 vss_inv1 vss_inv1 nfet w=280n l=80n ad=73f pd=1.1u\n")
   eldodeck.write(".ends INV1\n\n\n")

# ----------------------------------------------------------------------------
#   Create the string to write the prompt current source using
#       (ithresh-ihold) variable's quantity
# ---------------------------------------------------------------------------
  cl1 = 'I1P vdd n1 exp (0u ' + str(ithresh-ihold) + 'u 100p 2p 115p 4p)\n'
  eldodeck.write(cl1)

# ---------------------------------------------------------------------------
#   Create the string to write the hold current source using
#       ihold variable's quantity
# ---------------------------------------------------------------------------
  cl2 = 'I1H vdd n1 exp (0u ' + str(ihold) + 'u 100p 2p 500p 10p)\n\n'
  eldodeck.write(cl2)

# ---------------------------------------------------------------------------
#   Finish creating/writing the remainder of the netlist
# ---------------------------------------------------------------------------
  eldodeck.write("XP1 vdd in n1 vdd pfet w=840n l=80n ad=218f pd=2.2u\n")
  eldodeck.write("XN1 n1 in vss vss nfet w=280n l=80n ad=73f pd=1.1u\n")
  eldodeck.write("XINV1 n1 n2 vdd vss INV1\n")
  eldodeck.write("XINV2 n2 n3 vdd vss INV1\n")
  eldodeck.write("XINV3 n3 out vdd vss INV1\n")
  eldodeck.write("XINV4 out n4 vdd vss INV1\n\n\n")
  eldodeck.write("V1 vdd 0 0.9\n")
  eldodeck.write("V2 in 0 0.9\n")
  eldodeck.write("V3 vss 0 0.0\n")
  eldodeck.write(".tran 1ps 1ns\n")
  eldodeck.write(".printfile tran v(n1) v(n2) v(n3) v(out) file=alltransientI1.out\n")
  eldodeck.write(".printfile tran v(n1) v(out) file=transientI1.out\n")

# ---------------------------------------------------------------------------
#   Close the file when done, as required by Python
# ---------------------------------------------------------------------------
  eldodeck.close()

# ---------------------------------------------------------------------------
# Run circuittest.cir netlist in Eldo, use same OS command
# ---------------------------------------------------------------------------
  os.system('eldo -compat -i tmp/circuittestI1.cir')

# ---------------------------------------------------------------------------
# Open the newly created output file -transientI1.out- from resulting simulation
#   in /tmp directory
# Using variables:
#   eldoresult - pointer for transientI1.out file
#   erlines - Read all the lines of -transientI1.out-int an array of strings
#           erlines(1) is the first line, erlines(2) is the second and so on
#           Dynamically assigned the array based upon the length of the file
# Close the file
# ---------------------------------------------------------------------------
  eldoresult = open('tmp/transientI1.out', 'r')
  erlines = eldoresult.readlines()
  eldoresult.close()

# ---------------------------------------------------------------------------
# Determine if the threshold current guess is too low or too high
#   Begin by assuming the guess is too low and NOT too high
#
#   Variables:
#       toolow => default =1 (too low)
#       toohigh=> default =0 (Not too high)
# ---------------------------------------------------------------------------
  toolow = 1
  toohigh = 0

# ---------------------------------------------------------------------------
#   Loop through each line in the result file
#       find ('#')    - Ignores any line with a '#' (comments, not results)
#       strip()       - Remove carriage returns /n from the output lines
#       split()       - Split the line by a space between 2 numbers into
#                    strings a, b & c (three outputs to plot)
#        float()       - Convert b into a real number and call it volts
# ---------------------------------------------------------------------------
   for line in erlines:
      if line.find('#')<0:
         line=line.strip()
         a,b,c=line.split(' ')
         time=float(a)
         volts=float(b)

# ---------------------------------------------------------------------------
#    Determine the range for the threshold after simulation has settled (400ps)
#        If voltage exceeds 0.89 volts, then ithresh is NOT too low
#        If voltage exceeds 0.91 volts, then it is too high
# ---------------------------------------------------------------------------
         if time > 0.0000000004:
            if volts > 0.89:
               print volts
               toolow = 0
            # End if
            if volts > 0.91:
               print volts
               toohigh = 1
            # End if
         # End if
      # End if
   # End for

# ---------------------------------------------------------------------------
#    Once all the voltages in the output file are evaluated
#    Determine whether or not ihold was too low or high
#        If too low, add iincre to ihold and reloop
#        If too high, reduce ihold by iincre and cut iincre by factor of 10
#            then add the new iincre to ihold
#            It is assumed that the previous ihold was too low
#    If the result is neither too low or too high, exit loop when trials exceeds
#        a count of 25
# ---------------------------------------------------------------------------
   if toolow > 0:
      ihold = ihold + iincre
   # End if
   elif toohigh > 0:
      ihold = ihold - iincre
      iincre = iincre / 10.0
      ihold = ihold + iincre
   # End elif
   else:
      foundhold = 1
   # End else
   if trials > 25:
      foundhold = 1
   # End if
   print toolow, toohigh, foundhold

# ---------------------------------------------------------------------------
# End 2nd Main (While) loop for prompt/hold current determination
# ---------------------------------------------------------------------------

# ---------------------------------------------------------------------------
# Print to screen final ithresh, ihold values
# ---------------------------------------------------------------------------
print ithresh, ihold
```

APPENDIX D

PYTHON SCRIPT FOR MRED2SPICE

