ABSTRACT

Title of Document: STUDY OF THE IMPACT OF HARDWARE FAILURES ON SOFTWARE RELIABILITY

Bing Huang, Doctor of Philosophy, 2006

Directed By: Professor Joseph B. Bernstein
Professor Carol S. Smidts
Reliability Engineering

Software plays an increasingly important role in modern safety-critical systems. Reliable software becomes desirable for all stakeholders. Typical software related failures include software internal failures, input failures, output failures, support failures and multiple interaction failures. This dissertation provides a methodology to study the impact of hardware support failures on software reliability.

The hardware failures we are focusing on in this study are semiconductor device intrinsic failures that are directly related to software execution during device operation. The software execution on hardware devices, in essence, is a series of 0 and 1 signal alternations for the inputs of hardware components. Such signal alternations lead to voltage changes and current flows in the microelectronic hardware device, which serve as electrical stresses on the device and may lead to physical failures. The failure mechanisms include Hot Carrier Injection (HCI), Electromigration (EM), and Time Dependent Dielectric Breakdown (TDDB). During

device operation such hardware failures could propagate to circuit level in the form of signal delays, changes of circuit functionality, and signals stuck at a logic value (0 or 1), which could further propagate into the software layer and affect the reliability of the software.

The proposed methodology is divided into three parts: (i) analysis of the manifestations of permanent failures on circuit elements (logic gates, flip-flops, etc.), (ii) development of reliability models for the circuit elements as functions of the software execution, and (iii) calculation of failure probability distributions of the hardware circuit elements under the software execution.

The methodology is applied to a comprehensive case study, targeting all the CPU registers and ALU logic gates of a computer system based on the Z80 microprocessor. About 120 different types of failure manifestations are observed, and more than 250 reliability models for the different types of failure manifestations and circuit elements are developed. Such models allow us to calculate the failure probability distributions of the CPU registers and ALU gates of the Z80 computer system under the software execution. We also extend the methodology and the case study to the consideration of transient failures, also known as Single Event Upsets (SEUs).

STUDY OF THE IMPACT OF HARDWARE FAILURES ON SOFTWARE RELIABILITY

By

Bing Huang

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2006

Advisory Committee:
Professor Joseph B. Bernstein, Co-Chair/Advisor
Professor Carol S. Smidts, Co-Chair/Advisor
Professor Mohammad Modarres
Professor David L. Akin
Professor Patrick F. McCluskey

Dedication

This dissertation is dedicated to my beautiful wife Hui. Her endless support facilitated my work so much during the past six years. Especially in the final stage of my PhD study, while taking care of our newborn son, she made great efforts to make my research as smooth as possible.

I would also like to thank our son Raine, who entered the world while I worked on this dissertation. His smile makes our life so much nicer.

Acknowledgements

I am grateful for the assistance of so many people in producing this work.

First of all, I would like to thank Dr. Bernstein and Dr. Smidts for their support and guidance during my graduate studies. Their knowledge and wisdom guided me to establish the framework of this dissertation work. Their patience with me when I was struggling with some technical problems gave me great relief. They also taught me a lot of things about everyday life during the past several years.

I would like to thank Dr. Manuel Rodriguez for his inspiring discussions, which led to some important mathematical models in this work. He also worked with me on many details of this dissertation.

I would also like to thank Dr. Ming Li and Dr. Xiaojun Li for their help during the early stages of this work, and their insightful comments, which were invaluable for the completion of this dissertation.

Furthermore, I would like to thank other talented researchers in our group, which include Jin Qin, Xiaohu Zhang, Hu Huang, Yuan Wei, and Dongfeng Zhu, for their help and support to this work.

Finally, I would like to thank Dr. Mohammad Modarres, Dr. David Akin, and Dr. Patrick McCluskey for serving on my dissertation committee.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

1.1 Statement of Problem

As human technologies advance, software plays an increasingly important role in almost all systems (military, communication, transportation, space, energy, etc), and the development of software and systems that are safe and reliable becomes crucial. Ignoring software risks can lead to catastrophic consequences. About 430 people died in eight fatal accidents (1985-2003) where software was the root cause. A study by the FAA revealed that 40% of the problems in large aircrafts (1984-1994) were software related. In NASA, software has become a major risk factor in space missions and projects. Software failures account for a large percentage of problem reports for NASA projects. For one spacecraft, 33.9% of the total number of failures found during ground testing was software related. This rate was higher than any other category. Other missions, such as Magellan launched in 1989 and Voyager in 1977, experienced software failures as 19–20% of all failures. Other examples of well-known spacecraft accidents provoked by software malfunction include: the explosion of the Ariane 5 launcher on its maiden flight in 1996, the loss of the Mars Climate Orbiter in 1999, the destruction of the Mars Polar Lander in 2000, the placing of a Milstar satellite in an incorrect and unusable orbit in 1999, and the loss of contact with the Solar Heliospheric Observatory (SOHO) spacecraft in 1998 [1]. The use of reliability methods and techniques can help significantly reduce the risk of these kinds of disasters.

Probabilistic Risk Assessment (PRA) is today the most significant and extended technique from the reliability domain. It aims at assessing, predicting and reducing the risk of failures in large technological systems such as nuclear power plants, chemical plants and aerospace systems. In NASA, PRA is required for all manned missions as well as for all missions with nuclear payload or nuclear fuel (e.g., such as the Crew Exploration Vehicle and other Moon–Mars–Beyond missions). However, current practice in PRA systematically neglects the contribution of software to the risk of failure of the system. The classical PRA methodology accounts for hardware and human interventions but does not account for software. In certain domains (such as nuclear), software has been confined to some extent to non-safety related functions. There also exists a reticence in the software safety community to use quantitative estimates of software failures. It stems essentially from the fact that software is still a new artifact. However, more and more vital functions, which in the past were controlled by human operators or hardware components, are today implemented and controlled by software. Thus, traditional PRA techniques are no longer valid and need to be improved and extended to include software reliability. Ignoring the contribution of software to system risk can lead to catastrophic consequences, such as the examples described above.

Recently, important efforts have been undertaken to incorporate software risks into the PRA technique [2, 3]. The work reported in [3] proposes a taxonomy of software failures as a first step to integrate software risks into PRA. The software is seen as an essential component of a system, which interacts with its environment through input and output interfaces. The software being executed on a computer

platform will take inputs from other subsystems (either software or hardware or humans) and produce outputs that will be used by either humans, other software or hardware. According to this view, a distinction is made between failures occurring within the software component itself and failures occurring due to the interaction between the software and its environment. This leads to the distinction of several software-related failure modes: internal failures, input failures, output failures, support failures and multiple interaction failures. Internal failures are due to the presence of "bugs" within the software code. Input failures are those out-of-bound values sent to the software that may drive correct software to provide incorrect outputs. Output failures are actually the set of out-of-bound software output behaviors that are neither due to out-of-bound input behavior nor due to internal software malfunction. These are failures that occur because of inconsistencies between specifications of the software and its downstream component. Multiple interaction failures are related to synchronization/communication problems between software processes and other system processes (software, hardware or human processes) that execute concurrently. Support failures are those software failures induced by malfunctions in the hardware support platform that the software resides on. They include failures due to competition for computing resources (deadlock, lockout) and computer platform physical failures (CPU failures, memory failures and I/O devices failures). The research work proposed in this dissertation is one component of the current work being done to characterize of this latter type of failure, i.e. computer platform physical failures, also referred to as *hardware failures*.

That hardware failures may lead to abnormal software behavior has been long recognized. The very first bug report described a bug-related hardware failure (On September 9, 1945, when Mark II, the Aiken Relay Calculator was experiencing problems, an investigation showed that there was a moth trapped between the points of Relay #70) [4]. More recently Iyer and Velardi [5] discovered that 35 percent of software failures in MVS systems were determined to be hardware-failure-related. Fault injection techniques have emerged as the major means to study the hardware failure phenomenon and its impact on the system [6]. Fault injection techniques, as a supplement to traditional software testing, are ideal for revealing the software and system's behaviors under abnormal hardware conditions, which may not be able to be addressed by their counterparts (such as software testing). Fault injection aims at assessing the system behavior in the presence of faults [7-9]. A key concern related to fault injection is the representativeness of the injected faults, i.e., the plausibility of the fault model with respect to actual faults [10]. In the majority of published works, the fault location, the fault type, and the time at which the fault is injected are randomly selected [7, 11]. Such a fault injection profile does not represent the actual hardware conditions in system operation. Therefore software reliability due to hardware support failures cannot be credibly assessed.

The impact of hardware failure on the software and on the system has attracted substantial attention in the field:

Kumar et al. introduced a simulation-based software model for application-specific dependability analysis of a system [12]. The model represents an application program by decomposing it into a graph model consisting of a set of nodes, a set of

edges and a mapping of the nodes to memory. The model simulates the execution of the program while faults are injected into the program's memory space. The authors claim that all hardware-related faults can be mapped as memory faults, but do not prove this formally. In addition, the authors do not clearly explain the types of faults injected, and how the injection is accomplished.

Todd Delong et al conducted a similar study to estimate the dependability parameters of computer systems [13]. The system hardware was modeled using a VHDL script. Stuck-at-0 and stuck-at-1 faults were injected into all the visible (programmable) processor registers. This work did not cover bit flipping or transient hardware failures. Nor did it provide the location and frequency distribution of stuck-at-0 and stuck-at-1 faults.

Amendola et al conducted a study to investigate the fault behavior of a microprocessor system [14]. They studied the faults located in memory, internal registers of the processor, and busses. Single bit-flipping faults were introduced into the VHDL system model of busses, memory, and CPU. This work demonstrated whether the system could tolerate faults, but could not provide reliability information due to the lack of fault location and distribution information. Choi et al [15] conducted a similar study with identical drawbacks.

D. Gil et al performed a fault injection experiment to analyze the "error syndrome" of a microcomputer system [16]. The system under study was described using VHDL. His experiment considered different types of faults with different durations and injected them into different locations of the microcomputer system.

However there was no prior knowledge of fault frequency or the distribution of fault locations and types.

In these approaches, fault injection is used as an accelerated testing technique to speed up the occurrence of errors and failures (e.g., likely and unlikely faults are given the same probability over location and time), and so do not capture the actual environmental conditions leading to the production of the faults impacting the software. The set of real faults may not be completely covered by the injected faults. More important, reliability estimates on software failures related to a particular operational situation of the system cannot be obtained (e.g., probability that the software of a spacecraft fails due to faults in the microelectronics devices when it will be traveling to Mars following a particular trajectory). In order to use the fault injection technique to calculate reliability estimates of software failures due to hardware malfunction, it is first necessary to characterize the physical operating conditions leading to a hardware malfunction from a probabilistic viewpoint. This problem is related to the development of the so-called *software operational profiles*.

The operational profile (OP) is a quantitative characterization of the way in which a system will be used [17]. It associates a set of probabilities to the system input space, and therefore characterizes the input stimuli of the system in operation. The determination of the OP can help guide managerial and engineering decisions throughout the whole software development lifecycle [17]. For instance, the OP can assist in the allocation of resources and optimization of reviews and code inspections and act as a guideline for software testing. The OP of a system is also a major deciding factor in assessing its reliability. The OP is used to measure software

reliability by testing the software in a manner such that the OP represents the system's actual use. It is also used to quantify the propagation of defects (or unreliability) through finite state machine models [2, 3]. However, determining the OP of a system is a difficult part of software reliability assessment in general [18]. The OP is traditionally built by enumerating field inputs and evaluating their occurrence frequencies. Musa pioneered a five-step approach to develop the OP [17]. His approach is based on collecting information on customers and users, identifying the system modes, determining the functional profile and recording the input states and their associated occurrence probabilities experienced in field operation. Expert opinion is normally used to estimate the hardware components-related operational profile due to the lack of field data. Musa's approach has been widely utilized and adapted in the literature to generate the operational profile. Some of these applications are summarized hereafter:

Chruscielski and Tian applied Musa's approach to a Lockheed Martin Tactical Aircraft System's cartridge support system [19]. User surveys which were generated in the format of a Questionnaire for User Interface Satisfaction, were used instead of the field data.

Elbaum and Narla refined Musa's approach by addressing heterogeneous user groups. They discovered that a single operational profile only "averages" the usage and "obscures" the real information about the operational probabilities [20]. They utilized clustering to identify groups of similar customers.

Gittens et al proposed an extended OP model composed of the process profile, structural profile and data profile. The process profile addresses the processes and

associated frequencies. The structural profile accounts for the system structure, the configuration or structure of the actual application, while the data profile covers the inputs to the application from different users [21].

Different values of the environment inputs will have major effects on processing. So Musa's [17] recommended approach for identifying the environmental variables is to have several experienced system design engineers identify them by brainstorming a list of those variables that might necessitate the program to respond in different ways. Furthermore, Sandfoss [22] suggests that the estimation of occurrence probabilities could be based on figures obtained from project documentation, engineering, judgment and system development experience. According to Gittens [21], a specific operational profile should include all users and all operating conditions that can affect the system.

Musa's approach and other extended approaches all require either field data or historic usage data. They all use an assumption that field data or historic usage data cover the entire input domain. This assumption is not always true and their approaches are not always successful simply because some input data may not be available, especially for safety critical systems. At least two reasons lead to the unavailability of the entire input data spectrum. First, the system may not be widely used (e.g., a reactor control system of a nuclear power plant). Therefore, very little field and historic usage data can be obtained. Second, the field data does not cover the entire spectrum of the input domain because some conditions may be extremely rare (e.g., unexpected inputs such as hardware failures). Further, many inputs may not be visible (e.g., inputs coming from the hardware platform of the computer system).

None of the related research on OP addresses the problem of characterizing the abnormal (or unexpected) software inputs delivered by the hardware devices supporting the software execution. The main contributions of the methodology proposed in this dissertation with respect to the related work on OP are the following:

- Consideration of the hardware platform. The majority of the related research on OP focuses on the characterization of environmental system data from a high-level perspective (e.g., physical data captured by sensors). This means that the boundaries of the OP are external to the computer system executing the software. Our work pushes those boundaries into the computer system itself by considering the contribution of the computer hardware platform to the OP.
- Characterization of unexpected inputs. Available methods and techniques for building the OP normally consider functional software inputs. We focus on nonfunctional inputs based on unexpected (or abnormal) data delivered by the hardware platform to the software. In other words, our interest is in hardware failures that may impact the behavior of the software components of the computer system.
- Use of well-established reliability methods and techniques. Existing approaches rely on expert opinion, field data or historical usage data to build the OP. Our methodology is built from well-known and established reliability methods and techniques from the microelectronics domain.

The proposed methodology thus constitutes a step forward in the OP research field, in the sense that it contributes to the development of comprehensive OP models

providing precise estimates of the actual system's operating conditions. Such an OP is referred to as *software-specific hardware failure profile*. For a given system with a computer platform executing a particular application software, the software-specific hardware failure profile is defined by tuple <p, i, f, t> denoting the probability *p* that a hardware device *i* is affected by failure *f* at time *t*. The software-specific hardware failure profile is thus the basis to extend the use of the fault injection technique to the reliability prediction of the impact of hardware failures on software. The development of the software-specific hardware failure profile requires that the mechanisms leading to hardware failures in a computer platform be carefully considered. To calculate the different variables of the software-specific hardware failure profile, we have developed a set of analytical and simulation-based methods that account for the underlying physics and environmental phenomena leading to the production of hardware failures in computer platforms during system operation.

The software execution on hardware devices, in essence, is a series of 0 and 1 signal alternations for the inputs of hardware components. Such signal alternations lead to voltage changes and current flows in the microelectronic hardware device. The voltage and current act as electrical stresses on the device and may lead to physical changes, also referred to as degradations. Failure occurs when degradation reaches the point where the device can no longer perform its intended functions.

Hardware failures created during circuit operation can be categorized into intrinsic and extrinsic failures. Extrinsic failures are the failures not related to the device circuitry itself, but failures extrinsic to the chip, such as open wire bonds in device packaging. Intrinsic failures are caused by intrinsic defects of semiconductor

devices due to limitations of the material properties of the silicon chip or limitations of the manufacturing process. Examples of manufacturing process defects are ion contamination and atom gradients caused by mechanical stresses. Such non-lethal defects can grow into lethal ones when stressed by different failure mechanisms. One type of failure mechanism stresses the device through environmental conditions (e.g. temperature and humidity), which are not related to the software execution during device operation. Typical mechanisms of this type include Temperature Cycling [23, 24] and Corrosion [25, 26], and Stress Migration [27]. The other type of failure mechanism degrades the circuitry during device operation when the device is put under dynamic voltage and current stresses due to software execution. The primary and most studied failure mechanism of this type are Hot Carrier Injection (HCI) [28-31]; Electromigration (EM) [32-36] and Time Dependent Dielectric Breakdown (TDDB) [37-42]. This work examines the hardware intrinsic failures caused by the electronic stresses introduced by the execution of the software during device operation.

During device operation such failure mechanisms could cause shifting of device response parameters, such as voltage, capacitance, and resistance, to the point that they will not meet the designed values. For example, HCI could lead to shifts of threshold voltage, transconductance, mobility and saturation current of MOSFET transistors, while EM could increase the resistance of metal interconnects.

The best way to model the failure mechanisms is from the physics-of-failure point of view. If this can be accomplished, one could have a complete picture of how a device might fail. However, limited knowledge of these failure mechanisms

currently prevents us from completely modeling the physics-of-failure. Without detailed knowledge of the device physics, the best option to represent device failures is through probabilistic statistical models. Such models use observed relationships between failure times and various input parameters, such as voltage and current, to generate probabilistic assessments of when failures may occur [43]. Accelerated Life Testing (ALT) techniques are used to model the relationship between device lifetime and different electrical stresses [44].

Device failures due to such failure mechanisms will result in changes of circuit functionality, which will affect the execution of software running on the hardware platform. To study the impact of the hardware failures on software reliability, we have to investigate the circuit behaviors under the presence of hardware failures caused by these failure mechanisms. The connections between device failure mechanisms and circuit functionality are the failure equivalent circuit models. The underlying concept of the failure equivalent circuit models is to model device degradation with some additional lumped circuit elements (transistors, resistors, dependent current sources, etc.) to capture the behavior of a damaged circuit element in the circuit operation environment. In the past years, several failure equivalent circuit models have been developed for different failure mechanisms [45-51]. Most of these circuit models are based on the SPICE simulation platform, which is the de facto tool in circuit design.

Li et al adopted a one-dimensional HCI transistor degradation model, developed by Leblebici [47], and built a two-transistor HCI degradation model. The model is used to simulate the behavior of some benchmark circuits under the presence of Hot

Carrier Injection [52]. The study showed that circuit delays induced by HCI cannot be ignored in submicron devices.

Segura et al investigated the circuit functionality of CMOS gates with damaged gate oxide due to TDDB failure mechanism. The failure equivalent circuit model consists of a series connection of two transistors and a resistance between the gate and the common terminal [53].

The above studies focused on only one particular type of failure mechanism. Most similar works consider different failure mechanisms separately. However, in order to conduct a system wide reliability estimation, all related failure mechanisms should be accounted for.

Srinvasan et al developed an architecture-level microprocessor model that is used to calculate processor lifetime reliability. Multiple failure mechanisms are included in the model to investigate the hardware lifetime with the consideration of some environment stresses (thermal cycling and mechanical stresses). The emphasis of the work is to dynamically provide processor failure rate information under different software applications and environmental stress conditions [54]. The software applications are used to simulate hardware device operation. However, the work does not provide a way to use the hardware reliability information for the evaluation of software reliability due to hardware failures. The main contributions of the methodology proposed in this dissertation with respect to the related work on hardware reliability are the following:

- It systematically calculates the hardware reliability during device operation as a function of the software execution. The interdependencies between

hardware and software in the creation of hardware failures are thus taken into account.

- We have considered all those failure mechanisms (HCI, EM, and TDDB) that are activated as a result of hardware usage induced by software execution. In other words, it accounts for a comprehensive set of hardware device intrinsic failure mechanisms that are directly related to software execution during device operation.
- It not only accounts for the failure probability of the circuit, but also investigates the probability of all possible failure manifestations (delays, stuck-at signals, changes of circuit functionalities, etc.) induced by the different failure mechanisms considered.
- It provides software-specific hardware reliability information, which is the basis for estimating the software reliability induced by hardware support failures.
- Most related research focuses on hardware reliability [47, 52, 53], and does not analyze the impact of hardware failures on software. The study by Srinvasan [54] considers the software only as a means to simulate the hardware operation, without systematic consideration of the influence of the software on hardware failures. Therefore, none of the related work analyzes the interactions and interdependencies between hardware and software with respect to reliability. Thus, this work is a bridge between microelectronic reliability and software reliability.

1.2 Contents of the Dissertation

The contents of the dissertation are described as follows.

Chapter 2 analyzes the different types of permanent failures that impact semiconductor devices. The analysis is performed at the different hardware design levels: physical device level, logic level and register transfer level. In particular, we first study permanent failures at the physical level (e.g., intrinsic and extrinsic failures, electrical stress failures, etc.). In this dissertation, we focus on the intrinsic failure induced by HCI, EM, and TDDB failure mechanisms during device operation. Then the way in which the failures propagate and manifest at higher hardware levels (e.g., delay, stuck-at value, different functionality, etc.) is examined.

Chapter 3 describes the methodology proposed for the analysis of the impact of permanent hardware failures on software reliability. The methodology is divided into three parts: (i) analysis of the manifestations of permanent failures on circuit elements (logic gates, flip-flops, etc.), (ii) development of reliability models as functions of the software execution, and (iii) calculation of failure probability distributions of the hardware circuit elements under the software execution. The analysis of the failure manifestation is performed through the use of SPICE simulations and failure equivalent circuit models. The reliability models take into account existing models for the DC stress failure rates, and integrate new models for the corresponding duty factors. These models also consider the way in which software executes through the hardware circuit elements in a computer system. The calculation of the failure probability distributions of the circuit elements is performed using Synopsys VCS MX simulator.

Chapter 4 consists of a comprehensive case study. The methodology is applied to all the CPU registers and ALU logic gates of a computer system based on the Z80 microprocessor. About 120 different types of failure manifestations have been observed, and more than 250 reliability models for the different types of failure manifestation and circuit element developed. Several structures for the reliability models and different notations for the failure manifestations are proposed in order to handle the complexity of the reliability models and obtain a practical and reduced set of models. Such models are used for the calculation of the failure probability distributions of the CPU registers and ALU gates of the Z80 computer system under the software execution.

Chapter 5 extends the methodology and the case study to the consideration of transient failures or SEUs (Single Event Upsets). First, a study of the causes and manifestations of transient failure in semiconductor devices is provided. Then, we develop reliability models for transient failures, which integrate into the same framework a set of well-known analytical models for the failure rate calculation of Single Event Upsets (SEUs). These take into account SEUs induced by cosmic ray particles (heavy ions and protons), neutrons present in the atmosphere, as well as neutrons emitted by nuclear reactors such as the ones that will be used in future nuclear-owered space missions from NASA. The models use design and technology parameters of the IC hardware devices, the operational environment characteristics (radiation particle fluxes) as well as the specifications of the system and mission (e.g., spacecraft shielding and orbit). The case study is then extended to the consideration

of transient failures by calculating the failure probability distributions due to SEUs of the hardware devices of the Z80 based computer system.

Chapter 6 provides the conclusions of the dissertation and proposes future research directions.

1.3 Summary of Contributions

The main contributions of this dissertation are summarized as follows.

It takes into account the influence of the software execution, the operational environment and the semiconductor design and technology in the creation and activation phenomena of hardware failures.

It includes the whole spectrum of hardware failures that can arise during the system operation, i.e. not only Single Event Upsets (SEUs), but also permanent semiconductor device failures due to Hot Carrier Injection, Electromigration, and Oxide Breakdown.

It considers all the possible locations for the hardware failures, i.e. not only sequential logic circuits (registers, memory cells, etc.) but also combinational logic circuits (logic gates)

It analyzes the propagation of failures under particular operational conditions (including the software execution) and precisely determines the form under which each hardware failure manifests (stuck-at-1, stuck-at-0, bit-flip, change of functionality, etc.) at circuit level.

It takes into account the usage of the hardware circuit elements due to software execution during the operational life of the system and provides the failure probability distributions of the circuit elements. This information can facilitate both software and

hardware reliability engineers to improve the system reliability more efficiently by focusing on the most failure-prone circuit elements.

It can be used to extend the use of the fault injection technique to the software reliability prediction under hardware failures and allows for precisely defining representative fault models that can be used in fault injection techniques and tools. It also provides the basis to develop testbeds based on software implemented fault injection (SWIFI) to calculate the final failure probability of the software application. As far as we know, this is the first time that such an extension has been proposed.

Chapter 2 Hardware Failures

The term “hardware failures” refers to the malfunction of semiconductor devices (Physical Device Level) and their impact (or propagation) on higher hardware levels, namely the Logic Gate Level and the Register Transfer Level (RTL). In this work we also use the term hardware faults as a synonym for hardware failures, since the latter can also be the origin or cause of further errors and failures at higher layers of the system (e.g., software layer). Irrespective of the level considered, hardware failures can be classified according to their duration into permanent (remain indefinitely), transient (have a limited duration) and intermittent (as transients, but manifest repeatedly). In this work, we analyze the mechanisms and events leading to permanent failures at the physical level as well as their impact on the higher hardware levels. Intermittent failures are not addressed because they are produced by the same mechanisms as permanent failures (moreover many of them eventually transform into permanent failures). An analysis of the impact of transient failures on software reliability will be conducted in Chapter 5.

2.1 Permanent Physical Failures

Permanent failures are irreversible physical defects in semiconductor devices introduced during manufacturing or system operation. In general, permanent failures can be divided into intrinsic, extrinsic and electrical stress failures [32].

Intrinsic failures are related to defects of semiconductor devices due to limitations of material properties of the silicon chip or limitations of the manufacturing processes. These defects may be small enough so that they are not

lethal (e.g., material impurities), or result in a device being fatally defective (e.g., an open in metal interconnect). Examples of manufacturing processes related defects include ion contamination (Surface Inversion) and atom gradients caused by mechanical stresses (Stress Migration). Semiconductor material properties are stressed by both the environmental conditions (temperature and humidity) and the operational usage (voltage and current). These stressors are called wear-out mechanisms, and may cause non-lethal defects to become lethal. The environmental wear-out mechanisms include Temperature Cycling (mechanical fatigue of the devices due to the temperature) and Corrosion (due to humidity). The wear-out mechanisms related to the operational usage are Hot Carrier Injection (HCI), Electromigration (EM) and Time Dependent Dielectric Breakdown (TDDB) [55].

Extrinsic failures are identified with the interconnection and packaging of the silicon chips. Typical failure mechanisms include die fracture, open bond joints, voids at bonds, etc., which are external to the device circuitry itself.

Electrical stress failures are generally caused by discrete events introduced during device handling in service. These damaging events include Electrostatic Discharge (ESD) and Electrical Overstress (EOS).

The focus of this work is on hardware failures that are directly introduced during the device operation, that is, intrinsic permanent failures due to operational usage (HCI, EM and TDDB). Whenever software executes on hardware platform, the hardware device is stressed by these failure mechanisms. The corresponding failures, in turn, may cause software-execution errors, which means software reliability will be affected. This work focuses on studying the probability of such hardware failures due

to the execution of software. It also provides necessary information to evaluate software reliability induced by these hardware failures.

Intrinsic failures caused by environmental stressors can impact how software executes in a microprocessor. However, they are not induced by the execution of software. For example, corrosion failures occur when the hardware device are in the presence of moisture and contaminants. The lifetime of corrosion failures is expressed as a function of relative humidity and temperature [25, 26]. Stress migration failures are induced when the device is put under mechanical stresses. The lifetime of stress migration failures is expressed as a function of mechanical stress load and temperature [27]. Such environmental stresses are not introduced by software execution, and they could cause hardware failures even when the device is operating. They are not the focus of this work. The impact of such failures on software reliability could be studied in future work.

Similarly, even though extrinsic and electrical stress permanent failures can also impact the system in operation, they are not introduced directly due to software execution. Therefore, these failures are not considered in this study.

2.1.1 Hot Carrier Injection Failure Mechanism

Hot Carrier Injection (HCI) refers to the phenomenon by which carriers (electrons or holes) at the drain end of a MOSFET (Metal-Oxide Semiconductor Field-Effect Transistor) transistor gain sufficient energy to be injected into the gate oxide and cause shifts of some MOSFET parameters, such as threshold voltage, transconductance, mobility and saturation current. This occurs as carriers move along the channel of a MOSFET (the conductivity path between the source and drain of a

field effect transistor) and experience impact ionization near the drain end of the device due to a high lateral electric field. Some high-energy electrons and/or holes produced by the impact ionization are redirected and accelerated to the interface of the oxide and silicon surface. A few of these "lucky" carriers overcome the surface energy barrier, inject into the oxide, and generate interface states and oxide charges. The shifts of threshold voltage and transconductance are proportional to the average density of "traps" (imperfections in a semiconductor material that can capture a free electron or hole), which in turn is inversely proportional to the effective channel length.

Due to the higher mobility and lower energy barrier, hot electrons are much easier to be injected into the oxide than hot holes, which means that nMOSFET transistors are more prone to HCI effects than pMOSFET transistors. Therefore, pMOSFET transistors are seldom a limiting factor in the reliability of a CMOS technology, and can be usually ignored from reliability estimates [28, 56].

The HCI lifetime of a transistor can be determined by:

$$MTTF^{hci} = A_{HCI}\left(\frac{I^{sub}}{W}\right)^{-n} \exp\left(\frac{E_{aHCI}}{\kappa T}\right) \tag{2.1}$$

where

A_{HCI} the model prefactor determined from life testing

I^{sub} the average substrate current

n a technology related constant

W the transistor channel width

E_{aHCI} apparent activation energy for HCI

κ Boltzmann's constant

T the absolute temperature

2.1.2 Electromigration Failure Mechanism

Electromigration (EM) is the mass transport of a metal due to the momentum exchange between the conducting electrons that move in the applied electric field and the metal atoms that make up the interconnecting material. It exists wherever electric current flows through metal wires. The EM failure lifetime model of a metal interconnect is characterized as [36]:

$$MTTF^{em} = A_{EM}\left(\frac{I}{A}\right)^{-n} T^{-m} \exp\left(\frac{E_{aEM}}{\kappa T}\right) \tag{2.2}$$

where

A_{EM} the model prefactor

I the average current

A the cross section of the interconnects

T the absolute temperature

E_{aEM} the activation energy for EM failure mechanism

k Boltzmann's constant

n, m material and failure mode dependent parameters

EM decreases the reliability of ICs. In the worst case, it leads to the eventual loss of one or more connections and intermittent failure of the entire circuit. Since the reliability of interconnects is not only of great interest for space travel and military applications but also for civilian applications like the anti-lock braking system of cars, high technological and economic values are attached to this effect.

2.1.3 Time Dependent Dielectric Breakdown Failure Mechanism

When an electric field is applied to the dielectric-isolated gate of a MOSFET, the progressive degradation of the dielectric material will result in the formation of conductive paths in the oxide and a shortening of the anode and cathode. When this happens, the continuous stress of the electric field on the gate oxide may lead to excessive energy dissipation, or even thermal runaway, through breakdown paths. The electrical after-effects of oxide breakdown are an abrupt increase in gate current and a loss of gate voltage controllability on device current flowing between drain and source. This kind of failure mechanism is known as Time Dependent Dielectric Breakdown (TDDB) or oxide breakdown, and the degradation process will be accelerated as the thickness of the gate oxide decreases with continued device scaling.

The TDDB lifetime of a transistor is given by [40-42]:

$$MTTF^{tddb} = A_{TDDB}\left(\frac{1}{A}\right)^{\frac{1}{\beta}} V_{gs}^{a+bT} \exp\left(\frac{c}{T}+\frac{d}{T^2}\right) \tag{2.3}$$

where

A_{TDDB} the model prefactor

a, b, c, d empirically determined constants

A the device gate oxide area of the transistor, equivalent to W (channel width) × L (channel length)

V_{gs} the gate-to-source voltage

T the absolute temperature

β Weibull distribution slope parameter

2.2 Impact on Higher Hardware Levels

Microelectronic hardware circuit design, especially digital IC design, can be divided into four different abstraction levels, as shown in Figure 2.1.

- Behavioral Level: it describes the function (or behavior) components of the system. It specifies the input and output of the component and the function the component carries.
- Register Transfer Level (RTL): a behavioral component is decomposed into combinational logic and storage elements. The storage element (flip flop, latch) is normally controlled by the system clock. The combinational logic provides access control to the storage element.
- Logic Level: the design is represented as a netlist (or combination) with different logic gates (AND, OR, NOT, etc.) and storage elements. The difference between this level and the RTL level is that one can observe the individual gates at this level but only blocks that represent storage and combinational logic at the RTL level.
- Layout Level: this level is the bottom of the hierarchy. This level describes the layout of the actual transistors and their inter-connections.

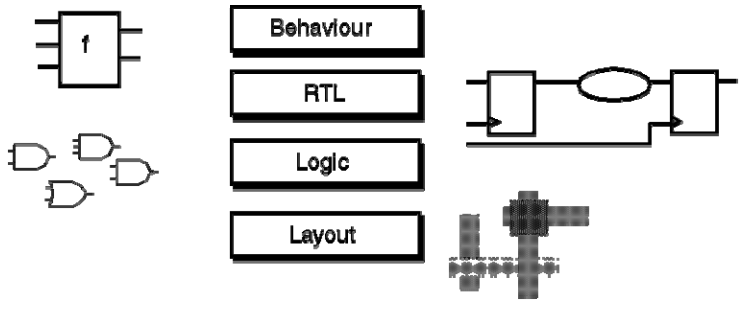


Figure 2.1 Different abstraction levels of circuit design

In the previous section, the analysis of permanent failures was described at the physical device level of the hardware, that is, with respect to the transistor elements and their physical interconnections, which are the building blocks of the circuit layout. Usually, IC design process starts from the top level (i.e., behavioral level) and goes down step-by-step to the layout level for the targeted semiconductor technology node.

RTL level consists of combinational logic circuits and storage elements, which are further decomposed into discrete logic gates and flip-flops at the logic gate level, as illustrated in an example of RTL design in Figure 2.2.

As explained in Section 2.1, a permanent failure at the physical level that becomes lethal leads to a transistor stuck-on/stuck-off or an open/short in a metal wire. At the logic and RT levels, these physical defects mainly manifest as stuck-at values (the logic voltage of a signal is stuck either at 0 or 1), indeterminate values (the logic voltage of a signal is neither 0 nor 1) and signal delays. Also, the propagation of permanent failures may lead to a functionality change of a combinational logic element, e.g., a transistor stuck-on failure in a NAND gate could change the truth table of the gate, making it behave differently than desired [57]. Besides, permanent failures can also propagate from the combinational logic to the storage elements in the form of bit-flips. Note that stuck-at values are not the only possible manifestations of permanent hardware failures.

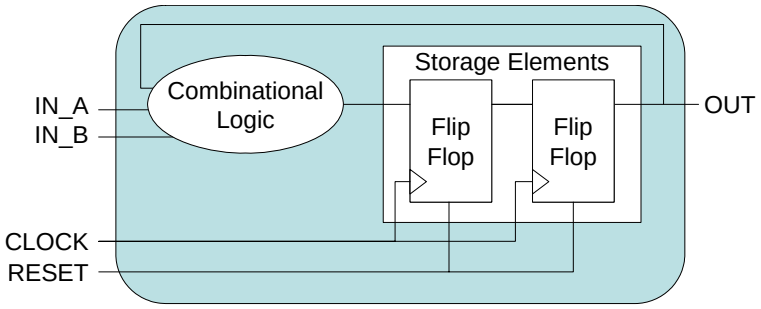


Figure 2.2 Example of a RTL Design

Chapter 3 Methodology for the Analysis of Permanent Failure Manifestations

This chapter describes the methodology for the analysis of permanent hardware failure manifestations. The methodology consists of three phases, which are described as follows.

During the first phase of the methodology (analysis of failure manifestations), SPICE simulation is performed to investigate the behavior of the circuit elements (logic gates and flip-flops) under study with a set of generic input stimuli, which covers all possible combinations of logic levels and transitions of the input signals. This allows for calculating the failure rates of different circuit elements. A set of failure equivalent circuit models for different failure mechanisms, including HCI, EM and TDDB is used to study the circuit failure manifestations under the presence of hardware failures. The main outcome of this phase consists of the set of manifestations of the permanent failures observed in the circuits' output signals (e.g., signal delays, functionality changes or stuck-at failures).

During the second phase of the methodology (development of reliability models), a set of reliability models are built that allow for calculating the occurrence rate of each failure manifestation of a circuit as a function of the software execution profile of a computer system. The models are based not only on existing expressions for the constant stress failure rate of permanent failures, but also on specifically developed models that account for the operational conditions of circuits (e.g., current and voltages) and for the usage of the computer hardware devices as a consequence of the

software execution. Different structures and notations are proposed for the reliability models in order to reduce huge numbers of failure manifestations into practical sets of expressions.

During the third phase of the methodology (calculation of failure probability distributions), the reliability models developed in the previous phase are applied to a particular computer platform. The usage of the hardware devices is obtained through VHDL simulations of the computer system under the execution of the software program of interest. This allows for solving the reliability models and calculating the failure probability distributions (per failure manifestation) of the various hardware devices of the computer platform (e.g., ALU gates, CPU registers, memories, etc.).

The methodology is divided into five steps, two steps for the first phase, one for the second phase, and two for the third phase, as illustrated in Figure 3.1. Each step will be described in detail in the following sections.

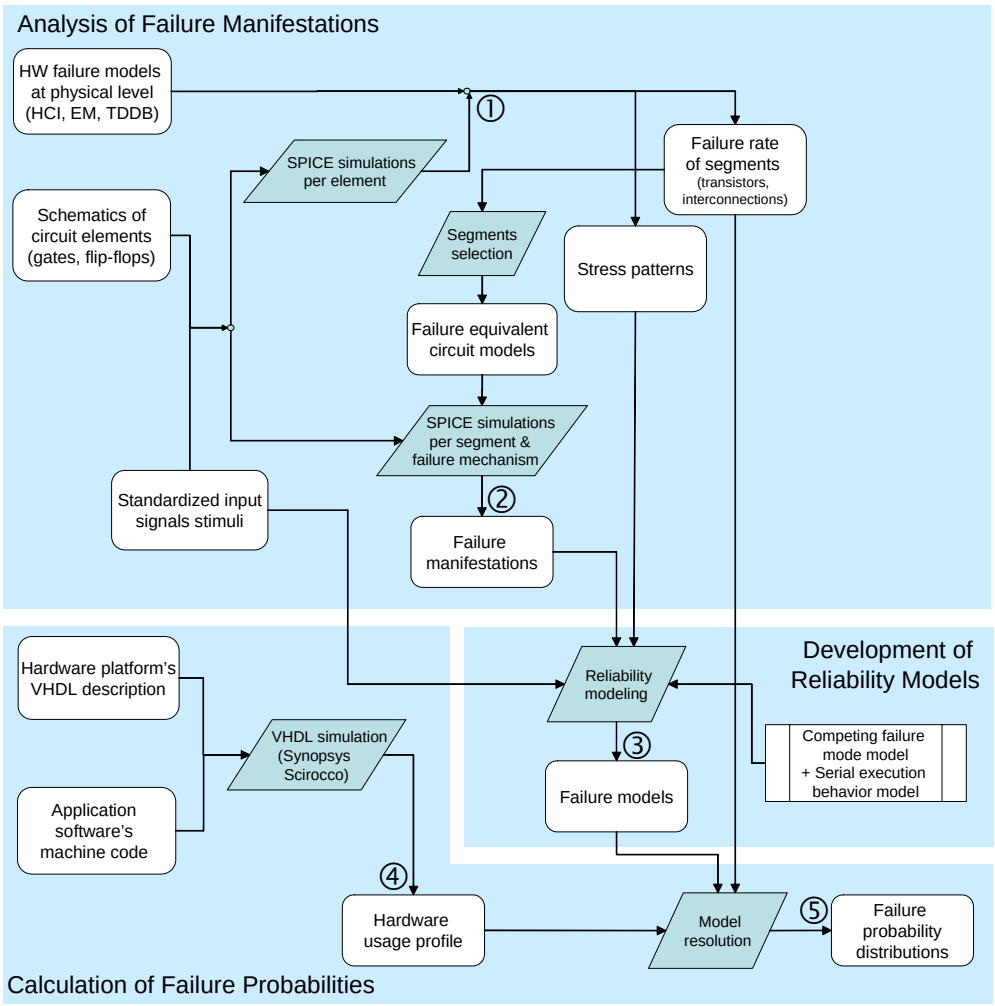


Figure 3.1 Methodology for the analysis of permanent failure manifestations

3.1 Analysis of Failure Manifestations

3.1.1 Calculation of Failure Rates and Characterization of Stress Patterns

This section describes step 1 of Figure 3.1.

As explained in Section 2.1, the hardware failures of interest in this study are intrinsic failures that are directly related to software execution. The execution of the software can be seen as a series of electrical signals based on logic 0s and 1s that activate the circuits of the computer hardware platform following a particular profile.

Of all the intrinsic failure mechanisms, the three most dominant mechanisms are considered in this study. They are Hot Carrier Injection (HCI), Electromigration (EM), and Time Dependent Dielectric Breakdown (TDDB).

Based on the lifetime models (2.1), (2.2) and (2.3) discussed in section 2.1, the corresponding failure rates of a circuit element (transistor or metal interconnection) are calculated as:

$$\lambda_i^{hci} = \frac{1}{MTTF_i^{hci}} \tag{3.1}$$

$$\lambda_i^{em} = \frac{1}{MTTF_i^{em}} \tag{3.2}$$

$$\lambda_i^{tddb} = \frac{1}{MTTF_i^{tddb}} \tag{3.3}$$

Failure rates expressed in (3.1), (3.2) and (3.3) are referred to as constant (or DC) stress failure rates, since lifetime models (2.1), (2.2) and (2.3) for the reliability evaluation of the different failure mechanisms are obtained from accelerated testing experiments under DC stress conditions. For example, to obtain the HCI lifetime model (equation (2.1)), a series of accelerated life testing experiments are conducted at several substrate current and temperature combinations for the transistors. For a particular combination, the testing is performed by keeping the substrate current and temperature constant during the total testing time. The failure data obtained from testing are then used to extract parameters in the lifetime model by means of maximum likelihood estimation techniques. The models for EM and TDDB (equations (2.2) and (2.3)) are developed in a similar way.

With the lifetime models, the failure rates of all the failure mechanisms can be calculated if we have the device operation conditions, such as the gate to source voltages (Vgs) and substrate currents of the transistors. The device operation condition can be obtained through SPICE simulation.

SPICE stands for Simulation Program with Integrated Circuits Emphasis. It is a general-purpose circuit simulation program for nonlinear dc, nonlinear transient, and linear ac analyses. SPICE provides a detailed analysis of active components including bipolar transistors, field effect transistors, and diodes, as well as lumped components, such as resistors, capacitors and inductors. Note that SPICE is a circuit simulation program, not a logic simulation program. Thus SPICE considers the voltages and currents in a circuit to be continuous quantities, not quantized into high/low values. Other constants, fitting parameters, and activation energies can be extracted from experimental data (e.g., accelerated stress tests or industry data).

SPICE uses as main inputs the schematics of a circuit of the technology under consideration (e.g. Figure 3.3), and the input signal stimuli (e.g., Figure 3.4 and Figure 3.5).

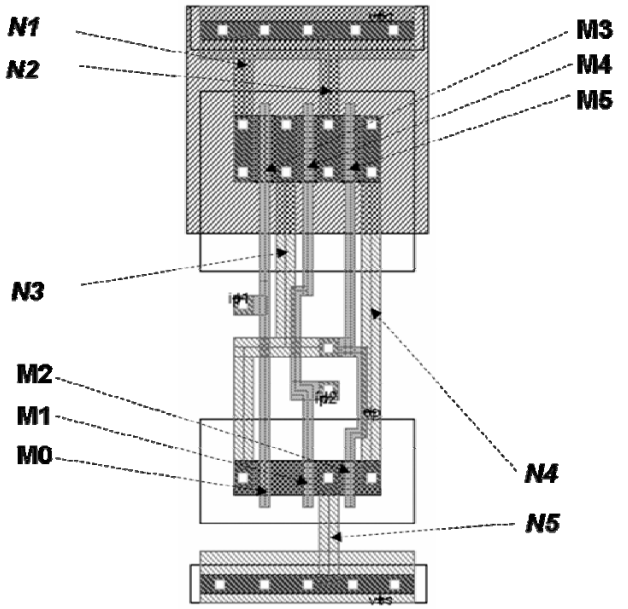


Figure 3.2 Circuit Layout of an AND2_1 logic gate

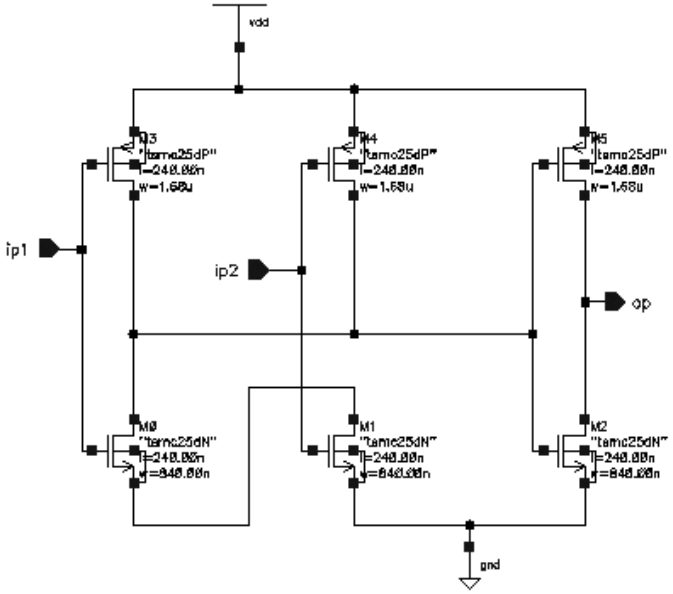


Figure 3.3 Schematic of the AND2_1 logic gate

As an example, consider an AND2_1 logic gate implementing logic operation "And" of two input signals (i.e., given two inputs a and b, the output is $a \wedge b$). The

gate belongs to standard cell library vtvtlib25 developed by the Virginia Tech VLSI for Telecommunication (VTVT) Group [58, 59], based on TSMC 0.25 μm technology. The physical layout of this gate is shown in Figure 3.2. It is composed of 6 transistors ($M0$, $M1$, $M2$, $M3$, $M4$, and $M5$) and 5 interconnections ($N1$, $N2$, $N3$, $N4$, $N5$). The corresponding schematic is shown in Figure 3.3. The AND2_1 logic gate is used throughout this section to illustrate the methodology.

For the purposes of our analysis, the input signals to a circuit are designed according to the following criteria:

The set of input signals includes all the possible combinations for logic levels (1's and 0's) and transitions (rising and falling edges). We assume that two or more transitions in different lines cannot happen at the same time.

The set of input signals leads to the same duration for every combination of logic levels. The actual duration of a pulse is not important for our analysis, while the value for the duration of a transition period will be given by the particular semiconductor technology under analysis.

We refer to a set of input signals matching these criteria as Standardized Inputs. Figure 3.4 and Figure 3.5 provide examples of Standardized Inputs.