```
# ---------------------------------------------------------------------------
# Filename:    mredtospice_inv1set.py
# Version:     v1
# Description: Python Code for determining Threshold/Prompt/Hold Currents
#              Create/Simulate a SPICE netlist
# ---------------------------------------------------------------------------
# Author:      Dolores Black, Vanderbilt University
# History:
#   DAB  4-30-2011     -- First Version
#   DAB  5-08-2011     ~~ Modify for script to add analysis for charge
#                         efficiencies in the q1/q2 analysis "loops"
#   DAB  5-09-2011     ** Modified for second "guess" at SV's (4 N/PMOS)
#   DAB  5-10-2011     ^^ Modified efficiencies of Substrate, too much charge
# ---------------------------------------------------------------------------
# Naming Conventions:
#
# ---------------------------------------------------------------------------
# Full Description:
# ---------------------------------------------------------------------------
#   Python script to:
#       1. Parse MRED charge collection (q1-q6, etc) output results
#          file given the ions, sensitive volumes, etc.
#       2. Extract the relevant data
#            Only operate on charge greater than the threshold necessary
#            to cause enough current to cause the voltage to cross the
#            circuit/cell's rail
#       3. Determine the IxP/H values resulting from charge converted
#            to current that exceeds the threshold
#            a. Calculate/Determine the pulse-width (td2) resulting from
#               excess charge greater than the minimum ihold current
#       4. Create a circuit netlist using IxP/H, td2 given the default dbl-
#            exp tau1, tau2 and td1 (2ps, 4ps(prompt)/10ps(hold), 100ps
#       5. Execute Spice on netlist
#            a. Output resulting pulse-widths and/or upset (output of latch)
# ---------------------------------------------------------------------------
# Boeing INV1 MRED results Parser determination
# ---------------------------------------------------------------------------
#   Q1 is charge collection on the output of the PMOSFET drain. Threshold
#   current is defined as the prompt current needed to drive the transient
#   voltage to the rail, but may not be necessarily the minimum to propagate
#   a transient signal. Hold current is defined as the current needed to
#   maintain the voltage at the rail once it's at the rail.
#
#   Q2 is charge collection on the output of the NMOSFET drain. Threshold
#   current is defined as the prompt current needed to drive the transient
#   voltage to the rail, but may not be necessarily the minimum to propagate
#   a transient signal. Hold current is defined as the current needed to
#   maintain the voltage at the rail once it's at the rail.
# ---------------------------------------------------------------------------


# ---------------------------------------------------------------------------
# import os is a command that allows the running of command line functions
# in Python
# import random is a command that allows for the import of random numbers to
#   be used for the placement of the dbl-exp SET current sources in circuit
# ---------------------------------------------------------------------------
```

```
import os
import random
latchcnt = 0

latchsetresult = open('latchresults9LET_v10.out', 'w')
latchsetresult.write("MRED# node # I1P 12P td21 td22\n")

# ----------------------------------------------------------------------------
# Open the MRED output results file, read it and close it
# in python
# ----------------------------------------------------------------------------
mredresult = open('mredresults/q0_q9boeinginvnsv9LET.out', 'r')
mredlines = mredresult.readlines()
mredresult.close()

# ----------------------------------------------------------------------------
# Main For loop to parse output file and calculate resulting charges and
#   convert to associated currents and the pulse-width variable td2
#   Variables:
#       line      => loop variable
#       event,c-h,
#           i-k => column in the output file to be read
#       q1       => charge collection output on PMOSFET from MRED
#                  q1thresh converted from i1thresh
#                   determined from cell characterization
#       q2       => charge collection output on NMOSFET from MRED
#                  q2prompt converted from i2prompt
#                   determined from cell characterization
#       q1h/q2h   => q(MRED result) - qprompt (pre-characterized)
#                  q1(2)prompt = 2.29fc/2.14fc converted from iprompt
#                  determined from cell characterization
#       td21      => SET pulse-width variable for PMOSFET
#       td22      => SET pulse-width variable for NMOSFET
#       runsim    => If true (1) then create Spice netlist to run, otherwise
#                  go to next Q value for evaluation
# ----------------------------------------------------------------------------
for line in mredlines:
    line=line.strip()
    event,b,c,d,e,f,g,h,i,j,k=line.split(',')
    i1p = 0.0
    i1h = 0.0
    td21 = 0.0
    i2p = 0.0
    i2h = 0.0
    td22 = 0.0
    q1=float(c)*0.6 + float(d)*0.2 + float(e)*0.70 + float(f)*0.10
    q2=float(g)*0.75 + float(h)*0.25 + float(i)*0.55 + float(j)*0.10
    runsim=0

# ----------------------------------------------------------------------------
# Start with PMOSFET pre-characterized for:
#   Qthresh   = 4.8 fc
#   I1prompt  = 109.0 uA
#   I1hold    = 121.0 uA
#   tau1+tau2 = 12.0 ps
#   td2-td1   = 115ps-100ps = 15ps
#   Q1prompt  ~ I1prompt*(21)/1000.0
#   td21      => time width for the SET pulse for PMOSFET
#   runsim    = 1, then create netlist for Spice simulation
# ----------------------------------------------------------------------------
    if q1 > 4.8:
        i1p = 109.0
        i1h = 121.0
        q1h = (q1 - 2.29)
        td21 = q1h/ i1h * 1000.0 - 12.0
        runsim = 1

# ----------------------------------------------------------------------------
#   Next NMOSFET determinations:
```

```
#   If Q2 is not < Q2prompt then
#       resulting currents will only be a single dbl-exp prompt current
# ---------------------------------------------------------------------------
        if q2 > 2.14:
            i2p = 102.0
            i2h = 118.0
            td22 = (q2 - 2.14) / 118.0 * 1000.0 - 12.0
        # End if
        else:
            i2p = q2 / 27 * 1000.0
            td22 = 0.0
        # End else
        print q1, q2, i1p, td21, i2p, td22
    # End if

# ---------------------------------------------------------------------------
#   Start with NMOSFET pre-characterized for:
#   Qthresh   = 4.6 fc
#   I2prompt  = 102.0 uA
#   I2hold    = 118.0 uA
#   tau1+tau2 = 12.0 ps
#   td2-td1   = 115ps-100ps = 15ps
#   Q2prompt  ~ I1prompt*(21)/1000.0
#   td22      => time width for the SET pulse for NMOSFET
#   runsim    = 1, then create netlist for Spice simulation
# ---------------------------------------------------------------------------
    elif q2 > 4.6:
        i2p = 102.0
        i2h = 118.0
        q2h = (q2 - 2.14)
        td22 = q2h / i2h * 1000.0 - 12.0
        runsim = 1