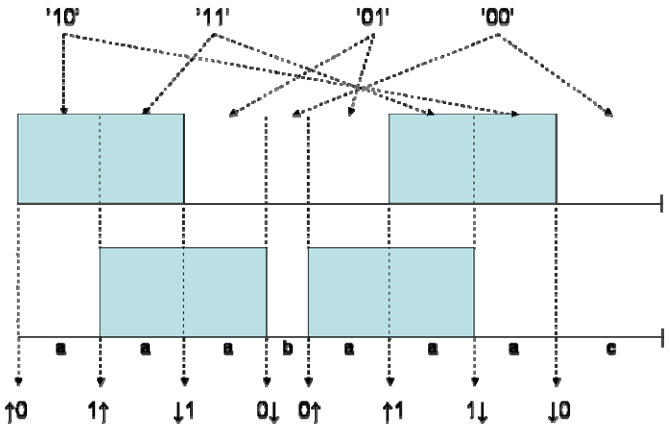


Figure 3.4 Asynchronous input signals

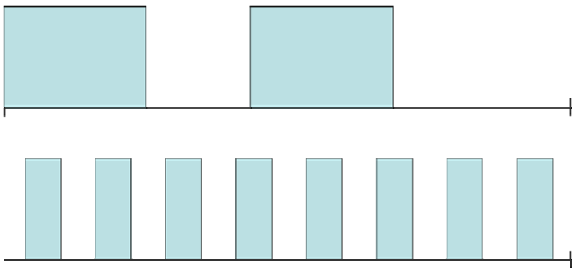


Figure 3.5 Asynchronous and synchronous (clock) input signals

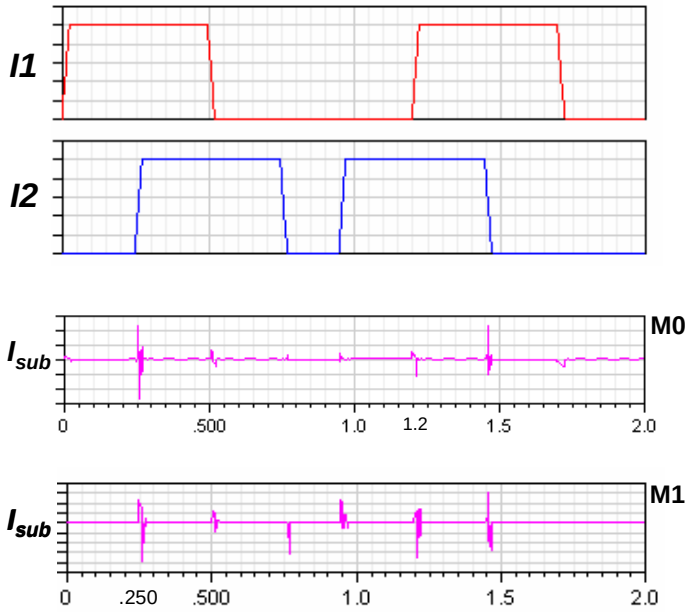
Consider the 2-asynchronous input signals of Figure 3.4. The first criterion is fulfilled since

- there is a rising transition in the first input when the second input is high or low,
- there is a falling transition in the first input when the second input is high or low,
- there is a rising transition in the second input when the first input is high or low,

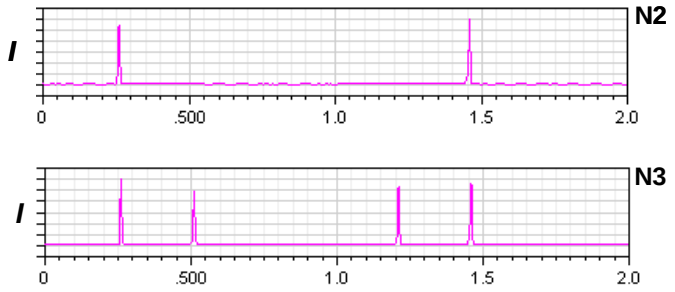
- there is a falling transition in the second input when the first input is high or low,
- all logic levels (‘00’, ‘01’, ‘10’, ‘11’) are present. The second criterion is fulfilled since the percentage of time for each logic combination of the input lines is the same (namely, $2a$ for combinations ‘01’, ‘10’, ‘11’, and $b+c$ for ‘00’, where $2a = b + c$).

It can be shown in a similar way that the two inputs of Figure 3.5 (consisting of an asynchronous signal and the clock line) also fulfill the criteria.

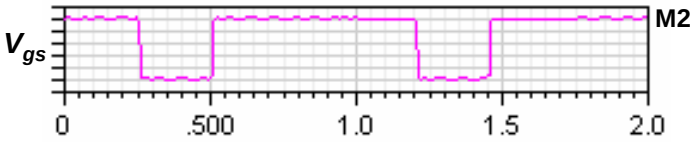
The criteria used for the design of the input signals allow for “capturing” all the different patterns (including their relative time intervals) of the occurrence of the failure mechanisms in a circuit. Indeed, transistors and interconnects suffer from HCI, EM and TDDB stresses only for specific combinations and time intervals of logic levels and transitions of the input signals. We refer to these combinations as stress patterns. This is illustrated in Figure 3.6, where some stress patterns of the AND2_1 logic gate obtained with SPICE are displayed.



a) HCI effect in transistors M0 (instant 1.2µs) and M1 (instant 0.250µs) in terms of I_{sub}



b) EM effect in interconnections N2 and N3 in terms of I



c) TDDB effect in transistor M2 in terms of V_{gs}

Figure 3.6 Stress patterns examples for HCI, EM and TDDB in the AND2_1 gate

A transistor suffers from HCI stress only during transition periods, when both gate voltage and drain voltage are high enough and there is current flowing through the channel (parameter I_{sub} of equation (2.1)). For transistor $M0$ of the AND2_1 gate, such conditions appear whenever there is a rising edge ('↑') in the first input (I1) and the second input (I2) is high ('1') (i.e., instant 1.2μs in Figure 3.6a). For transistor $M1$, it appears whenever the first input (I1) is high ('1') and there is a rising edge ('↑') in the second input (I2) (i.e., instant 0.250μs in Figure 3.6a).

A metal wire suffers from EM whenever there is an electric current flowing through it. For CMOS circuits, the current flow in an interconnection is negligible if the circuits are in static condition. However, switching input signals may introduce current pulses in the metal wires leading to EM. As shown in Figure 3.6b, EM arises in interconnect $N2$ whenever there is a rising transition in input I2 while input I1 is high, or there is a falling transition in input I2 while input I1 is high (i.e., instants 1.25μs and 1.45μs in Figure 3.6b). For interconnect $N2$, it arises whenever there is a transition in one line while the other is high (i.e., instants 0.25μs, 0.5μs, 1.2μs and 1.45μs in Figure 3.6b).

Unlike HCI, TDDB stresses the gate dielectrics of the transistors even when they are in static state operations. As described in equation (2.3), the transistors lifetime due to TDDB stress strongly depends on the gate to source voltage V_{gs}. For CMOS circuits, during normal device operation, most of the transistors experience certain periods during which the gate to source voltage V_{gs} value is equal to the power supply voltage V_{dd}. The higher the percentage of such periods, the higher the failure rates of the transistors due to the TDDB stress. As shown in Figure 3.6c, V_{gs} will

systematically equal V_{dd} in transistor $M2$ for logic levels '00', '01', and '10' of inputs I1 and I2 (i.e., time intervals [0, 0.25], [0.5, 1.2] and [1.45, 2.0] in Figure 3.6c), leading to TDDB stress.

3.1.2 Identification of Failure Manifestations

This section describes step 2 of Figure 3.1.

The identification of fault models is divided into the following steps:

1) Development of a failure-equivalent-circuit model for the segments,
2) Independent simulations of the element by substituting each time one segment by the failure-equivalent-circuit model,
3) Observation of the output signal in each simulation and determination of the fault models.

Once the failure rates (λ_i^{hci} , λ_i^{em} , λ_i^{tddb}) are calculated for every transistor Mi and interconnection Ni of a circuit element, a selection is made based on the elimination of those segments that are softly or not stressed by the failure mechanisms, so their impact in the global failure probability of the circuit can be neglected. This means that segment i under stress j will not be given any further consideration in our analysis if $\lambda_i^j \approx 0$. This is for instance the case of the pMOSFET transistors of a circuit, which as explained in Section 3.1.1 are barely impacted by the HCI stress.

Figure 3.7, Figure 3.8 and Figure 3.9 show an example of a segments selection for the AND2_1 logic gate.

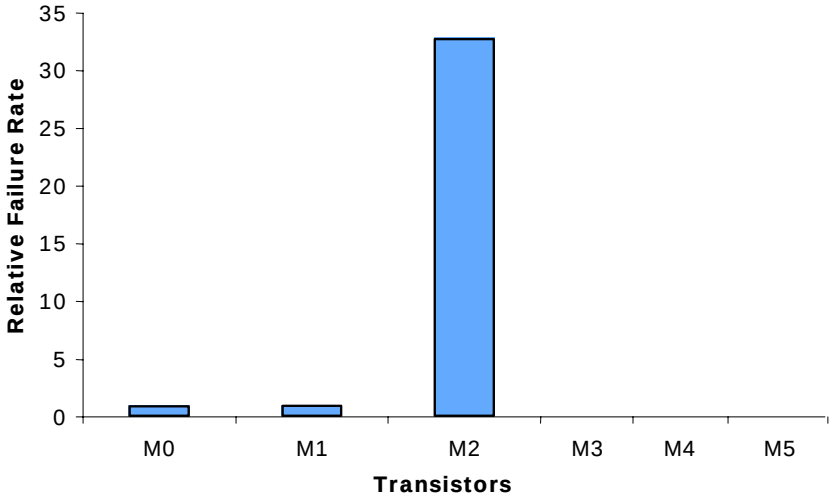


Figure 3.7 Relative HCI failure rates of transistors

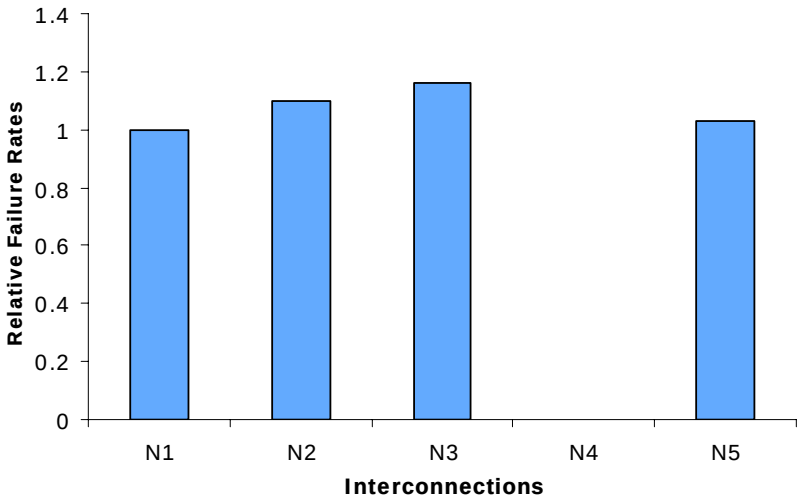


Figure 3.8 Relative EM failure rates of interconnections

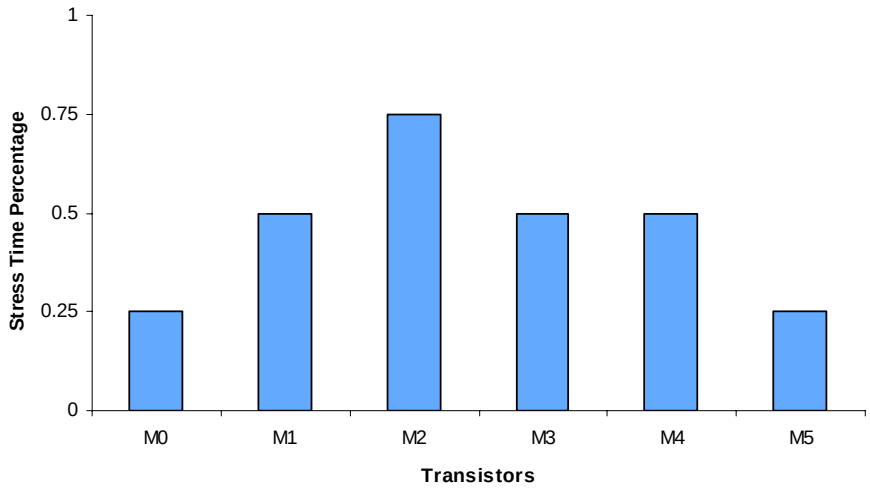


Figure 3.9 Percentage of time a transistor suffers from TDDB stress

Figure 3.7 shows the relative failure rates of the different transistors for HCI with respect to transistor M0. Figure 3.8 illustrates the relative failure rates of the different metal interconnects for EM with respect to N1. While Figure 3.9 displays the percentage time transistors suffer from TDDB stress. In this case, the segments selection simply leads to the elimination of the pMOSFET transistors under HCI stress (i.e., transistors $M3$, $M4$ and $M5$), and interconnection $N4$ under EM stress.

In order to account for the effect of different failure mechanisms on circuit functionality, several failure equivalent circuit models [51], one for each failure mechanism, are adopted to obtain circuit level failure manifestations through SPICE simulation (one per segment and failure mechanism). The underlying concept of the failure equivalent circuit models is to model device degradation with some additional lumped circuit elements (resistors, dependent current sources, etc.) to capture the behavior of a damaged circuit element in circuit operation environment. The larger

the magnitude of element values, such as the resistance of the lumped resistor, the more severe the damage to circuit functionality. The failure equivalent circuit models are provided in Figure 3.10, Figure 3.11, and Figure 3.12.

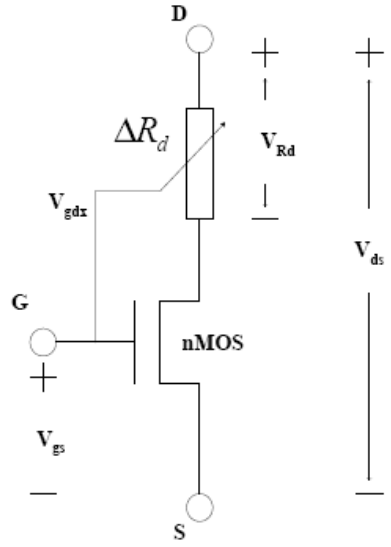


Figure 3.10 Failure equivalent circuit model for HCI mechanism

Figure 3.11 Failure equivalent circuit model for EM mechanism

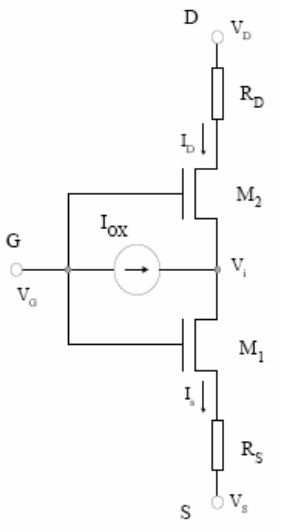


Figure 3.12 Failure equivalent circuit model for TDDB mechanism

The adopted HCI failure equivalent circuit model (Figure 3.10) is the Hot Carrier Induced Series Resistance Enhancement Model (HISREM), also named ΔR_d model (proposed by Hwang [48] and improved in [51]). The model is composed of the original nMOSFET transistor connected in series with a voltage dependent drain resistor ΔR_d , which reflects the process of hot carrier induced interface trap generation and therefore accounts for the channel mobility reduction and threshold voltage shift. The degree of circuit degradation is reflected by the value of resistor ΔR_d . The more severe the HCI damage to the circuit, the higher the resistance value.

The metal conductor used in current submicron CMOS technologies is constructed of a composite layered structure with a refractory metal layer on top and at the bottom of the aluminum alloy core metal. The effect of EM on the composite metal conductor is the increase of line resistance. Usually the failure criteria used in the EM lifetime test is an increase of the line resistance by 10 to 20 %, or a line

resistance increase by a fixed value [60]. The EM failure equivalent circuit model for a metal interconnect (Figure 3.11) is a resistor whose resistance value gets higher as the degradation becomes worse.

The TDDB failure equivalent circuit model used (Figure 3.12) corresponds to the Maryland Circuit Reliability Oriented (MaCRO) model [51]. The electrical effect of TDDB is that it provides a conduction path to inject electrons from channel into gate. Therefore, a voltage dependent current source I_{OX} can be used to connect the gate and channel of the damaged transistor to model the effect of TDDB. The circuit model for nMOSFET is shown in Figure 3.12, in which two split transistors imitate the channel separation by oxide breakdown path, and the voltage-dependent current source I_{OX} physically represents the conduction mechanism of hard breakdown path across the oxide. The magnitude of I_{OX} reflects the degree of degradation of the TDDB failure mode. The model can be extended to pMOSFET by properly changing current flowing direction in Figure 3.12.

The relation between the additional lumped circuit elements values used in the failure equivalent circuit models (such as the resistance of the resistors) and the lifetime models of the failure mechanisms (equations (2.1), (2.2) and (2.3)) is beyond the scope of this work. Therefore, the values for the mean time to failure of the circuit are calculated using equations (2.1), (2.2) and (2.3) irrespective of how high it is necessary to increase the value of the lumped circuit elements to readily observe a failure manifestation.

Assuming that the failure of a segment due to HCI, EM and TDDB damage could lead to a circuit functional error, the failure equivalent circuit models are used

to replace one segment at a time in the analysis. Each transistor of a circuit is to be replaced by the HCI and TDDB failure equivalent circuit models (Figure 3.10 and Figure 3.12), and each interconnection by the EM failure equivalent circuit model (Figure 3.11). Accordingly, if a circuit contains M transistors and N interconnections, $2M + N$ mutated versions of the same circuit are produced. A "mutant" is thus a circuit in which a segment (transistor or interconnection) considered to be faulty is replaced by a failure equivalent circuit model. An example of one of such "mutants" for the AND2_1 gate circuit is provided in Figure 3.13, where transistor $M5$ is replaced by the TDDB failure equivalent circuit model.

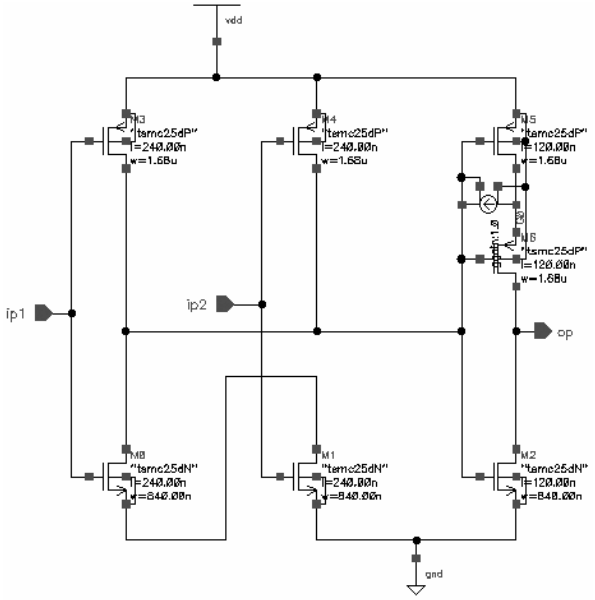
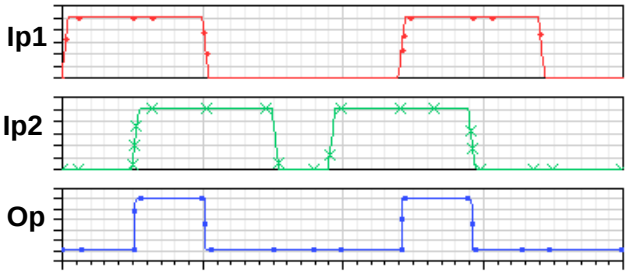


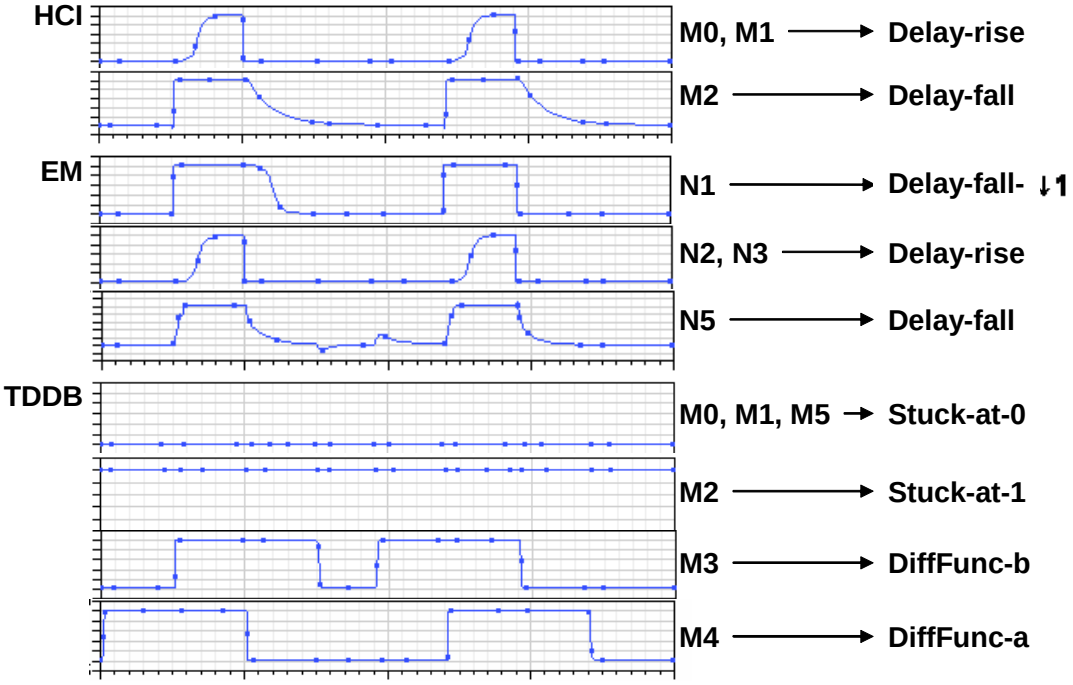
Figure 3.13 AND2_1 layout with transistor M5 replaced by the TDDB failure equivalent circuit model

The next step consists of running one independent SPICE simulation per mutated circuit. The objective is to determine whether the functional behavior of the circuit is impacted by the faulty segment and leads to a failure manifestation. To do

so, the waveform of the circuit's output signal is logged and analyzed after every simulation. The output waveforms obtained during this process for the AND2_1 gate are provided in Figure 3.14.



a) Correct behavior (Ip1 – input 1, Ip2 – input 2, Op – output)



b) Failure manifestations observed in the output (Op)

Figure 3.14 Failure manifestations of the AND2_1 logic gate due to HCI, EM and TDDB stresses in its circuit segments

Figure 3.14b displays the failure manifestations of the AND2_1 gate, i.e., those output waveforms that differ from the correct output waveform (i.e., signal *Op* in Figure 3.14a). As far as HCI is concerned, all the observed failure manifestations consist of output delays. The circuit behavior with a faulty transistor $M0$ due to HCI leads to a delay of the output signal whenever its logic level changes from '0' to '1' (referred to as Delay-rise). A similar behavior is observed for a faulty $M1$ transistor. On the other hand, the output is delayed during '1' to '0' transitions when transistor $M2$ is faulty (Delay-fall). Regarding EM stress, the observed failure manifestations correspond also to output delays: during '1' to '0' output transitions when there is a falling edge in signal $Ip1$ and interconnection $N1$ is faulty (Delay-fall-$\downarrow$1); during '0' to '1' output transitions when interconnection $N2$ (or $N3$) is faulty (Delay-rise); and during '1' to '0' output transitions when interconnection $N5$ is faulty (Delay-fall). As far as TDDB is concerned, the observed failure manifestations are the following: stuck-at-0 fault when transistor $M0$, $M1$ or $M5$ is faulty; stuck-at-1 fault when transistor $M2$ is faulty; and different-function fault when transistor $M3$ or $M4$ is faulty. In this latter case, the different functions observed are respectively "stuck-at-$Ip2$" (DiffFunc-b) and "stuck-at-$Ip1$" (DiffFunc-a).

The notation used for the failure manifestation is described in Table 3.1.

Table 3.1 Notation of the failure manifestations

	Failure manifestation	Description
"Detailed manifestation" notation	Delay-a	The output signal is always delayed, either during rise or fall transitions.
	Delay-{rise, fall}-{input	The output signal is delayed for specific

	combinations}	logic combinations and transitions of the input signals (input combinations), either during rise or fall transitions.
	Stuck-at-{0, 1}	The output signal is stuck at logic value 0 or 1
	DiffFunc-{logic function}	The circuit functionality changed to a different logic function.
"Simplified manifestation" notation	Delay-{value}	The output signal is delayed either during rise or fall transitions. The percentage of rise or fall transitions affected is indicated by value.
	Stuck-at-{0, 1}	The output signal is stuck at logic value 0 or 1
	DiffFunc-{value}	The circuit functionality changed to a different logic function. The percentage of logic combinations of the input signals leading to a different output logic level is indicated by value.

The proposed methodology allows for obtaining very detailed information about the failure manifestations of a circuit. For example, it allows for determining whether a delay affects a rising or falling transition of the output signal, what the particular status of each input signal is during a delay, or which new logical function the circuit

implements after a failure. These characteristics can be captured by the detailed manifestation notation proposed in Table 3.1. As an example, the failure manifestations presented in Figure 3.14b are labeled using the detailed notation. Table 3.1 also proposes a more compact notation for the failure manifestations, referred to as simplified manifestation. In this notation, delays are characterized by the number of deferred pulses of the output signal, while the different-function failure manifestation is characterized by the number of changes in the truth table of the circuit. As an example, failure manifestations Delay-rise and Delay-fall from Figure 3.14b would be under label Delay-1 using the simplified notation (i.e., 100% of the pulses of the output signal are delayed), Delay-fall-↓1 under label Delay-0.5 (i.e., 50% of the pulses of the output signal are delayed), and failure manifestations DiffFunc-a and DiffFunc-b under label DiffFunc-0.25 (i.e., one out of the four entries of the truth table of the AND2_1 gate is changed, namely entry "10" when the different function is "a", and entry "01" when the different function is "b").

3.2 Development of Reliability Models

This section describes step 3 of Figure 3.1.