# ---------------------------------------------------------------------------
#   Next PMOSFET determinations:
#   If Q1 is not < Q1prompt then
#       resulting currents will only be a single dbl-exp prompt current
# ---------------------------------------------------------------------------
        if q1 > 2.29:
            i1p = 109.0
            i1h = 121.0
            td21 = (q1 - 2.29) / 121.0 * 1000.0 - 12.0
        # End if
        else:
            i1p = q1 / 27 *1000.0
            td21 = 0.0
        # End else
        print q1, q2, i1p, td21, i2p, td22
    # End elif
# ---------------------------------------------------------------------------
#   Next Create Boeing INV1 netlist:
#   If runsim is true then
#       populate a cricruittest netlist such that:
#       Create INV1 Subcircuit (that will create a chain of 65 + 1 INV1's
#       Create a GuardGate subcircuit to add a delay to allow for the
#           distiguishability of longer SETs
#       Create an Asynchronous Latch to determine if the SET reaches a
#           memory element and would be captured
# ---------------------------------------------------------------------------
    if runsim == 1:
        eldodeck = open('tmp/circuittest.cir', 'w')
        eldodeck.write('* Test circuit for Python\n\n')
        eldodeck.write(".lib '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./skew.file' stats\n")
        eldodeck.write(".inc '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./fixed_corner'\n")
        eldodeck.write(".inc '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./hspice.param'\n")
        eldodeck.write(".inc '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./nfet.inc'\n")
        eldodeck.write(".inc '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./pfet.inc'\n\n\n")
        eldodeck.write(".subckt INV1 in_inv1 out_inv1 vdd_inv1 vss_inv1\n")
        eldodeck.write("XPINV1 vdd_inv1 in_inv1 out_inv1 vdd_inv1 pfet w=840n l=80n ad=218f pd=2.2u\n")
        eldodeck.write("XNINV1 out_inv1 in_inv1 vss_inv1 vss_inv1 nfet w=280n l=80n ad=73f pd=1.1u\n")
        eldodeck.write(".ends INV1\n\n\n")
        eldodeck.write(".subckt NAND2 in1_nand2 in2_nand2 out_nand2 vdd_nand2 vss_nand2\n")
        eldodeck.write("XPNAND21 vdd_nand2 in1_nand2 out_nand2 vdd_nand2 pfet w=490n l=80n ad=79f pd=0.8u\n")
        eldodeck.write("XPNAND22 vdd_nand2 in2_nand2 out_nand2 vdd_nand2 pfet w=490n l=80n ad=79f pd=0.8u\n")
        eldodeck.write("XNNAND21 out_nand2 in2_nand2 n10 vss_nand2 nfet w=280n l=80n ad=104f pd=1.3u\n")
        eldodeck.write("XNNAND22 n10 in1_nand2 vss_nand2 vss_nand2 nfet w=280n l=80n ad=73f pd=1.1u\n")
        eldodeck.write(".ends NAND2\n\n\n")
        eldodeck.write(".subckt NOR2 in1 in2 out vdd vss\n")
        eldodeck.write("XP1 vdd in1 n10 vdd pfet w=1740n l=80n ad=922f pd=7.3u\n")
        eldodeck.write("XP2 n10 in2 out vdd pfet w=1740n l=80n ad=278f pd=2.4u\n")
        eldodeck.write("XN1 out in2 vss vss nfet w=280n l=80n ad=70f pd=0.8u\n")
        eldodeck.write("XN2 out in1 vss vss nfet w=280n l=80n ad=70f pd=0.8u\n")
        eldodeck.write(".ends NOR2\n\n\n")
        eldodeck.write(".subckt GGATE in1_gg in2_gg out_gg vdd_gg vss_gg\n")
        eldodeck.write("XPGG1 vdd_gg in2_gg ngg1 vdd_gg pfet w=1730n l=80n ad=751f pd=7.2u\n")
        eldodeck.write("XPGG2 ngg1 in1_gg out_gg vdd_gg pfet w=1730n l=80n ad=313f pd=2.5u\n")
        eldodeck.write("XNGG1 ngg2 in1_gg out_gg vss_gg nfet w=480n l=80n ad=197f pd=1.8u\n")
        eldodeck.write("XNGG2 vss_gg in2_gg ngg2 vss_gg nfet w=480n l=80n ad=134f pd=1.5u\n")
        eldodeck.write(".ends GGATE\n\n\n")
        eldodeck.write(".subckt ALATCH in1_al in2_al q_al vdd_al vss_al\n")
        eldodeck.write("X1 in1_al nal1 q_al vdd_al vss_al NAND2\n")
        eldodeck.write("X2 q_al in2_al nal1 vdd_al vss_al NAND2\n")
        eldodeck.write(".ends ALATCH\n\n\n")

# ---------------------------------------------------------------------------
#    Next add a counter:
#    Variables:
#        cnt =>    Counter for number of INV1's to populate for INV1 chain
#                  Add counter number to string for correct netlist creation
#        Add a Guard Gate Latch to add a delay to distinguish Long SETs
#        Add an Asynchronous Latch to determine if the SET reaches a
#            memory element and would be captured
#    Add Voltage sources, V1(vdd), V2(Input to chain), V3(vss) and a Reset (V4)
# ---------------------------------------------------------------------------
        cnt = 0
        while (cnt < 65):
          xl = 'XINV' + str(cnt) + ' n' + str(cnt) + ' n' + str(cnt+1) + ' vdd vss INV1\n'
          eldodeck.write(xl)
          cnt = cnt + 1
        # End While
        eldodeck.write("XINV_Last1 n65 n66 vdd vss INV1\n")
        eldodeck.write("XINV_Last2 n65 n67 vdd vss INV1\n")
        eldodeck.write("XGG1 n66 n67 n68 vdd vss GGATE\n")
        eldodeck.write("XAL n68 reset out vdd vss ALATCH\n")
        eldodeck.write("V1 vdd 0 0.9\n")
        eldodeck.write("V2 n0 0 0.0\n")
        eldodeck.write("V3 vss 0 0.0\n")
        eldodeck.write("V4 reset 0 pulse(0 0.9 25p 10p 10p 1m 2m)\n")