The development of reliability models is divided into the following steps:

1) Modeling of the failure probability following a competing failure mode model.
2) Modeling of the reliability depending on the number of demands per element issued during the software execution,

The lifetime of an entire circuit results from a combination of the effects of the different failure mechanisms across different segments (transistors and interconnections). This requires information on the lifetime distribution of each failure mechanism. In a complex integrated circuit, the whole system will be extremely prone to failure if any segment fails. We can therefore approximate a complex integrated circuit using a competing failure mode model. We apply the standard sum-of-failure-rates (SOFR) model [44] widely used in industry to determine a system's failure rate from its individual failure mechanisms. Using the SOFR model, the failure rate λ of a circuit (e.g., logic gate, flip-flop, etc.) can be related to the lifetime of segments (equations (2.1), (2.2) and (2.3)) as shown in expression (3.4):

$$\lambda = \sum_{j \in H} \sum_{i \in S_j} \lambda_{i,j} \tag{3.4}$$

where

H the set of hardware failure mechanisms (e.g., HCI, EM, TDDB) that impact the circuit,

S_j the set of segments (transistors and interconnections) stressed by failure mechanism j, and

$\lambda_{i,j}$ the dynamic stress failure rate of segment i under stress j.

The dynamic stress failure rate $\lambda_{i,j}$ can be calculated in different ways, for example by means of quasi static values or duty factors. We use duty factors to calculate the dynamic stress failure rate as follows:

$$\lambda_{i,j} = w_{i,j}\lambda_i^j \tag{3.5}$$

where

λ_i^j the constant stress failure rate of segment i under stress j (i.e., equivalent to equations (3.1), (3.2) and (3.3)), and

$w_{i,j}$ the duty factor for λ_i^j, which is equivalent to the percentage of time that segment i is subjected to stress j during the circuit operation under a particular software execution.

The SOFR model elevates the reliability from transistor and interconnection levels to circuit level and is used to estimate lifetimes for various kinds of device families.

The SOFR model described in (3.4) provides the value for the failure rate of a circuit irrespective of the failure manifestation that impairs the circuit. In order to calculate the failure rate λ^F of a particular manifestation F of a permanent failure (e.g., stuck-at-0, stuck-at-1, delay, different function, etc.), we rewrite equation (3.4) as follows:

$$\lambda = \sum_{F\in\Gamma}\lambda^F = \sum_{F\in\Gamma}\sum_{j\in H_F}\sum_{i\in S_{j,F}} w_{i,j}\lambda_i^j \tag{3.6}$$

$$\lambda^F = \sum_{j\in H_F}\sum_{i\in S_{j,F}} w_{i,j}\lambda_i^j \tag{3.7}$$

where

Γ the set of failure manifestations F that impact the circuit,

H_F the set of hardware failure mechanisms (e.g., HCI, EM, TDDB) that impact the circuit leading to failure manifestation F,

$S_{j,F}$ the set of segments (transistors and interconnections) stressed by failure mechanism j which lead to failure manifestation F,

λ_i^j the constant stress failure rate of segment i under stress j, and

$w_{i,j}$ the duty factor for λ_i^j.

For example, expression λ^F for the AND2_1 gate is as follows:

$$\begin{aligned}
\lambda^F &= \sum_{j\in H_F}\sum_{i\in S_{j,F}} w_{i,j}\lambda_i^j \\
&= \sum_{i\in S_{hci,F}} w_{i,hci}\lambda_i^{hci} + \sum_{i\in S_{em,F}} w_{i,em}\lambda_i^{em} + \sum_{i\in S_{tddb,F}} w_{i,tddb}\lambda_i^{tddb} \\
&= \sum_{i\in\{M0,M1,M2\}} w_{i,hci}\lambda_i^{hci} + \sum_{i\in\{N1,N2,N3,N5\}} w_{i,em}\lambda_i^{em} + \sum_{i\in\{M0,M1,M2,M3,M4,M5\}} w_{i,tddb}\lambda_i^{tddb} \\
&= \left(w_{0,hci}\lambda_0^{hci} + w_{1,hci}\lambda_1^{hci} + w_{2,hci}\lambda_2^{hci}\right) \\
&\quad + \left(w_{1,em}\lambda_1^{em} + w_{2,em}\lambda_2^{em} + w_{3,em}\lambda_3^{em} + w_{5,em}\lambda_5^{em}\right) \\
&\quad + \left(w_{0,tddb}\lambda_0^{tddb} + w_{1,tddb}\lambda_1^{tddb} + w_{2,tddb}\lambda_2^{tddb} + w_{3,tddb}\lambda_3^{tddb} + w_{4,tddb}\lambda_4^{tddb} + w_{5,tddb}\lambda_5^{tddb}\right)
\end{aligned} \tag{3.8}$$

Note that parameters Γ, H_F and $S_{j,F}$ are determined as explained in Section 3.1.2. Also, as explained in Section 3.1.2, transistors $M3$, $M4$ and $M5$ under HCI and interconnection $N4$ under EM can be neglected for the reliability analysis of the AND2_1 gate.

To calculate duty factors $w_{i,j}$, we need to use the Standardized Inputs and the Stress Patterns described in Section 3.1.1, the Failure Manifestations described in Section 3.1.2, and also a Hardware Serial Model which is described hereafter.

The software execution on hardware devices (normally includes CPU, memory, busses etc.) is essentially a series of 0 and 1 signal alterations for hardware units. This

model assumes that the software execution on hardware devices is divided into a series of demand and idle intervals, as depicted in Figure 3.15.

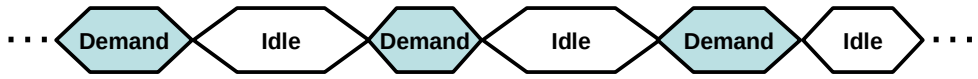
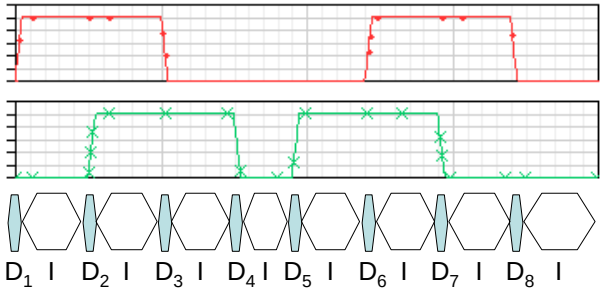
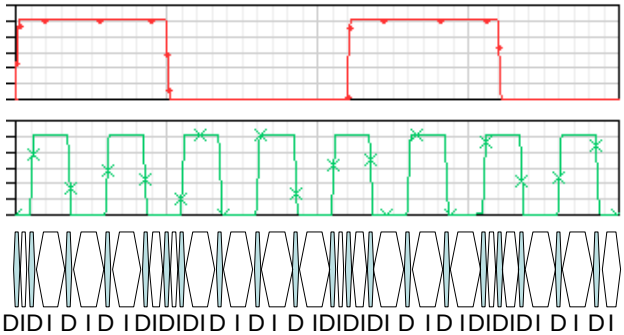


Figure 3.15 Hardware serial model during software execution

Thus the software execution in terms of a specific unit constitutes a series of being-demanded and not-being-demanded (idle) combinations. As an example, Figure 3.16 shows this execution process for a logic gate and a flip-flop when the Standardized Inputs considered in Section 3.1.1 are applied.



a) Demand (D) and idle (I) intervals of a logic gate



b) Demand (D) and idle (I) intervals of a flip-flop

Figure 3.16 Examples of the hardware serial model under a software

execution using the Standardized Inputs

As shown in Figure 3.16, a demand interval is actually triggered by a logic change of any of the input signals to the hardware device. We assume that the duration of a demand interval is equivalent to the duration of a transition period of a signal. Such a duration is symbolized by τ .

As stated above, coefficients $w_{i,j}$ can be calculated using the Hardware Serial Model described above in combination with the Standardized Inputs, the Stress Patterns and the Failure Manifestations. This is illustrated for the AND2_1 gate in Figure 3.17.

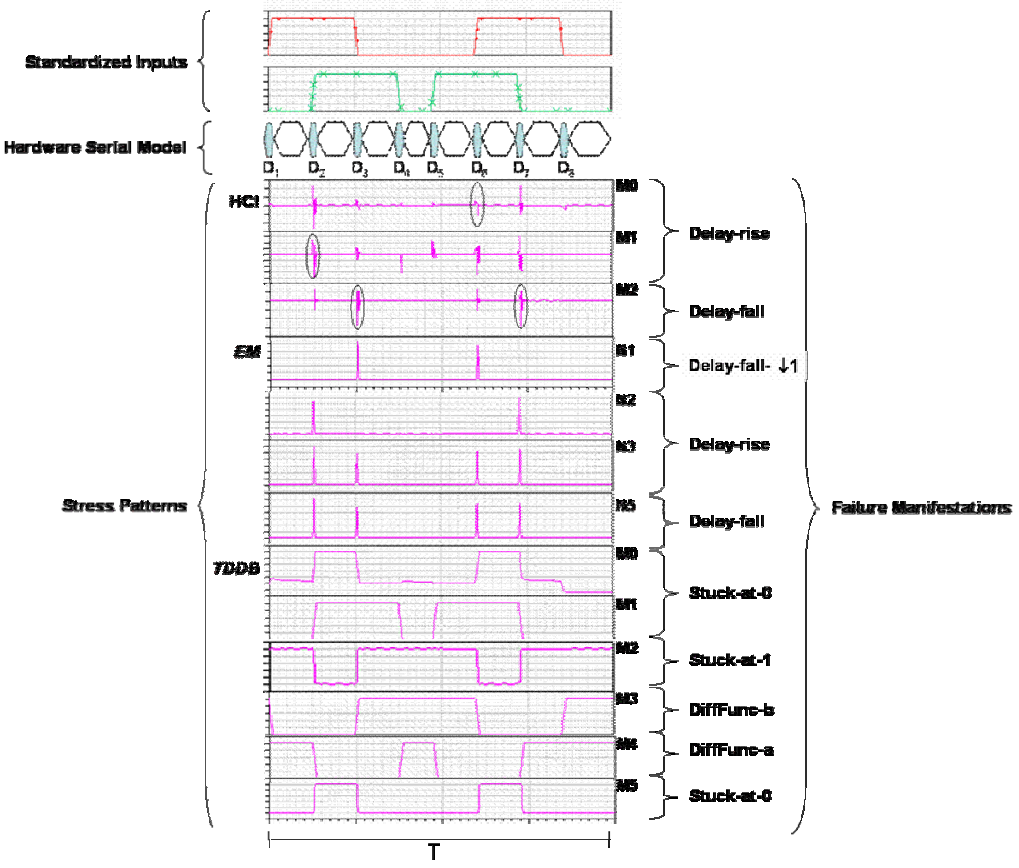


Figure 3.17 Stress patterns using the Standardized Inputs

As shown in Figure 3.17, the different current peaks of the stress patterns for the HCI and EM effects can be matched to a particular demand transition of the hardware serial model. For instance, transistor $M0$ is stressed by HCI only during demand D_6, and such a stress will lead to the Delay-rise failure manifestation. Then, coefficient $w_{0,hci}$ can be calculated as $\frac{d_6\tau}{T}$, where T is the duration of the time window within which the measurement is being performed, τ is the duration of a demand transition D_i, and d_6 is the number of demands of type D_6 that occurred within time window T (e.g., in Figure 3.17, d_i is equal to 1 for all i). On the other hand, the different voltage pulses of the stress patterns for the TDDB effect can be matched to a particular logic combination of the input signals. For instance, transistor $M0$ is stressed by TDDB only for logic combination 11 of the input signals, and such a stress will lead to a stuck-at-0 failure manifestation. Then, coefficient $w_{0,tddb}$ can be calculated as $\frac{t_{11}}{T}$, where t_{11} is the duration of logic combination 11 (e.g., in Figure 3.17, t_{ij} is equal to $\frac{T}{4}$ for all i, j).

In general, duty factor $w_{i,j}$ is given by the following expression:

$$w_{i,j} = \begin{cases} \frac{\sum_{k\in I(i,j)} d_k \cdot \tau}{T}, j \in \{hci, em\} \\ \frac{\sum_{c\in C(i,j)} t_c}{T}, j \in \{tddb\} \end{cases} \quad (3.9)$$

where

T the duration of the time window where the measurement is being performed,

$I(i,j)$ the set of sub-indexes y of those demand transition types D_y for which stress j impacts segment i (e.g., $I(3,em)=\{2,3,6,7\}$),

d_k the number of demands of type D_k that occurred within time window T,

τ the duration of a demand transition,

$C(i,j)$ the set of logic combinations of the input signals for which stress i impacts segment j (e.g., $C(2,tddb)=\{00,01,10\}$), and

t_c the duration of logic combination c of the input signals within time window T.

Note that the value of parameters d_k, t_c and T is software dependent. In other words, the execution of different software applications will lead to different values for d_k, t_c and T.

For example, using equations (3.8) and (3.9), expressions $\lambda^{Delay-rise}$ and $\lambda^{Stuck-at-0}$ for the AND2_1 gate are as follows:

$$\begin{aligned}\lambda^{Delay-rise} &= \left(\frac{d_6\tau}{T}\lambda_0^{hci}+\frac{d_2\tau}{T}\lambda_1^{hci}+0\cdot\lambda_2^{hci}\right)\\ &+\left(0\cdot\lambda_1^{em}+\frac{(d_2+d_7)\tau}{T}\lambda_2^{em}+\frac{(d_2+d_3+d_6+d_7)\tau}{T}\lambda_3^{em}+0\cdot\lambda_4^{em}+0\cdot\lambda_5^{em}\right)\\ &+\left(0\cdot\lambda_0^{tddb}+0\cdot\lambda_1^{tddb}+0\cdot\lambda_2^{tddb}+0\cdot\lambda_3^{tddb}+0\cdot\lambda_4^{tddb}+0\cdot\lambda_5^{tddb}\right)\\ &=\frac{d_6\tau}{T}\lambda_0^{hci}+\frac{d_2\tau}{T}\lambda_1^{hci}+\frac{(d_2+d_7)\tau}{T}\lambda_2^{em}+\frac{(d_2+d_3+d_6+d_7)\tau}{T}\lambda_3^{em}\end{aligned} \quad (3.10)$$

$$
\begin{aligned}
\lambda^{Stuck-at-0} &= \left(0\cdot\lambda_0^{hci} + 0\cdot\lambda_1^{hci} + 0\cdot\lambda_2^{hci}\right) \\
&\quad + \left(0\cdot\lambda_1^{em} + 0\cdot\lambda_2^{em} + 0\cdot\lambda_3^{em} + 0\cdot\lambda_4^{em} + 0\cdot\lambda_5^{em}\right) \\
&\quad + \left(\frac{t_{11}}{T}\lambda_0^{tddb} + \frac{\left(t_{11}+t_{01}\right)}{T}\lambda_1^{tddb} + 0\cdot\lambda_2^{tddb} + 0\cdot\lambda_3^{tddb} + 0\cdot\lambda_4^{tddb} + \frac{t_{11}}{T}\lambda_5^{tddb}\right) \\
&= \frac{t_{11}}{T}\lambda_0^{tddb} + \frac{\left(t_{11}+t_{01}\right)}{T}\lambda_1^{tddb} + \frac{t_{11}}{T}\lambda_5^{tddb} \\
&= \frac{(2+c)t_{11}+t_{01}}{T}\lambda^{tddb}
\end{aligned}
\tag{3.11}
$$

In equation (3.11), we assume that in the same circuit element all the nMOSFETs have the same channel width W_n, and all the pMOSFETs have identical channel width W_p. From equation (2.3) and (3.3) we can see that the same type of MOSFET transistors have the same failure rate if they are put under the same constant voltage stress. The ratio of pMOSFET transistors failure rate to that of nMOSFET transistors is a constant value c, which is calculated as:

$$
c = \frac{\lambda_{pMOSFET}^{tddb}}{\lambda_{nMOSFET}^{tddb}} = \left(\frac{W_p}{W_n}\right)^{1/\beta} \tag{3.12}
$$

where

W_p, W_n channel width of pMOSFET and nMOSFET transistors, respectively

β Weibull slope parameter in (2.3)

Actually, in vtvtlib25 standard cell library used for this study, not only the above assumption is true, but also all the cells of the same drive strength have identical nMOSFET and pMOSFET dimension. Therefore, the above analysis is applicable to other gates in the cell library. Based on this analysis, in (3.11), the failure rate of nMOSFETs due to TDDB is denoted as λ^{tddb}, and the failure rate of pMOSFETs is $c\cdot\lambda^{tddb}$.

In practice, the measurement of parameters d_k and t_c might make the calculation of the usage profile (i.e., last step of the methodology in Figure 3.1) complex and time consuming. We thus propose an alternative simplified version for expression $w_{i,j}$, as described hereafter:

$$w_{i,j} = \begin{cases} \dfrac{|I(i,j)|d}{I}\dfrac{\tau}{T}, j \in \{hci, em\} \\ \dfrac{|C(i,j)|}{C}, j \in \{tddb\} \end{cases} \quad (3.13)$$

where

T the duration of the time window where the measurement is being performed,

$|I(i,j)|$ the number of transition types D_x for which stress j impacts segment i (e.g., $|I(3,em)| = |\{2,3,6,7\}| = 4$),

I the total number of demand transition types (e.g., in general, for a n-asynchronous input signal circuit, I is equal to $n2^n$),

d the number of demand transitions of any type, $d = \sum_{\forall i} d_i$, that occurred during time window T (e.g., in Figure 3.17, d is equal to 8),

τ the duration of a demand transition D_j,

$|C(i,j)|$ the number of logic combinations of the input signals for which stress j impacts segment i (e.g., $|C(2,tddb)| = |\{00,01,10\}| = 3$),

C the total number of logic combinations of the input signals (e.g., in general, for a n-asynchronous input signal circuit, C is equal to 2^n),

Equation (3.13) is simplified since only parameters d and T are software dependent, and the measurement of parameter d is easier in practice than the measurement of d_k and t_c from equation (3.9).

Using equation (3.13), expression (3.10) for $\lambda^{Delay-rise}$ and (3.11) for $\lambda^{Stuck-at-0}$ can be recalculated as follows:

$$\lambda^{Delay-rise} = \frac{d}{8}\frac{\tau}{T}\lambda_0^{hci} + \frac{d}{8}\frac{\tau}{T}\lambda_1^{hci} + \frac{2d}{8}\frac{\tau}{T}\lambda_2^{em} + \frac{4d}{8}\frac{\tau}{T}\lambda_3^{em} \tag{3.14}$$

$$\lambda^{Stuck-at-0} = \frac{1}{4}\lambda_0^{tddb} + \frac{2}{4}\lambda_1^{tddb} + \frac{1}{4}\lambda_5^{tddb} = \frac{3+c}{4}\lambda^{tddb} \tag{3.15}$$

In expressions (3.14) and (3.15), only parameter d needs to be measured for the calculation of the usage profile.

The failure rates (both the detailed and simplified version) for the other failure manifestations of the AND2_1 gate can be calculated in a similar way using Figure 3.17 and equations (3.8), (3.9) and (3.13). These models are provided in Table 3.2.

Table 3.2 Failure rate models of the AND2_1 gate per failure manifestation

		Detailed models	Simplified models
Detailed manifestation	$\lambda^{Delay-rise}$	$\left[\begin{matrix} d_6\lambda_0^{hci} + d_2\lambda_1^{hci} + (d_2+d_7)\lambda_2^{em} \\ +(d_2+d_3+d_6+d_7)\lambda_3^{em} \end{matrix}\right]\frac{\tau}{T}$	$\left(\frac{1}{8}\lambda_0^{hci} + \frac{1}{8}\lambda_1^{hci} + \frac{1}{4}\lambda_2^{em} + \frac{1}{2}\lambda_3^{em}\right)\frac{d\tau}{T}$
	$\lambda^{Delay-fall}$	$\left[\begin{matrix} (d_3+d_7)\lambda_2^{hci} \\ +(d_2+d_3+d_6+d_7)\lambda_5^{em} \end{matrix}\right]\frac{\tau}{T}$	$\left(\frac{1}{4}\lambda_2^{hci} + \frac{1}{2}\lambda_5^{em}\right)\frac{d\tau}{T}$
	$\lambda^{Delay-fall-\downarrow 1}$	$(d_3+d_6)\frac{\tau}{T}\lambda_1^{em}$	$\frac{1}{4}\frac{d\tau}{T}\lambda_1^{em}$

	$\lambda^{Stuck-at-0}$	$\frac{(2+c)t_{11}+t_{01}}{T}\lambda^{tddb}$	$\frac{3+c}{4}\lambda^{tddb}$
	$\lambda^{Stuck-at-1}$	$\frac{t_{00}+t_{01}+t_{10}}{T}\lambda^{tddb}$	$\frac{3}{4}\lambda^{tddb}$
	$\lambda^{DiffFunc-a}$	$\frac{t_{00}+t_{10}}{T}c\lambda^{tddb}$	$\frac{1}{2}c\lambda^{tddb}$
	$\lambda^{DiffFunc-b}$	$\frac{t_{01}+t_{00}}{T}c\lambda^{tddb}$	$\frac{1}{2}c\lambda^{tddb}$
Simplified manifestation	$\lambda^{Delay-1}$	$\begin{bmatrix} d_6\lambda_0^{hci}+d_2\lambda_1^{hci}+(d_3+d_7)\lambda_2^{hci} \\ +(d_2+d_7)\lambda_2^{em} \\ +(d_2+d_3+d_6+d_7)\lambda_3^{em} \\ +(d_2+d_3+d_6+d_7)\lambda_5^{em} \end{bmatrix}\frac{\tau}{T}$	$\frac{1}{8}\frac{d\tau}{T}\begin{pmatrix}\lambda_0^{hci}+\lambda_1^{hci}+2\lambda_2^{hci}+2\lambda_2^{em} \\ +4\lambda_3^{em}+4\lambda_5^{em}\end{pmatrix}$
	$\lambda^{Delay-0.5}$	$(d_3+d_6)\frac{\tau}{T}\lambda_1^{em}$	$\frac{1}{4}\frac{d\tau}{T}\lambda_1^{em}$
	$\lambda^{Stuck-at-0}$	$\frac{(2+c)t_{11}+t_{01}}{T}\lambda^{tddb}$	$\frac{3+c}{4}\lambda^{tddb}$
	$\lambda^{Stuck-at-1}$	$\frac{t_{00}+t_{01}+t_{10}}{T}\lambda^{tddb}$	$\frac{3}{4}\lambda^{tddb}$
	$\lambda^{DiffFunc-0.25}$	$\frac{t_{01}+t_{10}+2t_{00}}{T}c\lambda^{tddb}$	$c\lambda^{tddb}$

λ^{F} Failure rate of the circuit for failure manifestation F

λ_i^j Failure rate of segment i due to stress j

λ^{tddb} Failure rate due to TDDB stress

d_i Number of demands of type D_j that occurred within time window T.

d	Number of demand transitions of any type occurred during time window T.
c	Relation between the TDDB failure rate of nMOSFET and pMOSFET transistors.
τ	Duration of a demand transition D_i.
T	Duration of the time window where the measurement is performed.
$t_{00}, t_{01}, t_{10}, t_{11}$	Durations of logic combinations 00, 01, 10, 11 of the input signals within time window T.

3.3 Calculation of Failure Probabilities

3.3.1 Calculation of the Hardware Usage Profile

This section describes step 4 of Figure 3.1.

As discussed in Section 3.1, a device failure probability over time depends on the number of times the unit is demanded. For digital devices, especially a CPU, the way in which each unit is accessed depends heavily on the set of instructions (collectively called the software) it executes. Using the tools Synopsys VCS MX [61] and Synopsys Design Analyzer [62] makes it feasible to simulate the software execution against the hardware device's VHDL script.

Synopsys VCS MX is the industry's most comprehensive RTL verification solution in a single product, providing advanced bug-finding technologies, a built-in debug and visualization environment and support for all popular design and

verification languages including Verilog, VHDL, SystemVerilog and SystemC. Synopsys VCS MX provides a high-performance, high-capacity full language VHDL simulator. It is used to analyze, compile, and simulate design descriptions written in IEEE VHDL 1076-1993 [63], and provides a set of VHDL simulation and debugging features to validate the VHDL design descriptions. These features provide capabilities for source level debugging and simulation result viewing. The tool supports all levels of design descriptions but is optimized for the behavioral and register-transfer levels.

Synopsys Design Analyzer is a powerful analysis tool that provides synthesis control, design management, and design analysis in a graphical environment. With Synopsys Design Analyzer the user can perform various design set-up and analysis functions, as well as view and interact with the synthesized schematic.

The procedure to obtain the hardware usage profile of the circuit elements is described in Figure 3.18.