# ---------------------------------------------------------------------------
#    Next add SET dbl-exp Current Sources at random inputs to the INV1 chain:
#    Variables:
#        node          =>   random node where to instantiate dbl-exp current sources
#        setcurrsrc     => string to write the appropriate current source
# ---------------------------------------------------------------------------
        node = random.randint(1,65)
        setcurrsrc = "I1P vdd n" + str(node) + " exp(0u " + str(i1p) + "u 100p 2p 115p 4p)\n"
        eldodeck.write(setcurrsrc)

# ---------------------------------------------------------------------------
#    If the pulse-width for I1P is greater than 0 then
#        add a hold current source I1H
# ---------------------------------------------------------------------------
        if td21 > 0.0:
            setcurrsrc = "I1H vdd n" + str(node) + " exp(0u " + str(i1h) + "u 100p 2p " + str(td21+100.0) + "p 10p)\n"
            eldodeck.write(setcurrsrc)
        # End if
        setcurrsrc = "I2P n" + str(node) + " vss exp(0u " + str(i2p) + "u 100p 2p 115p 4p)\n"
```

```
        eldodeck.write(setcurrsrc)

# ---------------------------------------------------------------------------
#    If the pulse-width for I2P is greater than 0 then
#        add a hold current source I2H
# ---------------------------------------------------------------------------
        if td22 > 0.0:
            setcurrsrc = "I2H n" + str(node) + " vss exp(0u " + str(i2h) + "u 100p 2p " + str(td22+100.0) + "p 10p)\n"
            eldodeck.write(setcurrsrc)
        # End if

# ---------------------------------------------------------------------------
#    Simulate for 3ns in 1ps steps
#    Write results to output file
# ---------------------------------------------------------------------------
        eldodeck.write(".tran 1ps 3ns\n")
        eldodeck.write(".printfile tran v(n1) v(out) file=transient.out\n")
        printsrc = ".printfile tran v(n1) v(n" + str(node) +  ") v(n65) v(n66) v(n67) v(n68) v(out) file=nodetransient.out\n"
        eldodeck.write(printsrc)
        eldodeck.close()

# ---------------------------------------------------------------------------
# Run/Invoke circuittest.cir netlist in Eldo, use same OS command
# ---------------------------------------------------------------------------
        os.system('eldo -compat -mgls_async -noconf -i tmp/circuittest.cir')

# ---------------------------------------------------------------------------
# Open the newly created output file -transient.out- from resulting simulation
#    in /tmp directory
# Using variables:
#    eldoresult - pointer for transient.out file
#    erlines - Read all the lines of -transient.out-int an array of strings
#            erlines(1) is the first line, erlines(2) is the second and so on
#            Dynamically assigned the array based upon the length of the file
# Close the file
# ---------------------------------------------------------------------------
        eldoresult = open('tmp/transient.out', 'r')
        erlines = eldoresult.readlines()
        eldoresult.close()

# ---------------------------------------------------------------------------
# Determine if the transient was latched/observed at the output of the
#    Asynchronous Latch
#
#    Variables:
#        latchyes => default = 0  (not latched)
#        latchcnt => default = 0  Counter for number of SETs latched
# ---------------------------------------------------------------------------
        latchyes = 0

# ---------------------------------------------------------------------------
#    Loop through each line in the result file
#        find ('#')     - Ignores any line with a '#' (comments, not results)
#        strip()        - Remove carriage returns \n from the output lines
#        split()        - Split the line by a space between 3 numbers into
#                         strings a, b & c
#        float()        - Convert c into a real number and call it volts
# ---------------------------------------------------------------------------
        for line in erlines:
            if line.find('#')<0:
                line=line.strip()
                a,b,c =line.split(' ')
                volts=float(c)

# ---------------------------------------------------------------------------
#    Determine if the output voltage of the latch ever exceeds 0.8V
#        If voltage exceeds 0.8 volts, then SET is latched
# ---------------------------------------------------------------------------
                if volts > 0.8:
                    print volts
                    latchyes = 1
                # End if
            # End if
        #End for

# ---------------------------------------------------------------------------
#    Count the number of SETs that were latched
#        Output results to file "latchresults.out
# ---------------------------------------------------------------------------
        if latchyes ==1:
            latchcnt = latchcnt + 1
            nodeinfo = event +" "+ str(node) + " " + str(i1p) + " " + str(i2p) + " " + str(td21) + " " + str(td22) + "\n"
            latchsetresult.write(nodeinfo)
        # End if
        print latchyes
    #End if
# ---------------------------------------------------------------------------
# End Main (For) loop
# ---------------------------------------------------------------------------
print latchcnt
latchsetresult.close()
```

APPENDIX E

MRED2LOGIC PYTHON SCRIPT

```
# ---------------------------------------------------------------------------
# Filename:    mredtospicetologic_inv1set.py
# Version:     v1
# Description: Python Code for determining Threshold/Prompt/Hold Currents
#              Create/Simulate a SPICE netlist
# ---------------------------------------------------------------------------
# Author:      Dolores Black, Vanderbilt University
# History:
#   DAB  4-30-2011      -- First Version
#   DAB  5-08-2011      ~~ Modify for script to add analysis for charge
#                          efficiencies in the q1/q2 analysis "loops"
#   DAB  5-09-2011      ** Modified for second "guess" at SV's (4 N/PMOS)
#   DAB  5-10-2011      ^^ Modified efficiencies of Substrate, too much charge
#   DAB  6-3-2011      && Modified to report pulsewidths of SET & logic depth
#   DAB  6-6-2011      -* Modified to report an output file for the Histograms
#                          of the pulsewidths starting a 5ps and up.
# ---------------------------------------------------------------------------
# Naming Conventions:
#
# ---------------------------------------------------------------------------
# Full Description:
# ---------------------------------------------------------------------------
#   Python script to:
#       1. Parse MRED charge collection (q1-q6, etc) output results
#          file given the ions, sensitive volumes, etc.
#       2. Extract the relevant data
#            Only operate on charge greater than the threshold necessary
#            to cause enough current to cause the voltage to cross the
#            circuit/cell's rail
#       3. Determine the IxP/H values resulting from charge converted
#            to current that exceeds the threshold
#            a. Calculate/Determine the pulse-width (td2) resulting from
#               excess charge greater than the minimum ihold current
#       4. Create a circuit netlist using IxP/H, td2 given the default dbl-
#            exp tau1, tau2 and td1 (2ps, 4ps(prompt)/10ps(hold), 100ps
#       5. Execute Spice on netlist
#            a. Output resulting pulse-widths and/or upset (output of latch)
# ---------------------------------------------------------------------------
# Boeing INV1 MRED results Parser determination
# ---------------------------------------------------------------------------
#   Q1 is charge collection on the output of the PMOSFET drain. Threshold
#   current is defined as the prompt current needed to drive the transient
#   voltage to the rail, but may not be necessarily the minimum to propagate
#   a transient signal. Hold current is defined as the current needed to
#   maintain the voltage at the rail once it's at the rail.
#
#   Q2 is charge collection on the output of the NMOSFET drain. Threshold
#   current is defined as the prompt current needed to drive the transient
#   voltage to the rail, but may not be necessarily the minimum to propagate
#   a transient signal. Hold current is defined as the current needed to
#   maintain the voltage at the rail once it's at the rail.
# ---------------------------------------------------------------------------


# ---------------------------------------------------------------------------
# import os is a command that allows the running of command line functions
# in Python
# import random is a command that allows for the import of random numbers to
#   be used for the placement of the dbl-exp SET current sources in circuit
# ---------------------------------------------------------------------------
```

```
import os
import random

# ---------------------------------------------------------------------------
# Initialize a pwhist array of dimension 100. The line below expands a 1x2 array
#    to a 1x100 array. pwhist[0] will be defined as number of pulses below 10 ps.
#    pwhist[1] will be pulses between 10 and 20 ps...
# ---------------------------------------------------------------------------
pwhist = [0, 0] * 50

# ---------------------------------------------------------------------------
# Changed the file name to reflect the pulsewidth output added at the "end".
# ---------------------------------------------------------------------------
latchsetresult = open('latchresults2LETpw_v1_3.out', 'w')
latchsetresult.write("MRED#  node#  pw\n")

# ---------------------------------------------------------------------------
# Open the MRED output results file, read it and close it
# in Python
# ---------------------------------------------------------------------------
mredresult = open('mredresults/q0_q9boeinginvnsv2LET.out', 'r')
mredlines = mredresult.readlines()
mredresult.close()

# ---------------------------------------------------------------------------
# Main For loop to parse output file and calculate resulting charges and
#    convert to associated currents and the pulse-width variable td2
#    Variables:
#        line      => loop variable
#        event,c-h,
#            i-k => column in the output file to be read
#        q1       => charge collection output on PMOSFET from MRED
#                    q1thresh converted from i1thresh
#                      determined from cell characterization
#        q2       => charge collection output on NMOSFET from MRED
#                    q2prompt converted from i2prompt
#                      determined from cell characterization
#        q1h/q2h   => q(MRED result) - qprompt (pre-characterized)
#                    q1(2)prompt = 2.29fc/2.14fc converted from iprompt
#                    determined from cell characterization
#        td21      => SET pulse-width variable for PMOSFET
#        td22      => SET pulse-width variable for NMOSFET
#        runsim    => If true (1) then create Spice netlist to run, otherwise
#                    go to next Q value for evaluation
# ---------------------------------------------------------------------------
for line in mredlines:
    line=line.strip()
    event,b,c,d,e,f,g,h,i,j,k=line.split(',')
    i1p = 0.0
    i1h = 0.0
    td21 = 0.0
    i2p = 0.0
    i2h = 0.0
    td22 = 0.0
    q1=float(c)*0.6 + float(d)*0.2 + float(e)*0.70 + float(f)*0.10
    q2=float(g)*0.75 + float(h)*0.25 + float(i)*0.55 + float(j)*0.10
    runsim=0

# ---------------------------------------------------------------------------
# Start with PMOSFET pre-characterized for:
#    Qthresh   = 4.8 fc
#    I1prompt  = 109.0 uA
#    I1hold    = 121.0 uA
#    tau1+tau2 = 12.0 ps
#    td2-td1   = 115ps-100ps = 15ps
#    Q1prompt  ~ I1prompt*(21)/1000.0
#    td21      => time width for the SET pulse for PMOSFET
#    runsim    = 1, then create netlist for Spice simulation
# ---------------------------------------------------------------------------
```

```
    if q1 > 4.8:
        i1p = 109.0
        i1h = 121.0
        q1h = (q1 - 2.29)
        td21 = q1h/ i1h * 1000.0 - 12.0
        runsim = 1

# ---------------------------------------------------------------------------
#   Next NMOSFET determinations:
#   If Q2 is not < Q2prompt then
#       resulting currents will only be a single dbl-exp prompt current
# ---------------------------------------------------------------------------
        if q2 > 2.14:
            i2p = 102.0
            i2h = 118.0
            td22 = (q2 - 2.14) / 118.0 * 1000.0 - 12.0
        # End if
        else:
            i2p = q2 / 27 * 1000.0
            td22 = 0.0
        # End else
        print q1, q2, i1p, td21, i2p, td22
    # End if