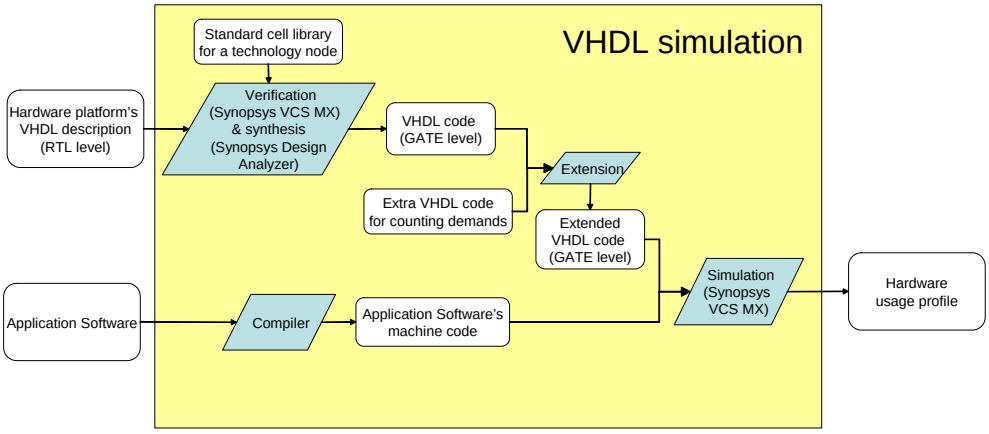


Figure 3.18 Description of the VHDL simulation step of the methodology

First, the functionality of the VHDL RTL script that describes the hardware platform (e.g., a basic platform including the CPU, the memory and the connection

between them) is verified by means of the Synopsys VCS MX simulator. The VHDL RTL script is then ready to be translated into an actual gate level netlist using Synopsys Design Analyzer. The process of converting the RTL description into a netlist for a given target technology is called logic synthesis. To produce the synthesized netlist, the synthesis tool requires the RTL code of the hardware devices and the cell libraries. The cell libraries provide information about all the available cells, including connectivity and functionality, timing, area, and corresponding symbol, among others. Some extra VHDL code is then inserted into the gate level VHDL scripts which will count the number of demands to the circuit elements (e.g., gates, flip-flops) during the VHDL simulation. On the other hand, the application software needs to be compiled into machine code. This code will be loaded into the system memory during the VHDL simulation. The gate level's VHDL scripts and the machine code are thus used by the Synopsys VCS MX simulator to simulate the execution of the application software on the computer system. The outcome of this simulation will consist of the software-specific hardware usage profile in terms of number of demands issued to the circuit elements under study.

An example of such a profile is provided in Figure 3.19. The "Circuit elements" axis represents a set of circuit elements under analysis. The "Time" axis defines the execution time of the system under a given software application (i.e., parameter T of Section 3.2), and the "Demands" axis provides the number of demands of each circuit element (i.e., parameter d of Section 3.2).

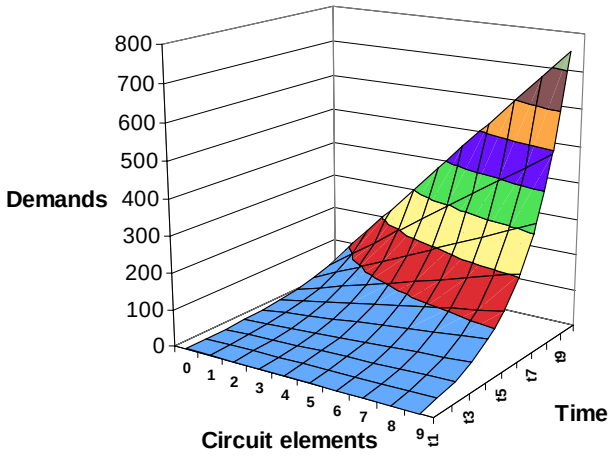


Figure 3.19 Software-specific hardware usage profile – an example

The number of demands of a circuit element will systematically increase monotonically over time. However, the speed of the increase will differ for different circuit elements and software applications [64].

3.3.2 Calculation of the Hardware Failure Probability Distributions

This section describes step 5 of Figure 3.1.

The *software-specific hardware failure probability* p_i^F of a circuit element i under failure manifestation F is calculated as follows:

$$p_i^F = 1 - e^{-\lambda_i^F T} \approx \lambda_i^F T \qquad (3.16)$$

where

λ_i^F the failure rate of failure manifestation λ_i^F of circuit element λ_i^F (equivalent to equation (3.7)),

T the duration of the time window where the measurement is being performed.

A hardware device of a computer system is composed of a set of circuit elements. For example, a register consists of a series of register bits (or flip-flops), and an ALU is composed of a netlist of logic gates. Therefore, the *software-specific hardware failure probability distribution* (or *profile*) P_i^F under failure manifestation F of a hardware device i composed of n circuit elements is as follows:

$$P_i^F = \left\{ p_1^F, p_2^F, ..., p_n^F \right\} \tag{3.17}$$

where

$p_1^F, ..., p_n^F$ the failure probabilities of circuit elements $1,2,...,n$ under failure manifestation F (equivalent to equation (3.16)).

Finally, the *combined software-specific hardware failure probability distribution* (or *profile*) P_i of a hardware device i under any failure manifestation is as follows:

$$P_i = \left\{ \sum_{F \in \Gamma_1} p_1^F, \sum_{F \in \Gamma_2} p_2^F, ..., \sum_{F \in \Gamma_n} p_n^F \right\} \tag{3.18}$$

where

$p_1^F, ..., p_n^F$ the failure probabilities of circuit elements $1,2,...,n$ under failure manifestation F (equivalent to equation (3.16)),

$\Gamma_1, \Gamma_2, ..., \Gamma_n$ the sets of all failure manifestations F of circuit elements $1,2,...,n$

An example of the combined software-specific hardware failure probability profile over time is provided in Figure 3.20. The "Hardware device" axis contains the set of circuit elements of the hardware device under analysis. The "Time" axis defines

the execution time of the system under a given software application (i.e., parameter T of equation (3.16)), and the Z-axis provides the software-specific hardware failure probability of each circuit element of the hardware device under any failure manifestation (i.e., parameter $\sum_{F \in \Gamma_1} p_i^F$ of equation (3.18)).

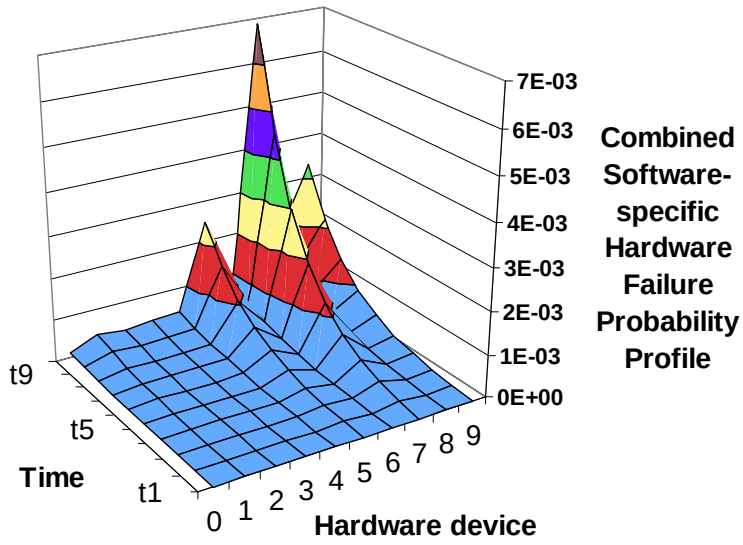


Figure 3.20 Combined software-specific hardware failure probability profile – an example

As in the case of the number of demands (see Figure 3.19), the failure probability of each circuit element of a hardware device will increase monotonically over time, while the speed of the increase will differ for different circuit elements [64].

Chapter 4 Calculation of Failure Probabilities

The methodology developed in Chapter 3 has been applied to an example system. The microprocessor considered in this system is the Zilog Z80 CPU, a CPU whose VHDL description was publicly available. There are no theoretical barriers to extend the proposed approach to other CPUs and devices such as memory or busses as long as their VHDL scripts are available.

4.1 System Description

The example system is configured as shown in Figure 4.1. The system consists of a Zilog Z80 microprocessor, and a RAM module. A set of control signals plus the data and address busses are also included to constitute a minimum system. The Z80 CPU is the pilot processor we used to demonstrate the methodology. The RAM module is used to store software program in machine code format.

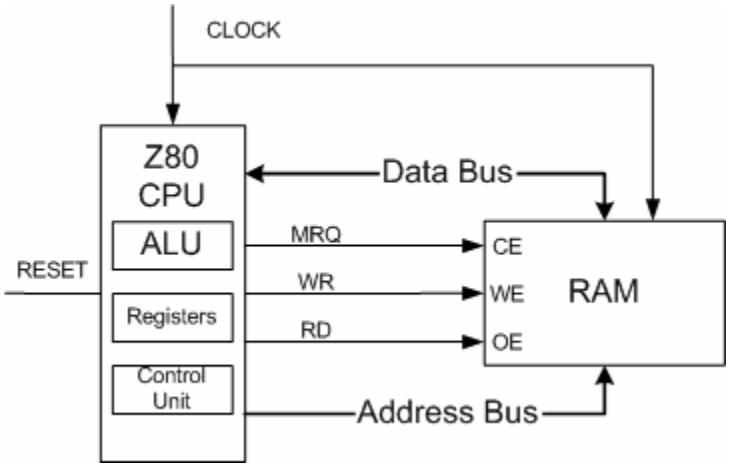


Figure 4.1 The Example Z80 Computer System

The Zilog Z80 microprocessor has been designed and manufactured by Zilog since 1976. It was widely used both in desktop and embedded computer designs and is one of the most popular CPUs of all time. Z80 was the heart of many computers like Spectrum, Partner, TRS703, Z-3. The Z80 microprocessor is an 8-bit CPU with a 16-bit address bus capable of direct access to 64k of memory space. The Z80 CPU can execute 158 different instructions.

The Z80 CPU contains 208 bits of read/write memory that are available to the programmer. The registers include two sets of six pairs of general-purpose registers (B, C, D, E, H, L, Bp, Cp, Dp, Ep, Hp, Lp) that may be used individually as 8-bit registers or in pairs as 16-bit registers. There are also two sets of accumulator and flag registers and six special-purpose registers. The special-purpose registers include Program Counter (PC), Stack Pointer (SP), Index Registers (IX and IY), Interrupt Page Address Register (I), and Memory Refresh Register (R). The other important but nonprogrammable register is the Instruction Register (IR) [65].

The Z80 VHDL script used in the case study was obtained from the T80 project at opencores.org [66]. The T80 is a configurable CPU core that supports Z80, 8080 and Gameboy instruction sets. The original T80 VHDL code is written in RTL level. The VHDL code is tailored to support the Z80 microprocessor instruction set only.

Besides the 22 user-programmable registers described above, the Z80 VHDL model has another 31 hidden registers, including the Instruction Register (IR), which are not visible to the programmer. The hidden registers are used by the CPU to store intermediate data during instruction execution.

Then the RTL level code is synthesized to logic gate level through Synopsys Design Analyzer using the vtvtlib25 standard cell library [58, 59]. The gate level netlist consists of about 5000 logic gates and flip-flops. The total number of transistors is on the order of 30000. The number of metal interconnects is in the neighborhood of 40000.

The gate level VHDL script is further modified to fit the simulation requirements. For example, as shown in Figure 4.2, the four new signals n2543, n2544, n2545, n2546 are introduced for the purpose of counting the number of demands for the logic gate U1250, which is of type ab_or_c_or_d.

```
architecture SYN_rtl of Z80_ALU is
-- original signal in the Z80 ALU gate level VHDL code
     signal n2543, n2544, n2545, n2546, …: std_logic;
   …
begin
   ALU_n2543 <= n2543;
   ALU_n2544 <= n2544;
   ALU_n2545 <= n2545;
   ALU_n2546 <= n2546;
-- ALU_n2543, ALU_n2543, ALU_n2543, ALU_n2543 are probes introduced to monitor the n2543, n2544, n2545, n2546 signals, which are connected to the inputs of gate ab_or_c_or_d.
   …
   U1250 : ab_or_c_or_d port map( ip1 => n2543, ip2 => n2544, ip3 => n2545, ip4
```

```
=> n2546, op => Q(0));

  …

end SYN_rtl;
```

Figure 4.2 Modified Z80 CPU script segment

4.2 Analysis of Failure Manifestations

The failure manifestations of different circuit elements, including D flip-flop used as CPU register bits and all the logic gates used by the ALU of the Z80 CPU are analyzed.

4.2.1 Analysis of the CPU Register Bits

The logic synthesis result indicates that D flip-flops from the vtvtlib25 standard cell library is used to represent the register bits of the Z80 CPU. The layout of the D flip-flop is shown in Figure 4.3. The corresponding schematic with input stimuli is illustrated in Figure 4.4.

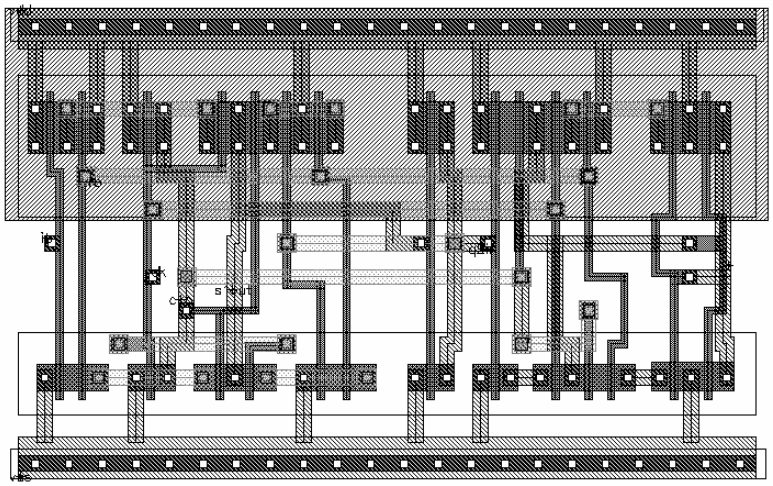


Figure 4.3 Circuit layout for D flip-flop

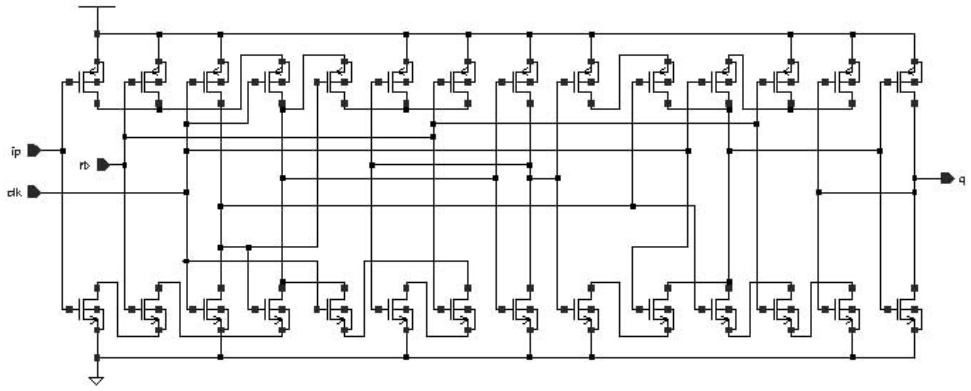


Figure 4.4 Circuit schematic for D flip-flop

The operation of the D flip-flop under a standardized input stimulus is shown in Figure 4.5, where the value of the output signal *Op* will be updated with the value of the input signal *Ip*1 at the rising edge of the clock signal.

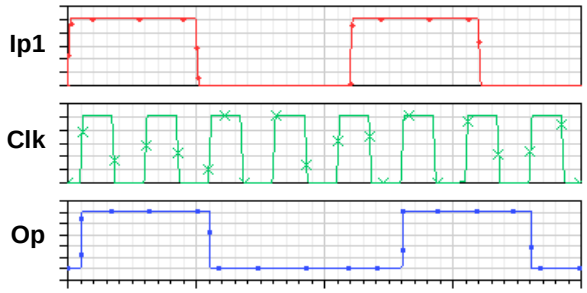
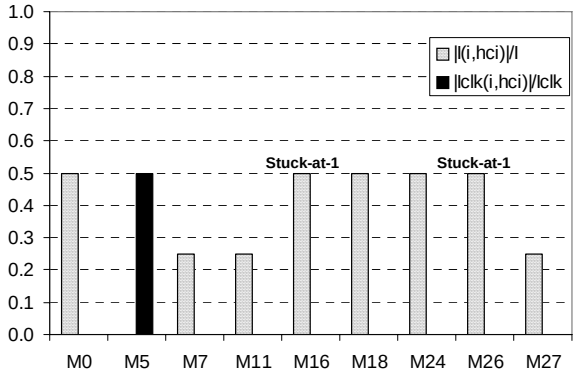


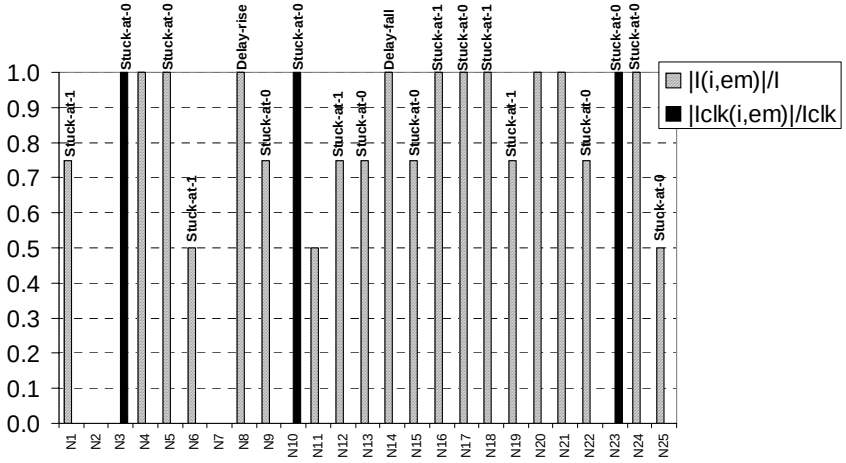
Figure 4.5 Transient response under normal operation

Figure 4.6 shows the results of the SPICE simulations performed on the D flip-flop to study its failure manifestations. The X-axis contains the different segments of the flip-flop ($M1, M2, ...N1, N2, ...$). For HCI and EM (Figure 4.6a and Figure 4.6b), the Y-axis represents the percentage of demand transitions of a segment either with respect to the asynchronous input signal ($|I(i, j)|/I$) or the clock signal controlling

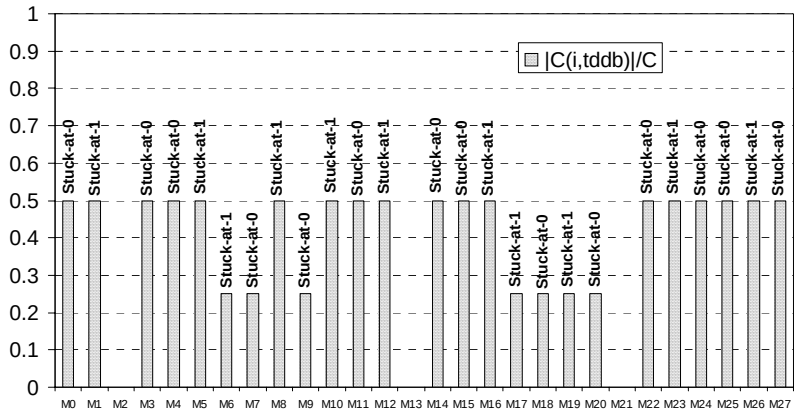
the flip-flop ($|I_{clk}(i,j)|/I_{clk}$). For TDDB (Figure 4.6c), it is shown instead the percentage of logic combinations of the asynchronous input signal under which the TDDB effect impacts a segment ($|C(i,j)|/C$). On the top of each column, we have included the failure manifestation observed for a segment (using the "detailed manifestation" notation). When nothing is indicated, it means that the failure of a segment has no effect on the circuit (i.e., the circuit behaves normally).



a) HCI stress



b) EM stress



c) TDDB stress

Figure 4.6 Z80 registers bits – Results of SPICE simulations.

The information provided by the SPICE simulations allows us to build the failure rate models of the flip-flop for each failure manifestation. Let's first focus on Figure 4.6a. The only failure manifestation observed at the circuit level is stuck-at-1, due to the failure of transistors $M16$ and $M26$ under the HCI effect. Using equations (3.7) and (3.13) and the demand percentages on Figure 4.6a, the failure rate for *stuck-at-1* type of failure is

$$\lambda^{stuck-at-1} = \frac{1}{2}\left(\lambda_{16}^{hci} + \lambda_{26}^{hci}\right)\frac{d\tau}{T}.$$

However, this expression is incomplete since we still need to include the contributions to the *stuck-at-1* failure rate from the other failure mechanisms. In Figure 4.6b, the failure of interconnections $N1$, $N6$, $N12$, $N16$, $N18$ and $N19$ under EM lead to *stuck-at-1*. Consequently, the contribution of the EM effect to the stuck-at-1 failure rate is given by expression

$\left(\frac{3}{4}\lambda_1^{em}+\frac{1}{2}\lambda_6^{em}+\frac{3}{4}\lambda_{12}^{em}+\lambda_{16}^{em}+\lambda_{18}^{em}+\frac{3}{4}\lambda_{19}^{em}\right)\frac{d\tau}{T}$. Finally, according to Figure 4.6c, the *stuck-at-1* failure manifestation under TDDB stress is provoked by nMOS transistors $M5$, $M16$ and $M26$ (for a time percentage equivalent to $3/2$) and pMOS transistors $M1$, $M6$, $M8$, $M10$, $M12$, $M17$, $M19$ and $M23$ (for a time percentage equivalent to $13/4$). The contribution of the TDDB stress to the *stuck-at-1* failure rate is $\frac{6+13c}{4}\lambda^{tddb}$. Combining these three partial results, the final expression for the *stuck-at-1* failure rate of the flip-flop is as follows:

$$\lambda^{stuck-at-1}=\left[\frac{1}{2}\left(\lambda_{16}^{hci}+\lambda_{26}^{hci}\right)+\left(\frac{3}{4}\lambda_1^{em}+\frac{1}{2}\lambda_6^{em}+\frac{3}{4}\lambda_{12}^{em}+\lambda_{16}^{em}+\lambda_{18}^{em}+\frac{3}{4}\lambda_{19}^{em}\right)\right]\frac{d\tau}{T}+\frac{6+13c}{4}\lambda^{tddb} \quad (4.1)$$

The *stuck-at-0* failure rate can be calculated in a similar way (see Table 4.1). Also, as shown in Figure 4.6b, transistors $M8$ and $M14$ lead to the *delay-rise* and *delay-fall* manifestations, respectively. The corresponding failure rates are given by expressions $\lambda^{Delay-rise}=d\tau\lambda_8^{em}/T$ and $\lambda^{Delay-fall}=d\tau\lambda_{14}^{em}/T$.

The reliability models for the flip-flop are summarized in Table 4.1 using the "simplified manifestation" notation.

Table 4.1 Reliability models for the flip-flop circuit element of the Z80 CPU

$\lambda^{Delay-[0,1]}$	$\lambda^{Stuck-at-0}$	$\lambda^{Stuck-at-1}$
$\left(\lambda_8^{em}+\lambda_{14}^{em}\right)\frac{d\tau}{T}$	$\left[\begin{array}{l}\left(\lambda_5^{em}+\frac{3}{4}\lambda_9^{em}+\frac{3}{4}\lambda_{13}^{em}+\frac{3}{4}\lambda_{15}^{em}\right.\\ \left.+\lambda_{17}^{em}+\frac{3}{4}\lambda_{22}^{em}+\lambda_{24}^{em}+\frac{1}{2}\lambda_{25}^{em}\right)d\\ +\left(\lambda_3^{em}+\lambda_{10}^{em}+\lambda_{23}^{em}\right)d_{clk}\end{array}\right]\frac{\tau}{T}$ $+\left(\frac{10+13c}{4}\right)\lambda^{tddb}$	$\left[\begin{array}{l}\frac{1}{2}\left(\lambda_{16}^{hci}+\lambda_{26}^{hci}\right)\\ +\left(\begin{array}{l}\frac{3}{4}\lambda_1^{em}+\frac{1}{2}\lambda_6^{em}+\frac{3}{4}\lambda_{12}^{em}\\ +\lambda_{16}^{em}+\lambda_{18}^{em}+\frac{3}{4}\lambda_{19}^{em}\end{array}\right)\end{array}\right]\frac{d\tau}{T}$ $+\frac{6+13c}{4}\lambda^{tddb}$

4.2.2 Analysis of Fault Models for Combinational Logic Elements

In Z80 CPU, all the arithmetic and logical instructions are executed in the Arithmetic Logic Unit (ALU). The logic synthesis results indicate that ALU is composed of pure combinational logic gates. There are no registers used in the ALU. In this work, all the combinational logic gates in the ALU are analyzed. The analysis can be extended to other combinational logic circuits of the microprocessor.

The logic synthesis results show that the ALU contains 461 logic gates of different types. These types are listed in Table 4.2.

Table 4.2 Different type of logic gates used in the ALU of the Z80 CPU

	# of Input Signals	Logic function
INV_1	1	$\overline{a}$
NOR2_1	2	$\overline{a+b}$
OR2_1	2	$a+b$
NAND2_1	2	$\overline{ab}$
AND2_1	2	ab
XOR2_1	2	$a\overline{b}+\overline{a}b$
XNOR2_1	2	$\overline{a\overline{b}+\overline{a}b}$
ABorC	3	$ab+c$
MUX2_1	3	$\overline{a}b+ac$
NOR3_1	3	$\overline{a+b+c}$
NAND3_1	3	$\overline{abc}$
NAND4_1	4	$\overline{abcd}$
NOR4_1	4	$\overline{a+b+c+d}$
OR4_1	4	$a+b+c+d$
ABorCorD	4	$ab+c+d$
NOT (ABorCorD)	4	$\overline{ab+c+d}$

Figure 4.7 and Figure 4.8 illustrate the standardized input signal stimuli used for the 3- and 4-inputs gates. The signal stimulus for the 2-inputs gates appears in Figure 3.6. Note that the various input signal stimuli follow the criteria stated in Section 3.1.1.

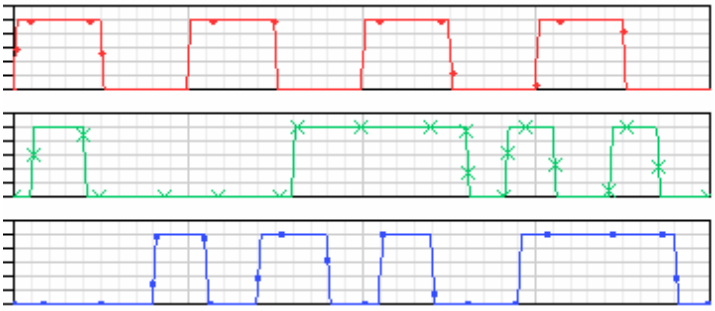


Figure 4.7 Standardized input signal stimuli for 3-inputs gates

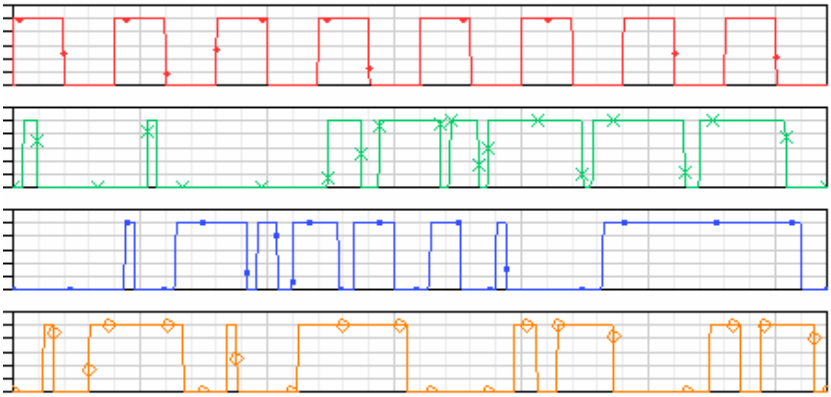


Figure 4.8 Standardized input signal stimuli for 4-inputs gates

The reliability models for the logic gates are developed in a similar way as explained in 3.2 for the AND2_1 gate. The results of the SPICE simulations for the logic gates are provided in Appendix A.

A total of 120 different types of failure manifestations have been observed for the ALU gates (see Appendix A):

66 types correspond to delays of the transitions of the output signal triggered at specific combinations of the inputs. From them, 32 impact the falling transitions of the output, while 34 impact the rising transitions.

51 types correspond to behavioral changes of the gates leading to different logic functions.

Two types correspond to the *stuck-at-0* and *stuck-at-1* manifestation types.

In order to cover all these different types of failure manifestations, we have developed more than 250 reliability models for the ALU gates. Using the "simplified manifestation" notation, it is possible to combine them and reduce these figures to 90 models for 19 manifestation types (namely, $Delay-1$, $Delay-0.17$, $Delay-0.25$, $Delay-0.33$, $Delay-0.38$, $Delay-0.4$, $Delay-0.5$, $Delay-0.6$, $Delay-0.67$, $Delay-0.75$, $Stuck-at-0$, $Stuck-at-1$, $DiffFunc-0.06$, $DiffFunc-0.13$, $DiffFunc-0.19$, $DiffFunc-0.25$, $DiffFunc-0.38$, $DiffFunc-0.5$, $DiffFunc-0.63$). Further, we have merged the various types of delays and different functions into a single category. It leads to 49 models for four manifestation types, as shown in Table 4.3.

Table 4.3 Reliability models for logic gates used in the ALU

	λ^{Delay}	$\lambda^{Stuck-at-0}$	$\lambda^{Stuck-at-1}$	$\lambda^{DiffFunc}$
INV_1	$\frac{1}{2}\frac{d\tau}{T}\left(\lambda_0^{hci}+2\lambda_1^{em}+2\lambda_2^{em}+2\lambda_3^{em}\right)$	$\frac{1}{2}c\lambda^{tddb}$	$\frac{1}{2}\lambda^{tddb}$	
NOR2_1	$\frac{1}{2}\frac{d\tau}{T}\left(\lambda_1^{em}+\lambda_2^{em}\right)$ $\frac{1}{8}\frac{d\tau}{T}\left(\lambda_0^{hci}+\lambda_1^{hci}+2\lambda_3^{em}+2\lambda_4^{em}\right)$	$\frac{3}{4}c\lambda^{tddb}$		λ^{tddb}
OR2_1	$\frac{1}{4}\frac{d\tau}{T}\left(\lambda_2^{hci}+2\lambda_1^{em}+2\lambda_2^{em}+\lambda_5^{em}\right)$ $\frac{1}{8}\frac{d\tau}{T}\left(\lambda_0^{hci}+\lambda_1^{hci}+2\lambda_4^{em}\right)$	$\frac{3}{4}c\lambda^{tddb}$	$\frac{3c+1}{4}\lambda^{tddb}$	λ^{tddb}
NAND2_1	$\frac{1}{8}\frac{d\tau}{T}\left(\lambda_0^{hci}+\lambda_1^{hci}+4\lambda_3^{em}+4\lambda_4^{em}\right)$ $\frac{1}{4}\frac{d\tau}{T}\left(\lambda_1^{em}+\lambda_2^{em}\right)$		$\frac{3}{4}\lambda^{tddb}$	$c\lambda^{tddb}$

	λ^{Delay}	$\lambda^{Stuck-at-0}$	$\lambda^{Stuck-at-1}$	$\lambda^{DiffFunc}$
XOR2_1	$\frac{1}{4}\frac{d\tau}{T}\begin{pmatrix}\lambda_3^{hci}+2\lambda_1^{em}+2\lambda_4^{em}\\+2\lambda_5^{em}+2\lambda_7^{em}\end{pmatrix}$ $\frac{1}{2}\frac{d\tau}{T}\lambda_6^{em}$ $\frac{1}{8}\frac{d\tau}{T}\begin{pmatrix}2\lambda_0^{hci}+\lambda_5^{hci}+\lambda_7^{hci}\\+\lambda_9^{hci}+\lambda_{11}^{hci}+2\lambda_2^{em}\\+2\lambda_3^{em}+6\lambda_8^{em}\end{pmatrix}$			$\frac{9+10c}{4}\lambda^{tddb}$
NOR3_1	$\frac{d\tau}{T}\left(\frac{1}{6}\lambda_4^{em}+\frac{1}{4}\lambda_1^{em}\right)$ $+$ $\frac{1}{24}\frac{d\tau}{T}\left(\lambda_0^{hci}+\lambda_3^{hci}+\lambda_5^{hci}+2\lambda_3^{em}\right)$ $+\frac{1}{4}\frac{d\tau}{T}\lambda_2^{em}$	$\frac{7}{8}c\lambda^{tddb}$		$\frac{3}{2}\lambda^{tddb}$
MUX2_1	$\frac{1}{12}\frac{d\tau}{T}\left(3\lambda_{11}^{hci}+5\lambda_5^{em}\right)$ $+\left(\frac{1}{6}\lambda_0^{hci}+\frac{1}{3}\lambda_4^{em}\right)\frac{d\tau}{T}$ $+$ $\frac{1}{24}\frac{d\tau}{T}\begin{pmatrix}\lambda_3^{hci}+2\lambda_5^{hci}+2\lambda_7^{hci}\\+\lambda_9^{hci}+12\lambda_1^{em}+6\lambda_2^{em}\\+4\lambda_8^{em}\end{pmatrix}$ $+\frac{1}{2}\frac{d\tau}{T}\lambda_7^{em}$	$\frac{1}{2}c\lambda^{tddb}$	$\frac{1}{2}\lambda^{tddb}$	$\frac{1+2c}{2}\lambda^{tddb}$ $+\lambda^{tddb}$ $+$ $\left(1+\frac{1}{2}c\right)\lambda^{tddb}$ $+\frac{1}{2}c\lambda^{tddb}$ $+\frac{1}{2}c\lambda^{tddb}$
ABorC	$\frac{1}{24}\frac{d\tau}{T}\left(5\lambda_7^{hci}+3\lambda_3^{em}+4\lambda_5^{em}+5\lambda_7^{em}\right)$ $+$ $\frac{1}{24}\frac{d\tau}{T}\left(\lambda_0^{hci}+\lambda_3^{hci}+\lambda_1^{em}+2\lambda_2^{em}\right)$ $+\frac{3}{24}\frac{d\tau}{T}\lambda_5^{hci}$	$\frac{5}{8}c\lambda^{tddb}$	$\frac{3}{8}\lambda^{tddb}$	$\left(\frac{1}{2}+c\right)\lambda^{tddb}$ $+\frac{7}{8}\lambda^{tddb}$ $+\frac{3}{8}c\lambda^{tddb}$

	λ^{Delay}	$\lambda^{Stuck-at-0}$	$\lambda^{Stuck-at-1}$	$\lambda^{DiffFunc}$
NAND3_1	$\left(\begin{array}{l}\frac{1}{24}\lambda_0^{hci}+\frac{1}{24}\lambda_1^{hci}\\+\frac{1}{24}\lambda_2^{hci}+\frac{1}{4}\lambda_3^{em}+\frac{1}{4}\lambda_4^{em}\end{array}\right)\frac{d\tau}{T}$ $+\frac{1}{12}\frac{d\tau}{T}\lambda_1^{em}$ $+\frac{1}{6}\frac{d\tau}{T}\lambda_2^{em}$		$\frac{7}{8}\lambda^{tddb}$	$\frac{3}{2}c\lambda^{tddb}$
XNOR2_1	$\frac{1}{2}\frac{d\tau}{T}\left(\lambda_{13}^{hci}+\lambda_7^{em}\right)$ $+$ $\frac{1}{4}\frac{d\tau}{T}\left(\begin{array}{l}\lambda_0^{hci}+\lambda_3^{hci}\\+2\lambda_1^{em}+2\lambda_6^{em}+2\lambda_9^{em}\end{array}\right)$ $+\frac{1}{8}\frac{d\tau}{T}\left(\begin{array}{l}\lambda_5^{hci}+\lambda_7^{hci}+\lambda_9^{hci}\\+\lambda_{11}^{hci}+6\lambda_2^{em}+2\lambda_3^{em}\end{array}\right)$	$\frac{1}{2}c\lambda^{tddb}$	$\frac{1}{2}\lambda^{tddb}$	$\frac{10+9c}{2}\lambda^{tddb}$
NAND4_1	$\left(\begin{array}{l}\frac{3}{64}\lambda_0^{hci}+\frac{1}{64}\lambda_1^{hci}+\frac{1}{32}\lambda_2^{hci}\\+\frac{1}{64}\lambda_3^{hci}+\frac{1}{8}\lambda_4^{em}+\frac{1}{8}\lambda_5^{em}\end{array}\right)\frac{d\tau}{T}$ $+\frac{1}{32}\frac{d\tau}{T}\left(\lambda_1^{em}+\lambda_3^{em}\right)$ $+\frac{1}{16}\frac{d\tau}{T}\lambda_2^{em}$		$\frac{15}{16}\lambda^{tddb}$	$2c\lambda^{tddb}$
NOR4_1	$\frac{1}{8}\frac{d\tau}{T}\lambda_1^{em}$ $+\frac{1}{64}\frac{d\tau}{T}\left(\begin{array}{l}\lambda_0^{hci}+\lambda_3^{hci}+\lambda_5^{hci}\\+\lambda_7^{hci}+\lambda_3^{em}+2\lambda_5^{em}\end{array}\right)$ $+\frac{1}{16}\frac{d\tau}{T}\lambda_4^{em}$ $+\frac{1}{8}\frac{d\tau}{T}\lambda_2^{em}$	$\frac{15}{16}c\lambda^{tddb}$		$2\lambda^{tddb}$
OR4_1	$\frac{1}{16}\frac{d\tau}{T}\left(\lambda_9^{hci}+2\lambda_1^{em}+2\lambda_2^{em}\right)$ $+\frac{1}{64}\frac{d\tau}{T}\left(\begin{array}{l}\lambda_0^{hci}+\lambda_3^{hci}\\+\lambda_5^{hci}+\lambda_7^{hci}+2\lambda_4^{em}\\+4\lambda_5^{em}+2\lambda_6^{em}\end{array}\right)$	$\frac{15}{16}c\lambda^{tddb}$	$\frac{1+15c}{16}\lambda^{tddb}$	$2\lambda^{tddb}$

	λ^{Delay}	$\lambda^{Stuck-at-0}$	$\lambda^{Stuck-at-1}$	$\lambda^{DiffFunc}$
NOT (ABorCor D)	$\frac{1}{4}\frac{d\tau}{T}\left(\lambda_3^{em}+\lambda_4^{em}\right)$ $+\frac{1}{64}\frac{d\tau}{T}\begin{pmatrix}2\lambda_5^{hci}+3\lambda_7^{hci}\\+10\lambda_1^{em}+10\lambda_2^{em}\\+6\lambda_6^{em}+5\lambda_7^{em}\end{pmatrix}$ $+\left(\frac{1}{64}\lambda_0^{hci}+\frac{1}{64}\lambda_3^{hci}+\frac{1}{16}\lambda_5^{em}\right)\frac{d\tau}{T}$			$\frac{8+17c}{16}\lambda^{tddb}$ $+\frac{1}{2}c\lambda^{tddb}$ $+\frac{23}{16}\lambda^{tddb}$
ABorCor D	$\frac{1}{8}\frac{d\tau}{T}\left(\lambda_9^{hci}+2\lambda_3^{em}+2\lambda_5^{em}\right)$ $+\left(\frac{1}{64}\lambda_0^{hci}+\frac{1}{64}\lambda_3^{hci}+\frac{1}{16}\lambda_7^{em}\right)\frac{d\tau}{T}$ $+\frac{1}{64}\frac{d\tau}{T}\begin{pmatrix}2\lambda_5^{hci}+3\lambda_7^{hci}+10\lambda_1^{em}\\+10\lambda_2^{em}+6\lambda_8^{em}+5\lambda_9^{em}\end{pmatrix}$	$\frac{13}{16}c\lambda^{tddb}$	$\frac{3}{16}\lambda^{tddb}$	$\left(\frac{1}{2}+c\right)\lambda^{tddb}$ $+\frac{9}{16}c\lambda^{tddb}$ $+\frac{19}{16}\lambda^{tddb}$

4.3 Lifetime Model Parameters Calculation

In order to calculate the failure rates of the reliability models developed in the previous section, we need to obtain all the lifetime model parameters, especially the model prefactors for all three failure mechanisms.

The parameters in the lifetime models of different failure mechanisms are estimated from accelerated testing experiments. For example, for the EM failure mechanism, the current density exponent and thermal activation energy in equation (2.2) can be estimated by testing the metal interconnect at different current and temperature level, respectively. These parameter values usually remain unchanged at different foundries if the manufacturing processes are almost identical.

However, the lifetime model prefactors will depend on the quality control and the material properties. Even though they can also be obtained from accelerated testing, the actual value varies from one foundry to another. The values of the prefactors are set by each semiconductor foundry to keep the lifetime of semiconductor devices within a specific reliability target. A general assumption is that the foundry will do their best to make sure there is no dominating failure mechanism, which indicates that the failure rates for different mechanisms are roughly the same. Without conducting device life testing, we can estimate the model prefactors from the above assumption.

The reliability target for a semiconductor chip is usually about 30 years [60], which translates to a FIT value of around 3,800. This FIT value is for the device nominal operating conditions, which means the device is operating under dynamic stress conditions. With the assumption of no dominant failure mechanism, the FIT value is evenly distributed for the three failure mechanisms, which means each failure mechanism has a FIT value of about 1,300.

The lifetime model prefactors of all three failure mechanisms can be calculated based on the above conditions for the case of the Z80 microprocessor. The results of logic synthesis with the vtvtlib25 standard cell library show that the total number of logic gates and flip-flops is on the order of 5,000. The total number of transistors is about 30,000. The number of metal interconnects is in the neighborhood of 40,000. The nominal voltage bias is 2.5V, and the temperature is 300K.

4.3.1 TDDB Lifetime Model Prefactor

Based on the above assumption, the total failure rate for the TDDB mechanism is

$$\lambda_{TDDB} = 1300FIT = 1.3 \times 10^{-6} hr^{-1}$$

From the standard inputs pattern simulation, the average duty factor for the TDDB effect is about 0.5, which means that a transistor is stressed by the TDDB mechanism about 50% of the time during circuit operation. However, in the TDDB lifetime model, the failure rate is calculated assuming the gate oxide is stressed constantly. Therefore, the failure rate we should use in the lifetime model is the failure rate at operating condition divided by the average duty factor. The adjusted failure rate is

$$\lambda'_{TDDB} = \frac{\lambda_{TDDB}}{0.5} = 2.6 \times 10^{-6} hr^{-1}$$

The total lifetime for TDDB is

$$t_f'(TDDB) = \frac{1}{\lambda'_{TDDB}} = 3.85 \times 10^5 hr$$

The channel length of a MOSFET in the vtvtlib25 cell library is 240nm. The channel width of a nMOSFET is 840nm, while the pMOSFET channel width is 1680nm. The average area of the gate oxide of a MOSFET is the average of a nMOSFET and pMOSFET gate area.

$$A_{MOSFET} = 240 \times \frac{840 + 1680}{2} \times 10^{-18} m^2 = 3.02 \times 10^{-13} m^2$$

At nominal operating conditions, with the typical value $\beta = 1.64$, $a = -78$, $b = 0.081$, $c = 8.81 \times 10^3$, and $d = -7.75 \times 10^5$ in the lifetime model [51], the model prefactor is

$$A_{TDDB} = \frac{t_f'(TDDB) \times 30000}{\left(\frac{1}{A_{MOSFET}}\right)^{\frac{1}{\beta}} V_{gs}^{a+bT} \exp\left(\frac{c}{T} + \frac{d}{T^2}\right)} = 6.06 \times 10^{14}$$

4.3.2 HCI Lifetime Model Prefactor

With the same assumptions as before, the total failure rate for the HCI effect is

$$\lambda_{HCI} = 1.3 \times 10^{-6} hr^{-1}$$

The average duty factor for HCI effect is about 0.01. Based on an analysis similar to the one conducted for the TDDB failure rate, the adjusted total failure rate for the HCI mechanism is

$$\lambda'_{HCI} = \frac{\lambda_{HCI}}{0.01} = 1.3 \times 10^{-4} hr^{-1}$$

The total lifetime for HCI is

$$t_f'(HCI) = \frac{1}{\lambda'_{HCI}} = 7.69 \times 10^3 hr$$

The nMOSFET transistor channel width is

$$W = 8.4 \times 10^{-7} m$$

The average substrate current is about

$$I_{sub} = 1.7 \times 10^{-8} A$$

With the typical value of $n = 1.5$ and $E_{aHCI} = -0.15eV$ in the lifetime model [51], The lifetime prefactor is

$$A_{HCI} = \frac{t_f'(HCI) \times 15000}{\left(\frac{I_{sub}}{W}\right)^{-n} \exp\left(\frac{E_{aHCI}}{\kappa T}\right)} = 1.10 \times 10^{8}$$

4.3.3 EM Lifetime Model Prefactor

The total failure rate for the EM effect is

$$\lambda_{EM} = 1.3 \times 10^{-6} hr^{-1}$$

The average duty factor for the EM mechanism is about 0.03 as observed from the input pattern simulation. Similarly, the adjusted total failure rate for EM is

$$\lambda'_{EM} = \frac{\lambda_{EM}}{0.03} = 4.33 \times 10^{-5} hr^{-1}$$

The total lifetime for EM is

$$t_f'(EM) = \frac{1}{\lambda'_{EM}} = 2.31 \times 10^{4} hr$$

The interconnect cross section area is

$$A = 2.88 \times 10^{-13} m^2$$

The average value of current flow in the interconnect is

$$I = 6.5 \times 10^{-5} A$$

With the typical value of $n = 2$ and $E_{aEM} = 0.8eV$ in the lifetime model [51],

The EM lifetime model prefactor is

$$A_{EM} = \frac{t_f'(EM) \times 40000}{\left(\frac{I}{A}\right)^{-n} \exp\left(\frac{E_{aEM}}{\kappa T}\right)} = 1.73 \times 10^{12}$$

With the actual values of the lifetime model prefactors, the failure rates for all the reliability models can be calculated.

4.4 Usage and Failure Probability Distribution Profiles

The calculation of the hardware failure probability of hardware components using the reliability models developed in Section 4.2 requires hardware usage information for the software under consideration. Once the usage profile is obtained, we can build the hardware failure probability distribution profile, which can be used to study the software reliability induced by hardware failures.

The software code used here is a 32-bit floating point division program that does the division of two single precision floating point numbers and returns the quotient also as a single precision floating point number. Division is the most complicated operation compared with other elemental operations, such as addition, subtraction and multiplication. Since Z80 is an 8-bit CPU, a 32-bit floating point division operation is quite complex to implement on this microprocessor. The software code is first written in C language and compiled into Z80 assembly code using SDCC compiler, which is a retargettable, optimizing ANSI - C compiler that targets the Intel 8051, Maxim 80DS390 and the Zilog Z80 based MCUs [67]. The assembly program contains 968 lines of code.

Figure 4.9 shows the usage profile of the registers of the Z80 CPU at the end of the execution of the floating point division program with a particular operational profile used. The X axis denotes the index of all the Z80 registers, including the user programmable registers and the hidden ones. The indexes of all the user programmable registers are listed in Table 4.4. The hidden registers are indexed from 23 to 53. The Y axis denotes the indexes of register bits in the registers. The number of demands at the end of the software execution is displayed in the Z axis.