# ---------------------------------------------------------------------------
#   Start with NMOSFET pre-characterized for:
#   Qthresh   = 4.6 fc
#   I2prompt  = 102.0 uA
#   I2hold    = 118.0 uA
#   tau1+tau2 = 12.0 ps
#   td2-td1   = 115ps-100ps = 15ps
#   Q2prompt  ~ I1prompt*(21)/1000.0
#   td22      => time width for the SET pulse for NMOSFET
#   runsim    = 1, then create netlist for Spice simulation
# ---------------------------------------------------------------------------
    elif q2 > 4.6:
        i2p = 102.0
        i2h = 118.0
        q2h = (q2 - 2.14)
        td22 = q2h / i2h * 1000.0 - 12.0
        runsim = 1

# ---------------------------------------------------------------------------
#   Next PMOSFET determinations:
#   If Q1 is not < Q1prompt then
#       resulting currents will only be a single dbl-exp prompt current
# ---------------------------------------------------------------------------
        if q1 > 2.29:
            i1p = 109.0
            i1h = 121.0
            td21 = (q1 - 2.29) / 121.0 * 1000.0 - 12.0
        # End if
        else:
            i1p = q1 / 27 *1000.0
            td21 = 0.0
        # End else
        print q1, q2, i1p, td21, i2p, td22
    # End elif
# ---------------------------------------------------------------------------
#   Next Create Boeing INV1 netlist:
#   If runsim is true then
#       populate a cricruittest netlist such that:
#       Create INV1 Subcircuit (that will create a chain of 65 + 1 INV1's
#       Create a GuardGate subcircuit to add a delay to allow for the
#           distiguishability of longer SETs
#       Create an Asynchronous Latch to determine if the SET reaches a
#           memory element and would be captured
# ---------------------------------------------------------------------------
    if runsim == 1:
```

```
    eldodeck = open('tmp/circuittest.cir', 'w')
    eldodeck.write('* Test circuit for Python\n\n')
    eldodeck.write(".lib '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./skew.file' stats\n")
    eldodeck.write(".inc '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./fixed_corner'\n")
    eldodeck.write(".inc '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./hspice.param'\n")
    eldodeck.write(".inc '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./nfet.inc'\n")
    eldodeck.write(".inc '/usr/local/isde/PDK/IBM_PDK/cmos9sf/relIBM/HSPICE/models/./pfet.inc'\n\n\n")
    eldodeck.write(".subckt INV1 in_inv1 out_inv1 vdd_inv1 vss_inv1\n")
    eldodeck.write("XPINV1 vdd_inv1 in_inv1 out_inv1 vdd_inv1 pfet w=840n l=80n ad=218f pd=2.2u\n")
    eldodeck.write("XNINV1 out_inv1 in_inv1 vss_inv1 vss_inv1 nfet w=280n l=80n ad=73f pd=1.1u\n")
    eldodeck.write(".ends INV1\n\n\n")
    eldodeck.write(".subckt NAND2 in1_nand2 in2_nand2 out_nand2 vdd_nand2 vss_nand2\n")
    eldodeck.write("XPNAND21 vdd_nand2 in1_nand2 out_nand2 vdd_nand2 pfet w=490n l=80n ad=79f pd=0.8u\n")
    eldodeck.write("XPNAND22 vdd_nand2 in2_nand2 out_nand2 vdd_nand2 pfet w=490n l=80n ad=79f pd=0.8u\n")
    eldodeck.write("XNNAND21 out_nand2 in2_nand2 n10 vss_nand2 nfet w=280n l=80n ad=104f pd=1.3u\n")
    eldodeck.write("XNNAND22 n10 in1_nand2 vss_nand2 vss_nand2 nfet w=280n l=80n ad=73f pd=1.1u\n")
    eldodeck.write(".ends NAND2\n\n\n")
    eldodeck.write(".subckt NOR2 in1 in2 out vdd vss\n")
    eldodeck.write("XP1 vdd in1 n10 vdd pfet w=1740n l=80n ad=922f pd=7.3u\n")
    eldodeck.write("XP2 n10 in2 out vdd pfet w=1740n l=80n ad=278f pd=2.4u\n")
    eldodeck.write("XN1 out in2 vss vss nfet w=280n l=80n ad=70f pd=0.8u\n")
    eldodeck.write("XN2 out in1 vss vss nfet w=280n l=80n ad=70f pd=0.8u\n")
    eldodeck.write(".ends NOR2\n\n\n")
    eldodeck.write(".subckt GGATE in1_gg in2_gg out_gg vdd_gg vss_gg\n")
    eldodeck.write("XPGG1 vdd_gg in2_gg ngg1 vdd_gg pfet w=1730n l=80n ad=751f pd=7.2u\n")
    eldodeck.write("XPGG2 ngg1 in1_gg out_gg vdd_gg pfet w=1730n l=80n ad=313f pd=2.5u\n")
    eldodeck.write("XNGG1 ngg2 in1_gg out_gg vss_gg nfet w=480n l=80n ad=197f pd=1.8u\n")
    eldodeck.write("XNGG2 vss_gg in2_gg ngg2 vss_gg nfet w=480n l=80n ad=134f pd=1.5u\n")
    eldodeck.write(".ends GGATE\n\n\n")
    eldodeck.write(".subckt ALATCH in1_al in2_al q_al vdd_al vss_al\n")
    eldodeck.write("X1 in1_al nal1 q_al vdd_al vss_al NAND2\n")
    eldodeck.write("X2 q_al in2_al nal1 vdd_al vss_al NAND2\n")
    eldodeck.write(".ends ALATCH\n\n\n")