Table 4.4 Index of user programmable registers

Name	ACC	Ap	F	Fp	B	Bp	C	Cp	D	Dp	E
Index	1	2	3	4	5	6	7	8	9	10	11
Name	Ep	H	Hp	L	Lp	I	R	IX	IY	SP	PC
Index	12	13	14	15	16	17	18	19	20	21	22

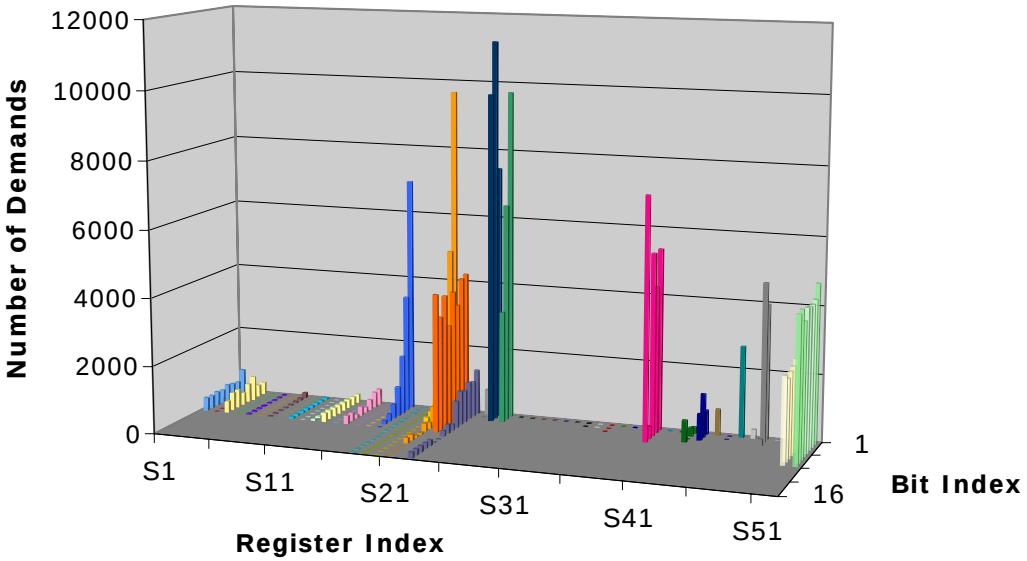


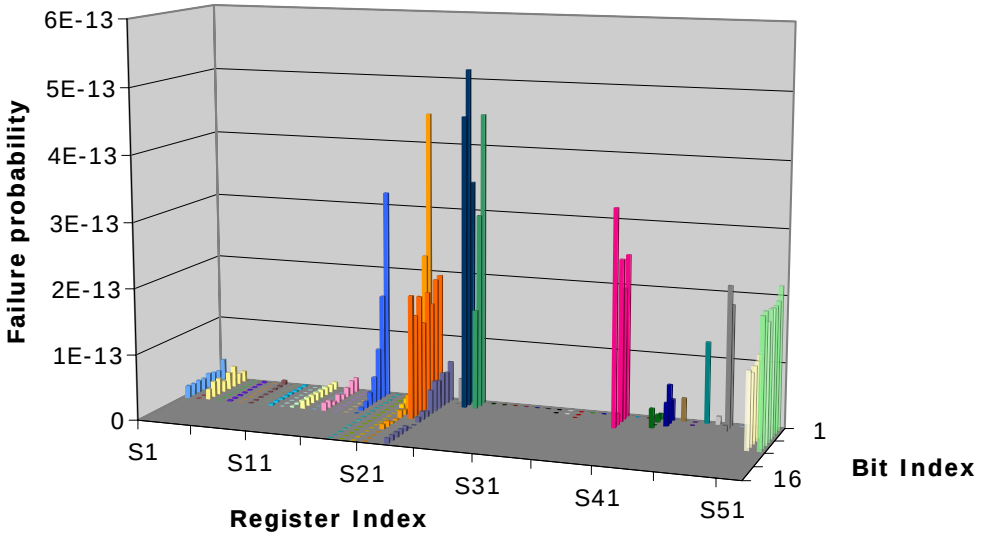
Figure 4.9 Usage profile for all the CPU registers for the division software program

We can see that the usage varies for different registers. Some of them are used quite frequently, while there are several registers not used at all during software execution. Some of the hidden register bits have higher demand than the user programmable ones. The usage of different register bits within a particular register also varies. For registers that are used to store data information, the usage is roughly

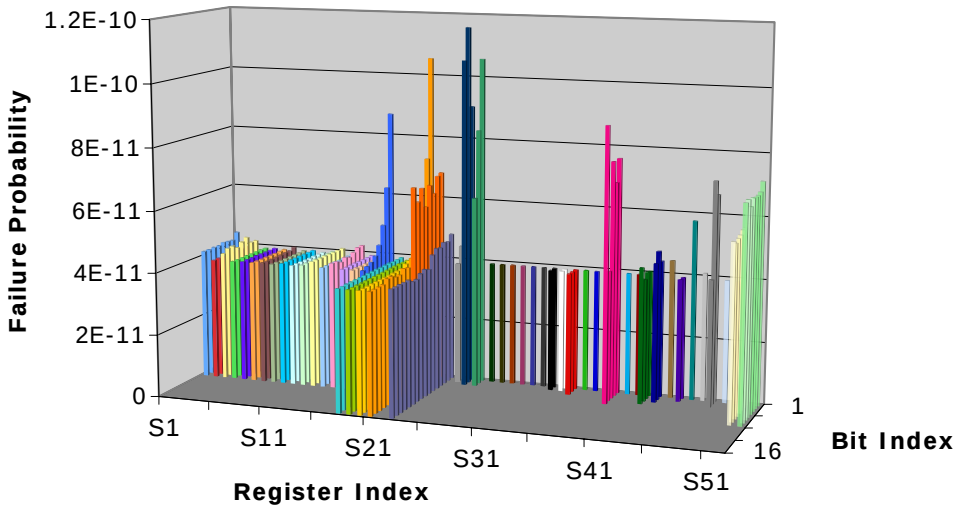
randomly distributed. But for the registers storing address information, for example the program counter (PC) register, the lower bits are used more often than the higher bits.

Quantitatively, the total number of demands to the registers at the end of the software execution is about 200000. Out of the 53 registers, 8 registers (e.g., PC, R) are demanded within [31000, 10001], 11 registers (e.g., F, ACC, L, SP, H) within [10000, 1001], 9 registers (e.g., E, C, B, D) within [1000, 101], 3 registers (e.g., IX, IY) within [100, 1], and 22 registers (e.g., Ap, Fp, Bp, Cp, Dp, Ep, Hp, Lp, I) are not used. The coefficients of variation (i.e. the ratio of the standard deviation to the mean) for the usage of different register bits within the registers are also calculated. The user programmable registers with the highest coefficients of variation are SP (2.17) and PC (1.92), and the lowest are E (0.08), and H (0.01).

Figure 4.10 shows the failure probability distributions of the registers of the Z80 CPU at the end of the execution of the division application with the particular operational profile used.



a) Delay failure manifestation.



b) Stuck-at failure manifestation.

Figure 4.10 Failure probability distributions for all the CPU registers for the division software program

The software-specific hardware failure profile of the Z80 registers for the *delay* manifestation is shown in Figure 4.10a. The total *delay* probability of all registers at the end of the software execution is about 9E-12. Out of the 53 registers, 2 registers (all hidden registers) have probability within [2E-12, 1E-12], 14 registers (e.g., PC, R, F, ACC) within [10E-13, 1E-13], 10 registers (e.g., L, SP, H, E, C, B) within [10E-14, 1E-14], 3 registers (e.g., D, IX) within [10E-15, 1E-15], 2 registers (e.g., IY) within [10E-16, 1E-16], and 22 registers (e.g., Ap, Fp, Bp, Cp, Dp, Ep, Hp, Lp, I) are not affected. The user programmable registers with the highest coefficients of variation of *delay* probability are SP (2.17) and PC (1.92), and the lowest are E (0.08), H (0.01).

The software-specific hardware failure profile of the Z80 registers for the *stuck-at* manifestation is shown in Figure 4.10b. The total *stuck-at* probability of all registers at the end of the software execution is about 1E-08. Out of the 53 registers, 32 registers (including all the user programmable registers in the following order: PC, SP, IX, IY, R, F, ACC, L, H, E, C, B, Ap, Fp, Bp, Cp, Dp, Ep, Hp, Lp, I, D) have failure probability within [7E-10, 1E-10], and the other 21 registers (all hidden) within [10E-11, 3E-11]. The user programmable registers with the highest coefficients of variation of *stuck-at* probability are PC (0.37) and R (0.34), and the lowest are registers Lp (1.48E-08) and I (1.48E-08).

As can be seen, the results for the *delay* manifestations follow the same pattern as the number of demands (Figure 4.9). This means the *delay* failure probability is highly dependent on the hardware usage induced by the software execution. On the contrary, the *stuck-at* failure probability is always positive even when the register bits are not used. This is due to the fact that the *stuck-at* manifestation is partly triggered

by the TDDB effect, and the latter is stressing the semiconductor devices even when the device is under static usage conditions (as explain in Section 3.1.1).

Figure 4.11 illustrates the combined failure probability distribution profile of the Z80 CPU registers. The failure profile is similar to the *stuck-at* failure profile in Figure 4.10b, since the *stuck-at* manifestation is dominant over the delay manifestation by a difference of several orders of magnitude.

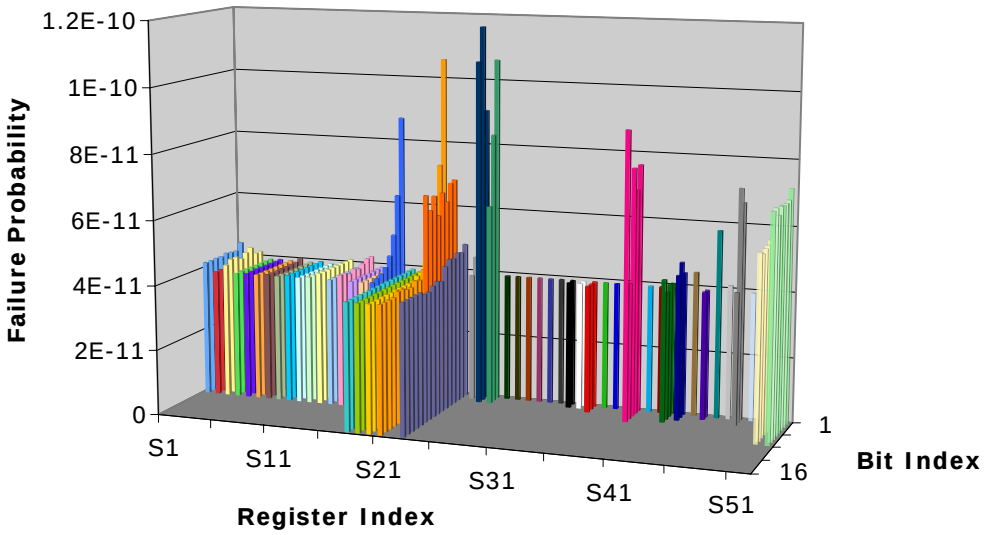


Figure 4.11 Combined failure probability distribution for all the registers for the division software program

Figure 4.12 shows the usage information of the ALU gates of the Z80 CPU for the division application software. All the gates are included in a possible ALU layout in the 2-D map. We use a gray scale to indicate usage information: the deeper the colors in the map, the higher the number of demands during the software execution. It can be seen that the usage varies for different gates in the ALU.

The total number of demands for all the gates is about 4E10. Out of the 461 logic gates, 128 are demanded within [6E4, 1E4], 294 gates within [10E3, 1E3], 27 gates within [10E2, 1E2], and 12 are not demanded. Per type, the most demanded gate types are mux2_1 and nor2_1 with the number of demands within [75000, 80000], and the less demanded are or2_1 and or4_1 (less than 30,000). For the gates of the same type, the highest demand variance is experimented by types nor4_1 and inv_1, while the lowest variance is experienced by nand2_1 and xnor2_1 (with coefficients of variation of 1.45, 1.02, 0.42 and 0.06, respectively). Per individual gates of the same type, the gates with the maximum number of demands are of type nor4_1 (with one gate with 55,681 demands) and mux2_1 (with one gate with 45,271 demands), and the minimum number of demands are of type ABorC, inv_1, nor2_1 and or2_1 (with at least one gate with 0 demand). The highest demand average are observed in gates of type or4_1 (15,460 demands) and not_ab_or_c_or_d (15,454 demands), while the lowest in types inv_1 (4,070 demands) and nand2_1 (3,150 demands).

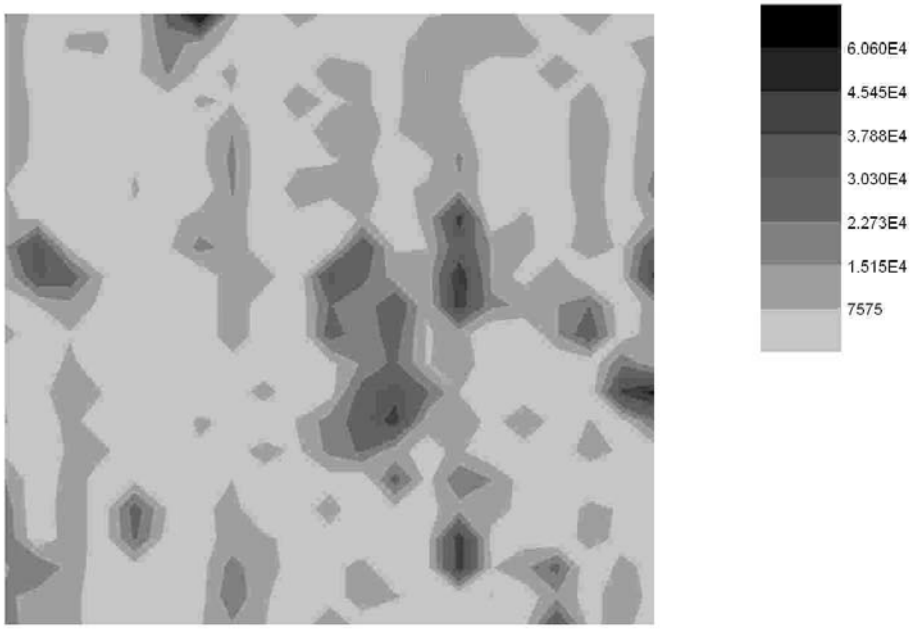
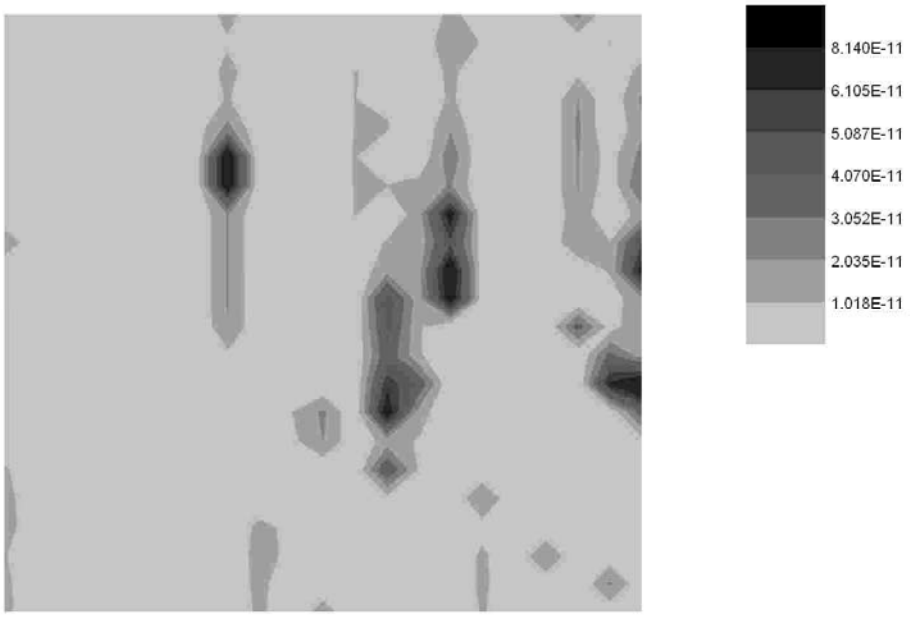
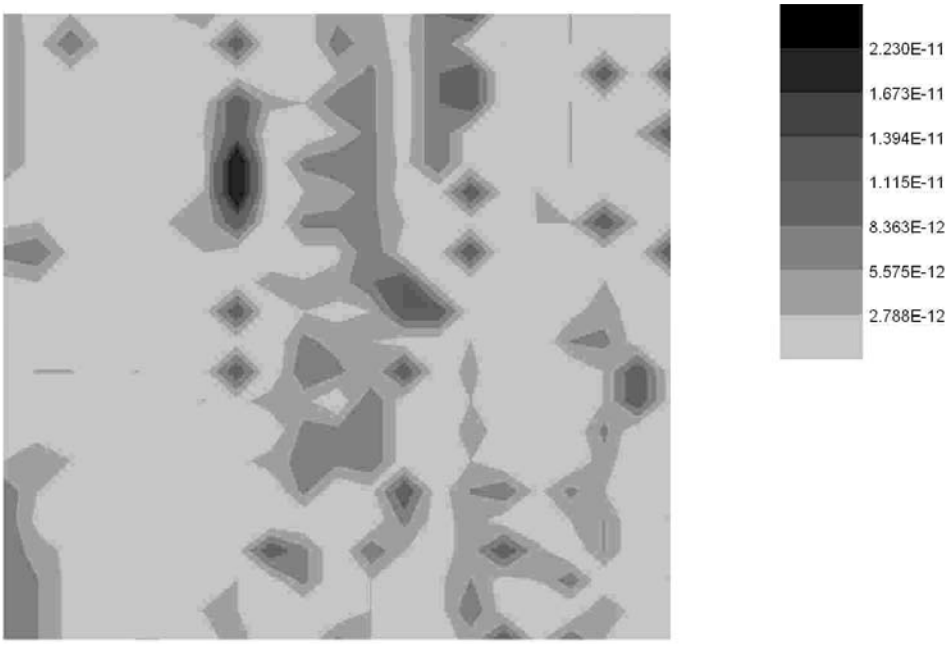


Figure 4.12 ALU usage map for the division software program

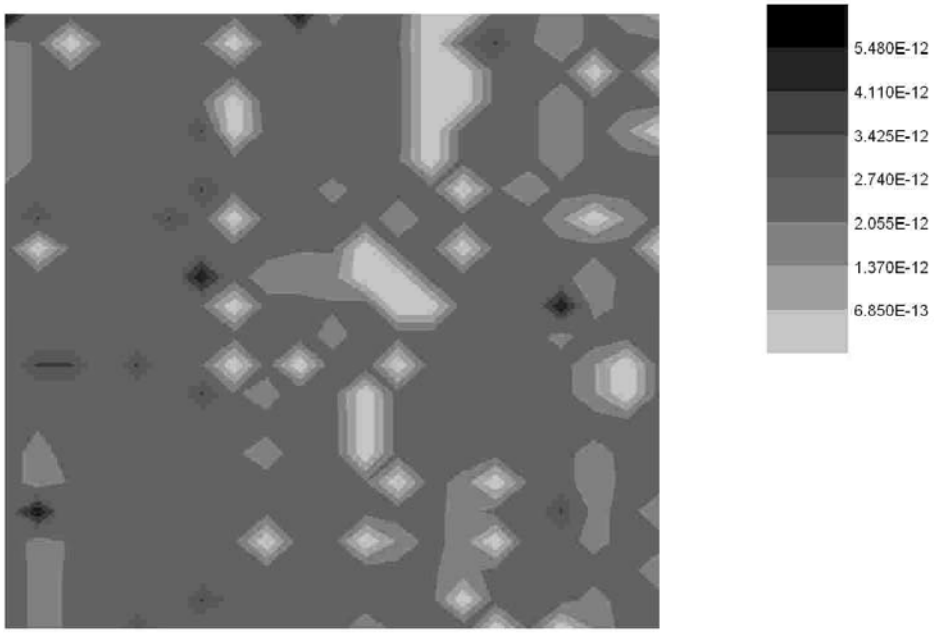
Figure 4.13 shows the failure probability information for the ALU gates of the Z80 CPU after one run of the division application software.



a) Delay failure manifestation.



b) Different-Function failure manifestation.



c) Stuck-at failure manifestation.

Figure 4.13 ALU map of probability distributions of different failure manifestations for the division program

The software-specific hardware failure profile of the Z80 ALU gates for the delay manifestation is shown in Figure 4.13a. The total delay failure probability of the ALU gates at the end of the software execution is 2.6E-09. Out of the 461 gates, 86 have probabilities within [8E-11, 1E-11], 92 gates within [10E-12, 1E-12], 247 gates within [10E-13, 1E-13], 24 gates within [10E-14, 1E-14], and 12 are not affected. Per gate type, the highest failure probability types are mux2_1 (1.33E-09) and xor_1 (5.55E-10), and the lowest are of type or2_1 (3.45E-12) and nor4_1 (2.10E-12). Among the gates of a same type, the highest probability variance is experienced by gates of type nor4_1 and inv_1, while the lowest variance is experienced by nand2_1 and xnor2_1 gates (with coefficient variations of 1.45, 1.02, 0.42 and 0.06, respectively). Per individual gates within a type, the gates with the maximum failure probability are of type mux2_1 (with one gate with probability 7.6E-11) and xnor2_1 (with one gate with probability 7.3E-11), and gates with the minimum failure probability are of type ABorC, inv_1 nor2_1, and or2_1 (with at least one gate with 0 probability). The highest probability averages are observed in gates of type xnor2_1 (6.97E-11) and xor2_1 (2.13E-11), while the lowest in types nand2_1 (2.74E-13) and nor4_1 (2.10E-13).

The failure probability profile of the Z80 ALU gates for the different function manifestation is displayed in Figure 4.13b. The total failure probability of the different function manifestation of the ALU gates at the end of the software execution is 1.12E-09. Of all the gates in the ALU, 2 gates have probabilities within [1E-11, 2E-11], 312 gates within [10E-12, 1E-12], 70 gates within [10E-13, 9E-13], 77 gates (the inv_1 type) are not affected. Per gate type, the highest failure probability types are

xor2_1 (2.36E-10) and nor2_1 (1.99E-10), and the lowest are or4_1 (2.99E-12) and inv_1 (0). Among the gates of the same type, the highest probability variance is experienced by gates of type nor2_1 and nand2_1 (with coefficients of variation of 8.07E-08, 2.93E-08), while the lowest variance is experienced by gates of type ab_or_c_or_d, mux2_1, nor3_1, nor4_1, or2_1, xnor2_1 and xor2_1 (0 variation coefficient). Per individual gates of the same type, the gates with the maximum failure probability are of type xnor2_1 (with one gate with probability 1.8E-11) and xor2_1 (with one gate with probability 9.1E-12), and the minimum are types inv_1 (with at least one gate with 0 probability). The highest probability averages are observed in gates of type xnor2_1 (1.78E-11) and xor2_1 (9.09E-12), while the lowest in types nor2_1 (1.50E-12) and mux2_1 (9.46E-13).

The failure probability profile for all the gates in the ALU for the stuck-at manifestation is shown in Figure 4.13c. The total stuck-at failure probability of all the ALU gates at the end of the software execution is 7.65E-10. Out of the 461 gates, 417 have probabilities within [4.5E-12, 1E-12], while 44 gates (of types not_ab_or_c_or_d and xor2_1) are not affected. Per gate type, the highest failure probability types are nor2_1 (2.28E-10) and inv_1 (1.46E-10), and the lowest are or4_1 (2.99E-12) and not_ab_or_c_or_d and xor2_1 (0). Within the gates of a same type, the highest probability variance is experienced by gates of type nor2_1, ab_or_c_or_d, nor4_1 and nand2_1 (with coefficients of variation within [8.53E-08 1.46E-08], while the lowest variance is experienced by gate type ABorC, and2_1, inv_1, mux2_1, nand3_1, nand4_1, nor3_1, or2_1 and xnor2_1 (0 variation). Per individual gates of the same type, the gates with the maximum failure probability are

of type or4_1 (with one gate with probability 4.40E-12) and or2_1 (with one gate with probability 3.80E-12), and gates with the minimum probability are types not_ab_or_c_or_d and xor2_1 (with at least one gate with 0 probability). The highest probability averages are observed in gates of type or4_1 (4.37E-12) and or2_1 (3.79E-12), while the less in types nand3_1 (1.31E-12) and nand2_1 (1.15E-12).

Figure 4.14 shows the combined failure probability distribution of the ALU gates of the Z80 CPU for the division application software (i.e., Delay+DiffFunc+Stuck-at in Figure 4.13).