# --------------------------------------------------------------------------
#   Next add a counter:
#   Variables:
#       cnt =>    Counter for number of INV1's to populate for INV1 chain
#                 Add counter number to string for correct netlist creation
#       Add a Guard Gate Latch to add a delay to distinguish Long SETs
#       Add an Asynchronous Latch to determine if the SET reaches a
#           memory element and would be captured
#   Add Voltage sources, V1(vdd), V2(Input to chain), V3(vss) and a Reset (V4)
# --------------------------------------------------------------------------
    cnt = 0
    while (cnt < 65):
      xl = 'XINV' + str(cnt) + ' n' + str(cnt) + ' n' + str(cnt+1) + ' vdd vss INV1\n'
      eldodeck.write(xl)
      cnt = cnt + 1
    # End While
# --------------------------------------------------------------------------
# The only modification I made to the circuit is to delete the ggate and the load
# inverters, but the chain drive two inverters at the end which then go into the
# latch. The pulsewidth is currently measured at the last part of the chain, which
# is also the input to the latch. The latch state does not matter.
# --------------------------------------------------------------------------
#       eldodeck.write("XINV_Last1 n65 n66 vdd vss INV1\n")
#       eldodeck.write("XINV_Last2 n65 n67 vdd vss INV1\n")
#       eldodeck.write("XGG1 n66 n67 n68 vdd vss GGATE\n")
# --------------------------------------------------------------------------
    eldodeck.write("XAL n65 vss out vdd vss ALATCH\n")
    eldodeck.write("V1 vdd 0 0.9\n")
    eldodeck.write("V2 n0 0 0.0\n")
    eldodeck.write("V3 vss 0 0.0\n")
# --------------------------------------------------------------------------
# No longer need a reset for the latch
# --------------------------------------------------------------------------
#       eldodeck.write("V4 reset 0 pulse(0 0.9 25p 10p 10p 1m 2m)\n")
# --------------------------------------------------------------------------
# ---------------------------------------------------------------------------
#    Next add SET dbl-exp Current Sources at random inputs to the INV1 chain:
#    Variables:
#        node          =>   random node where to instantiate dbl-exp current sources
#        setcurrsrc     => string to write the appropriate current source
#
# ---------------------------------------------------------------------------
# You can leave the circuit as is above and just change the depth by choosing
# the node in this next statement. If depth is 10, then the range is (56, 65)
# and it will only inject the pulse in the last 10 cells in the chain. It
# writes some extra circuits now, but that won't slow it down much at all.
# This just keeps node 65 at the measurement node.
# ---------------------------------------------------------------------------
        node = random.randint(63,65)
        setcurrsrc = "I1P vdd n" + str(node) + " exp(0u " + str(i1p) + "u 100p 2p 115p 4p)\n"
        eldodeck.write(setcurrsrc)

# ---------------------------------------------------------------------------
#    If the pulse-width for I1P is greater than 0 then
#        add a hold current source I1H
# ---------------------------------------------------------------------------
        if td21 > 0.0:
            setcurrsrc = "I1H vdd n" + str(node) + " exp(0u " + str(i1h) + "u 100p 2p " + str(td21+100.0) + "p 10p)\n"
            eldodeck.write(setcurrsrc)
        # End if
        setcurrsrc = "I2P n" + str(node) + " vss exp(0u " + str(i2p) + "u 100p 2p 115p 4p)\n"
        eldodeck.write(setcurrsrc)

# ---------------------------------------------------------------------------
#    If the pulse-width for I2P is greater than 0 then
#        add a hold current source I2H
# ---------------------------------------------------------------------------
        if td22 > 0.0:
            setcurrsrc = "I2H n" + str(node) + " vss exp(0u " + str(i2h) + "u 100p 2p " + str(td22+100.0) + "p 10p)\n"
            eldodeck.write(setcurrsrc)
        # End if

# ---------------------------------------------------------------------------
#    Simulate for 3ns in 1ps steps
#    Write results to output file
# ---------------------------------------------------------------------------
        eldodeck.write(".tran 1ps 3ns\n")

# ---------------------------------------------------------------------------
# You do not need these files written anymore, but you could still output
# them if you want.
# ---------------------------------------------------------------------------
#         eldodeck.write(".printfile tran v(n1) v(out) file=transient.out\n")
#         printsrc = ".printfile tran v(n1) v(n" + str(node) +  ") v(n65) v(n66) v(n67) v(n68) v(out) file=nodetransient.out\n"
#         eldodeck.write(printsrc)
# ---------------------------------------------------------------------------
        eldodeck.write('.extract tran label=crossmid-1 tcross(v(n65), vth=0.45, occur=1)\n')
        eldodeck.write('.extract tran label=crossmid-2 tcross(v(n65), vth=0.45, occur=2)\n')
        eldodeck.close()

# ---------------------------------------------------------------------------
# Run/Invoke circuittest.cir netlist in Eldo, use same OS command
# ---------------------------------------------------------------------------
        os.system('eldo -compat -mgls_async -noconf -i tmp/circuittest.cir')

# ---------------------------------------------------------------------------
# Open the newly created output file -circuittest.chi- from resulting simulation
#    in /tmp directory
# Using variables:
#    eldoresult - pointer for circuittest.chi file
#    erlines - Read all the lines of -circuittest.chi-into an array of strings
#            erlines(1) is the first line, erlines(2) is the second and so on
#            Dynamically assigned the array based upon the length of the file
# Close the file
# ---------------------------------------------------------------------------
        eldoresult = open('tmp/circuittest.chi', 'r')
        erlines = eldoresult.readlines()
        eldoresult.close()

# ---------------------------------------------------------------------------
# Determine the pulsewidth at the input to the
#   Asynchronous Latch
#
#   Variables:
#       pulsewidth => Full width, half max voltage signal
# ---------------------------------------------------------------------------

# ---------------------------------------------------------------------------
#   Loop through each line in the result file
#       find ('cross') - Find the lines with the pulsewidth extraction
#       find ('-1 =')  - Only choose the lines with the actual data
#       strip()        - Remove carriage returns \n from the output lines
#       split()        - Split the line by a space between 2 numbers into
#                        strings a & b
#       float()        - Convert b into a real number and call it time
# ---------------------------------------------------------------------------
        pulsewidth = 0.0
        start = 0.0
        for line in erlines:
            if line.find('crossmid') >= 0:
                if line.find('-1 =') >= 0:
                    line = line.strip()
                    a,b = line.split('=')
                    start = float(b)
                # End if
                if line.find('-2 =') >= 0:
                    line = line.strip()
                    a,b = line.split('=')
                    pulsewidth = float(b) - start
                # End if
            # End if
        #End for

# ---------------------------------------------------------------------------
#       Check to see if the output SET pulsewidth is > 0, then
#       Output results to file "latchresults~LETpw_v1.out
#       Multiply the pulsewidth by 1E11 and then convert that number to an
#         integer to determine the histogram index. This will make each bin
#         of the histogram equal to 10 ps, the first being 0-10 ps, then
#         second being 10-20 ps, and the last to be 990-1000 ps.
#       Increment the count of pulses in the histogram bin
# ---------------------------------------------------------------------------
        if pulsewidth > 0.0:
            nodeinfo = event +" "+ str(node) + " " + str(pulsewidth) + "\n"
            latchsetresult.write(nodeinfo)
            histindex = int(pulsewidth * 100000000000)
            pwhist[histindex] = pwhist[histindex] + 1
        #End if
    #End if
# ---------------------------------------------------------------------------
# End Main (For) loop
# ---------------------------------------------------------------------------
latchsetresult.close()

# ---------------------------------------------------------------------------
# Write histogram results to a separate file. Loop through each element in the
#   histogram array and output the count in each bin and then a carriage return.
#   Close the file and the end of the writing
# ---------------------------------------------------------------------------
pwhistresult = open('pwhist2LET_v1_3.out', 'w')
for n in range(0, 100):
    pwhistresult.write(str(pwhist[n]))
    pwhistresult.write('\n')
#End for
pwhistresult.close()
```

APPENDIX F

ALU TESTBENCH CODE

```
//
// Module: tb_set_test_structure_8bit_core
//
// Description: test bench of the 8bit_core
//
// Author:  DL
//
// Last created:  4/24/2008

// SET signal added to testbench 04/01/09
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Modified by AS, 3/2011
//
//-----------------------------------------
// Modified by DAB, 5/2011
//
// Added BS, fault injection library
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

module tb_set_test_structure_8bit_core;
  parameter PERIOD = 10;
  parameter BUS_WIDTH = 8;
  real se_delay;
  real end_delay;
  real range_end;
  real se_len;
  reg [BUS_WIDTH-1:0] a_in, b_in, a_in_1, b_in_1, a_in_2, b_in_2;
  reg [2:0] func_in, func_in_1, func_in_2;
  reg reset_in, clk_in;


        wire [BUS_WIDTH*2-1:0] y_out;
        reg [BUS_WIDTH*2-1:0] compare;
        set_test_structure_8bit_core set_test_structure_8bit_core(
        .a_in(a_in),
        .b_in(b_in),
        .func_in(func_in),
        .reset_in(reset_in),
        .clk_in(clk_in),
        .y_out(y_out)
        );

        initial begin
                clk_in = 1'b0;
                reset_in = 1'b0;
                #(PERIOD);
                #(PERIOD);
                reset_in = 1'b1;
        end

        initial begin
                forever #(PERIOD/2) clk_in = ~clk_in;
        end
```

```
    //SET Loop
    initial begin
      $singleEventInit();
//     $pseudoRandomSeed(seed);
//     $display ("SET SEED TO %d",seed);
      se_delay=($pseudoRandom(range_end*10*1000)/1000.0);
     end_delay=se_delay+10.0;
     #se_delay begin

    $singleEventTransient(tb_set_test_structure_8bit_core.set_test_structure_8bit_core,se_len*1000.0);
        $display("UPSET %16d, %16d, %16d, %16d, %16d, %16d, %16f", $time,
$stime,$realtime,func_in,a_in,b_in,se_len);
     end
     #end_delay $stop;
    end


    always @* begin
        case (func_in_2)
                //nothing
                3'b000:
                begin
                    compare <= 16'd0;
                end
                //add
                3'b001:
                begin
                    compare <= a_in_2 + b_in_2;
                end
                //sub
                3'b010:
                begin
                    compare <= a_in_2 - b_in_2;
                end
                //multi
                3'b011:
                begin
                    compare <= a_in_2 * b_in_2;
                end
                //div
                3'b100:
                begin
                    compare <= a_in_2 / b_in_2;
                end
                //comp

                3'b101:
                begin
                    if(a_in_2 > b_in_2) begin
                        compare <= 16'd2;
                    end else if (a_in_2 < b_in_2) begin
                        compare <= 16'd1;
                    end else if (a_in_2 == b_in_2) begin
                        compare <= 16'd0;
                    end else  begin
                        compare <= 16'd3;
                    end
                end
                //a
```

```
                                3'b110:
                                begin
                                        compare <= a_in_2;
                                end
                                //b
                        3'b111:
                        begin
                                compare <= b_in_2;
                        end
                endcase
        end

        //this basically delay the inputs for 2 cycle, and the compare to the output of the core
        //it first loops through b, from 127 to 0, then a, from 0 to 127
        //then it goes from func from 0 to 7
        // if there is a mis compare, it will print out the error
        always @ (negedge clk_in) begin
                for(func_in = 0; func_in < 8; func_in = func_in + 1) begin
                        for(a_in = 0; a_in < 127; a_in = a_in + 1) begin
                                for(b_in = 127; b_in != 0; b_in = b_in -1) begin
                                        a_in_1 <= a_in;
                                        a_in_2 <= a_in_1;
                                        b_in_1 <= b_in;
                                        b_in_2 <= b_in_1;
                                        func_in_1 <= func_in;
                                        func_in_2 <= func_in_1;
                                        //$display("TIME %16d, %16d, %16d, %16d, %16d, %16d",
$time, $stime,$realtime,func_in,a_in,b_in);
                                        if (y_out != compare) begin
                                                $display("ERROR %16b, %16b, %16d, %16d, %16d,
%16d, %16d, %16f", y_out, compare,func_in,a_in,b_in, y_out, compare,se_len);
                                                $stop;
                                        end
                                        #(PERIOD);
                                end
                        end
                end
                $finish;
        end

endmodule;
```