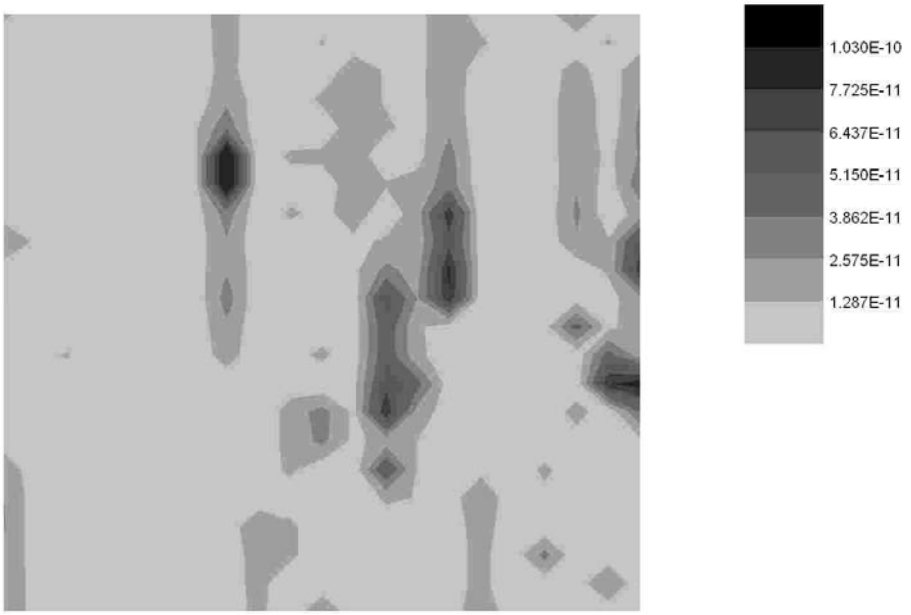


Figure 4.14 ALU map of combined failure probability distribution for the division program

The total failure probability of all the ALU gates at the end of the software execution is 4.48E-09. Out of the 461 gates, 115 gates have probabilities within [10E-11, 1E-11], and 346 gates within [10E-12, 1E-12]. Per gate type, the highest failure probability types are mux2_1 (1.53E-09) and xor2_1 (7.91E-10), and the lowest are

or2_1 (2.46E-11) and or4_1 (1.35E-11). For the gates of the same type, the highest probability variance is experienced by gates of type mux2_1 and xor2_1, while the lowest variance is experienced by gates of type xnor2_1 and nand2_1 (with coefficient variations of 0.82, 0.48, 0.05 and 0.03, respectively). Per individual gates of the same type, the gates with the maximum failure probability are of type xnor2_1 (with one gate with probability 9.2E-11) and mux2_1 (with one gate with probability 7.8E-11), and the minimum are of types nor2_1 (with one gate with probability 3E-12) and inv_1 (with at least one gate with probability 2E-12). The highest probability averages are observed in gates of type xnor2_1 (8.94E-11) and xor2_1 (3.04E-11), while the lowest occur in types nor2_1 (3.70E-12) and inv_1 (2.73E-12).

The hardware failure probability distributions discussed before is based on the execution of the division software program with a particular set of input values. The probability distribution profile is not constant for different software programs. Figure 4.15 shows the ALU gates usage map for a 16-bit bubble sorting software program. The program sorts an array of integer numbers in descending order. The assembly language program contains 342 lines of code. The corresponding failure probability distribution map of all the gates in the ALU is illustrated in Figure 4.16.

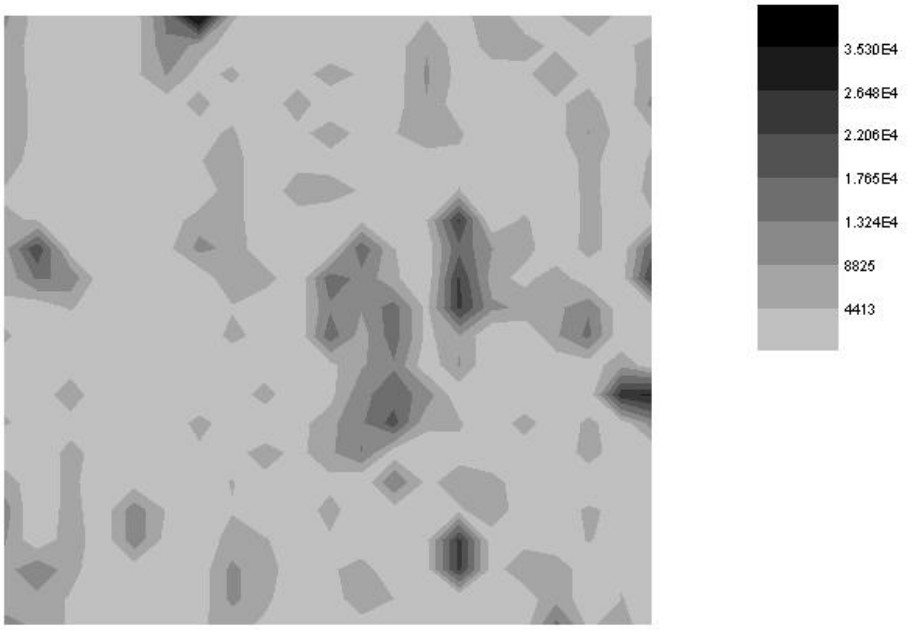


Figure 4.15 ALU usage map for the bubble sorting program

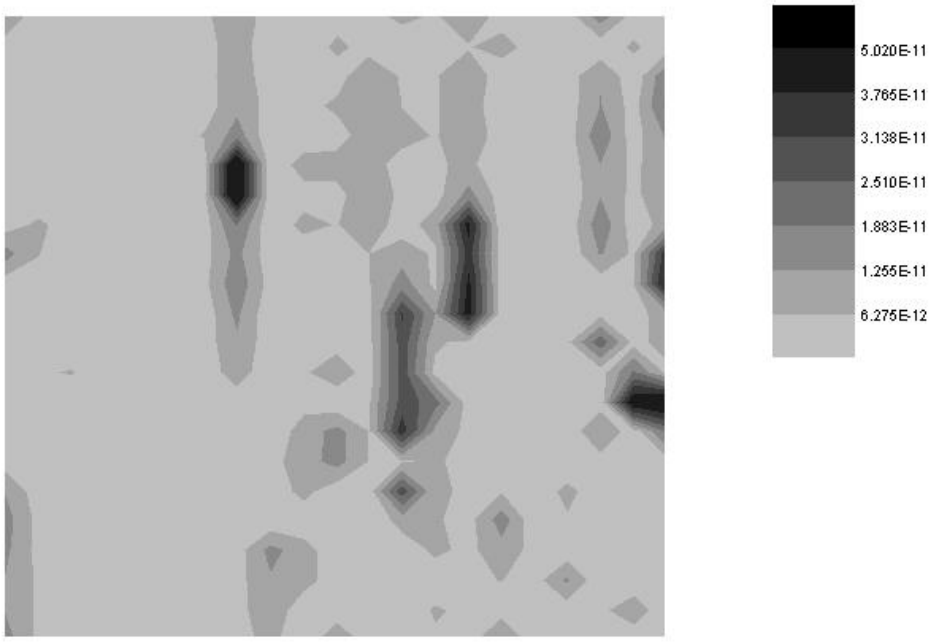


Figure 4.16 ALU map of combined failure probability distribution for the bubble sorting program

By comparing the ALU gate usage map of the bubble sorting program to that of the division program (Figure 4.12), we can see that the maximum number of demands is higher in the division program. The usage distributions for all the gates are different as shown in the difference of gray areas on the usage map. However, there are also similarities between the two maps. The most frequently demanded gates are located roughly in the same spots, which means that even though these are two different software programs, the most demanded hardware components are almost the same. Similar conclusions can be made for the comparison of the failure probability distribution maps of the two programs (Figure 4.14 and Figure 4.16).

This information is useful for both software and hardware reliability engineers, especially when working on developing embedded systems, where the software programs used are more or less fixed. For software reliability engineers, they should put more emphasis on studying the hardware components with high failure probabilities during the execution of the software program, rather than assuming that all the hardware components have the same probability of failure. For hardware reliability engineers, they should work with the hardware designer to decrease the failure probability of failure by making proper adjustments for the highly stressed hardware components. This will be the most efficient way to increase both the hardware reliability and the reliability of the software running on the hardware device.

The effects of different software input values and different compilers on the hardware failure probability distribution are also analyzed in [68]. The analyses indicate that while different inputs may yield slightly different failure probability values, the shape of the failure distribution is roughly the same. Different compilers

could also lead to different failure distributions. Compilers that tend to use more registers to store intermediate data will result in shorter execution time.

Chapter 5 Transient Failures and Models

The impact of hardware failures on software reliability is discussed in detail in the previous three chapters, where only permanent failures are considered. However, for some systems, such as those used for space applications, transient failures caused by external radiation become an important factor, which could impact the reliability of the system. In this chapter, transient failures from different sources of radiation are studied. Then the case study used in Chapter 4 is extended to consider the transient failure probability distributions of the same hardware devices in a satellite application environment.

5.1 Transient Failures

5.1.1 Transient Failure Introduction

Transient failures, also called soft errors or Single Event Upsets (SEUs) [69], appear in semiconductor devices during system operation due to electrical noise (e.g., noisy power supply) or external radiation such as α-particles, cosmic rays or nuclear reactions. SEUs mainly consist of the generation of electron-hole pairs due to the collision of energetic particles with the silicon atoms, which in turn can lead to temporal voltage and current peaks in the circuit. So, contrary to permanent failures, SEUs do not introduce physical defects in the circuit.

Electrical noise may come from well-known sources such as a noisy power supply or radiation from lightning. Extensive design efforts have been made during the last decades to make electronics immune to such noise.

The α-particles are emitted by radioactive impurities (e.g., uranium) present in packaging materials and the interconnect wires of integrated circuits [70]. Nowadays, the SEU rate induced by α-particles can be drastically reduced by the use of highly purified materials (e.g., α-particle emission from chip metallization can be reduced by a factor of approximately 1000 by using a highly purified aluminum with an impurity concentration of the order of 2ppb [71]).

Cosmic rays are the main the source of radiation in deep space leading to SEUs in microelectronic devices, in particular due to proton and heavy ion particles [72]. Within the earth's atmosphere the major causes of SEUs are the neutron particles from spallation reactions occurring when the galactic cosmic rays collide with the oxygen and nitrogen atoms in the air. At sea level, about 97% percent of the remaining cosmic ray particles are neutrons. Neutrons are expected to cause upsets in microelectronic devices within 18km in the atmosphere. Energetic protons are also abundant in the near-Earth Van Allen belts.

Nuclear reactions (e.g., from spacecrafts or nuclear power-plant reactors) also lead to an important emission of neutrons [72]. In particular, in future nuclear-powered space missions from NASA, neutrons induced from space nuclear reactors will have an important impact on the reliability of microelectronic devices used in the spacecraft.

5.1.2 Impact on Higher Hardware Levels

SEUs mainly manifest themselves in the form of pulses in combinational logic (a temporal peak of current or voltage in a signal) and bit-flips in storage elements (the stored bit changes from 0 to 1 or vice-versa) [73]. However, unlike the case of

permanent failures, SEUs in combinational logic are less likely to propagate to the storage elements due to a set of well-known masking mechanisms (i.e., logical masking, electrical masking and latching-window masking [73]).

Therefore, this chapter focuses on investigating the impact of SEUs on hardware storage elements, such as flip-flops. The failure rates of bit-flips from different radiation sources are introduced in the following section.

5.2 Failure Rate Calculation

5.2.1 Heavy Ions Induced SEUs

Due to their high Linear Energy Transfer (LET) values, heavy ions can cause direct ionization when passing through microelectronic devices and leading to SEUs. The LET corresponds to the amount of energy lost by the radiation particles per unit of distance traveled, which is deposited into the device. The fundamental assumption of the upset mechanism is that there is a Sensitive Volume (SV) within a semiconductor device that can be upset by the passage of the radiation particles. The SV is thus independent from the radiation particles considered. The SV is generally modeled as a right Rectangular Parallelepiped (RPP) shape with lateral dimensions x and y and thickness z. Associated with the SV is a Cross Section (CS) that can be interpreted as the projection of the SV in the direction of the movement of the radiation particles.

The data needed to calculate the heavy ions induced failure rate is shown in Figure 5.1.

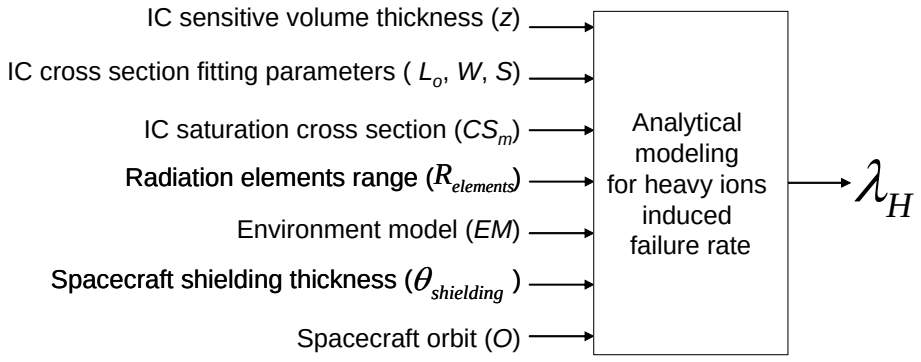


Figure 5.1 Heavy ions SEU rate calculation

According to Figure 5.1, three different types of data are necessary to calculate the heavy ions induced failure rate:

- Data dependent upon the design and technology of the IC storage elements (z, L_o, W, S, CS_m),
- Data dependent upon the operational space environment of the spacecraft ($R_{elements}$, EM),
- Data dependent upon the particular characteristics of the mission and the spacecraft design ($\theta_{shielding}$, O).

The physical meaning of each parameter is described as follows:

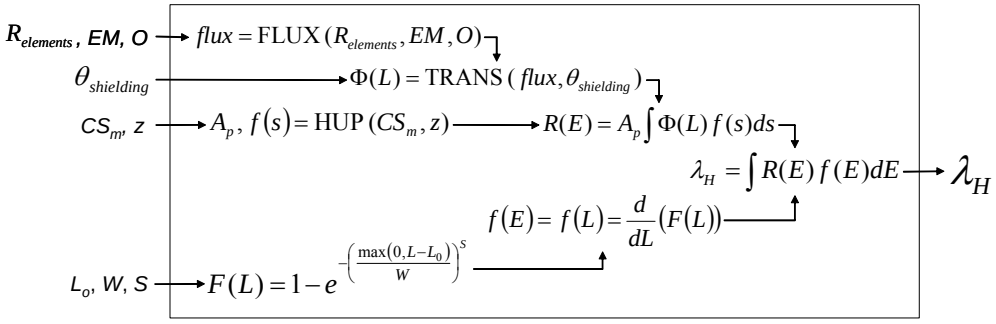
- IC sensitive volume thickness (z) is the thickness of the Sensitive Volume (SV).
- IC cross section fitting parameters (L_o, W, S) are a set of values used to approximate the Cross Section (CS) curve as a function of the Linear Energy Transfer (LET). They correspond to the onset threshold LET (L_o), width (W) and shape (S) of a Weibull distribution.

- IC saturation cross section (CS_m) is the value that CS approaches as LET gets very large. It is equivalent to area xy of the SV.
- Radiation elements range ($R_{elements}$) refers to the range of radiation elements present in the environment (e.g., heavy ions He^{+2} to Ni^{+28}).
- Environment model (EM) corresponds to the operational environment of the spacecraft. It considers long-term average and worst case particle fluxes.
- Spacecraft shielding thickness ($\theta_{shielding}$) corresponds to the thickness of the spacecraft shielding.
- Spacecraft orbit (O) corresponds to the spacecraft orbit, including Near-Earth Interplanetary orbits (e.g., earth to mars) and orbits inside the magnetosphere.

Experimental data for the (saturation) cross section (CS, CS_m) and the fitting parameters (L_o, W, S) are available from on-ground based radiation tests. These tests are performed by subjecting the device to radiation particles of a range of LETs. The sensitive volume thickness (z) is given by the semiconductor technology specifications. The radiation elements range ($R_{elements}$), environmental model (EM), spacecraft shielding thickness ($\theta_{shielding}$) and orbit (O) can be determined from the spacecraft and mission specifications.

The analytical modeling for the heavy ions induced failure rate is shown in Figure 5.2, and is based on the IRPP model [74-76]. The model has been implemented by CREME96 program [77, 78]. CREME96 (Cosmic Ray Effects on Micro-Electronics) is a suite of computer program developed at the Naval Research

Laboratory. It has become a widely used design tool in the aerospace industry for SEU rate calculation.



E, ∫dE	Variable E is the threshold energy for generating a device critical charge, whose integration range is provided by the CREME96 FLUX module.
flux	Heavy ions flux spectra outside of the spacecraft, calculated using the CREME96 FLUX module.
Φ(L)	Integral flux over ion LET for the environment of concern, calculated using the CREME96 TRANS module.
A_p	Average projected area of the right rectangular parallelepiped shaped sensitive volume.
f(s)	Distribution of path lengths through the sensitive volume.
R(E)	The upset rate for a particular threshold energy.
F(L)	The integral Weibull distribution describing the shape of the CS versus LET curve.
L	Threshold LET
f(E)	Probability density function converted from the CS versus LET curve.

Figure 5.2 Analytical modeling for heavy ions induced failure rate

5.2.2 Protons Induced SEUs

The basic physics of the upset interaction for protons is the same as for heavy ions. Both types of upsets are caused by the ionization of a device after it collects charge produced by the ionization of a passing radiation particle. The difference is that heavy ions can produce SEUs directly due to the high LET values, while proton upsets are caused by the ionization of secondary particles produced by a nuclear reaction in the vicinity of the sensitive volume.

The data needed to calculate the protons induced failure rate is shown in Figure 5.3.

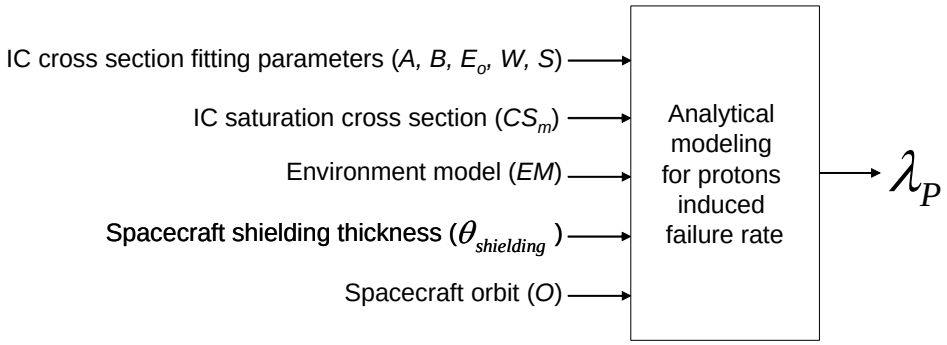
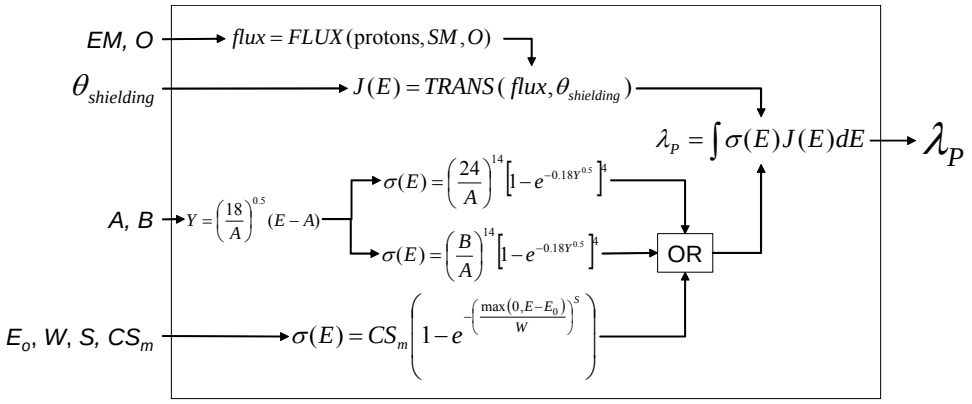


Figure 5.3 Protons SEU rate calculation

The environment model (EM), the spacecraft shielding thickness ($\theta_{shielding}$), the spacecraft orbit (O), and the saturation cross section (CS_m) are defined in Section 5.2.1. Parameters A, B, E_o, W, S are used to adjust the cross section curve. The analytical modeling for the proton induced failure rate is shown in Figure 5.4, and is based on the Bendel and Petersen models [79] and the Weibull distribution.



E, ∫dE Variable E is the proton energy, whose integration range is provided by the CREME96 FLUX module.

σ(E) Proton cross section as a function of proton energy.

J(E) Differential proton flux at the sensitive volume.

Figure 5.4 Analytical modeling for protons induced failure rate

5.2.3 Neutrons Induced SEUs

Similar to the protons induced SEUs, neutron upsets are caused by the ionization of secondary particles produced by a nuclear reaction around the sensitive volume. We distinguish between the upset rate caused by neutrons present in the earth's atmosphere (Figure 5.5) and neutrons emitted by spacecrafts nuclear reactors (Figure 5.6).

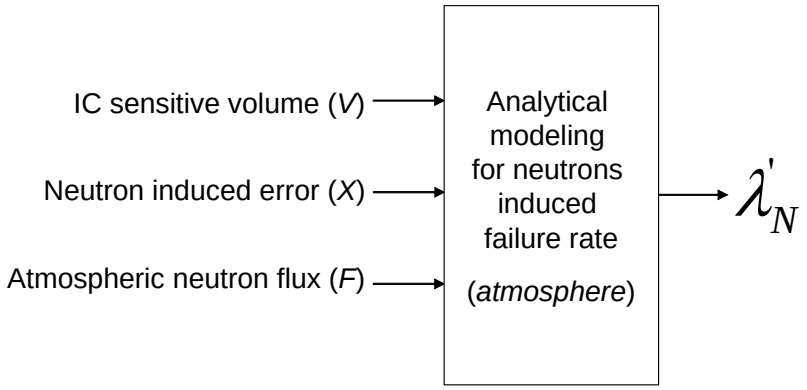


Figure 5.5 Atmospheric neutron SEU rate calculation

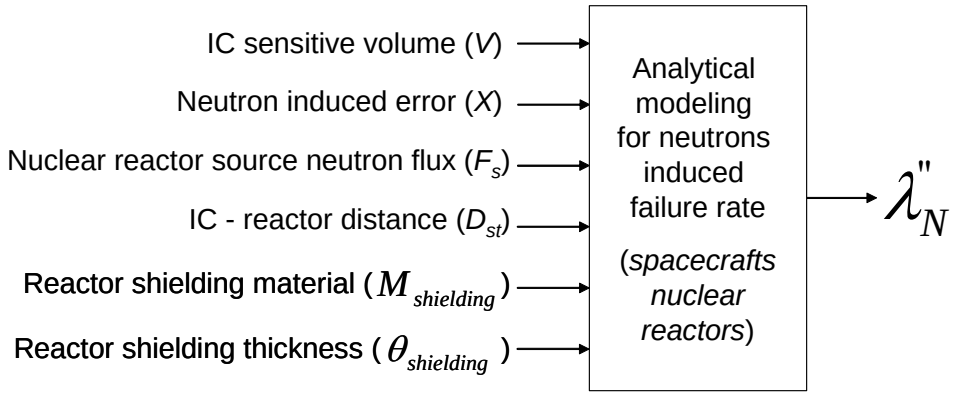


Figure 5.6 Spacecraft nuclear reactor neutron SEU rate calculation

The concept of IC sensitive volume (parameter V) has already been described in Section 5.2.1. Parameter X is the value of the neutron-induced error (NIE) as a function of the device critical charge [80]. F is the integral flux of neutrons present in the atmosphere with energy above 1MeV. Regarding spacecraft nuclear reactors, F_s represents the neutron flux emitted by the reactor, D_{st} is the distance between the reactor and the target semiconductor devices, $M_{shielding}$ is the shielding material of the reactor, and $\theta_{shielding}$ is the reactor shielding thickness.

Letaw and Normand calculated the neutron-induced error (NIE) as a function of the device critical charge for different environments [80]. They proposed a simple analytical model for the neutron induced SEU rate as the product of parameters F, V and X, as shown in Figure 5.7 and Figure 5.8.

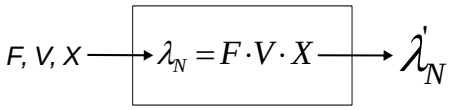


Figure 5.7 Analytical modeling for atmospheric neutrons induced failure rate

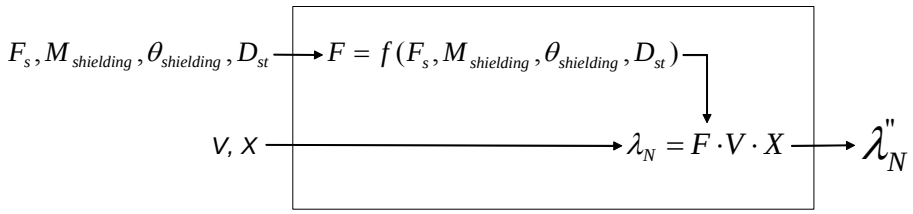


Figure 5.8 Analytical modeling for spacecraft nuclear reactor neutrons induced failure rate

Value F for the neutron flux depends on the environment. For atmospheric neutrons (Figure 5.7), a typical neutron flux above 1 MeV is $1cm^{-2}s^{-1}$. For nuclear reactor induced neutrons, the neutron flux at the surface of the microelectronic devices *F* is inversely proportional to the square of the distance to the neutron source D_{st}. The flux value can also be reduced by the shielding material. The effectiveness of the shielding should be obtained through radiation testing. For example, the neutron flux can be reduced by 1 magnitude with LiH shielding of 7.8cm thickness [81]. Therefore, the final flux value *F* can be calculated as a function of F_s, $M_{shielding}$, $\theta_{shielding}$ and D_{st}, as shown in Figure 5.8.

5.3 Extension of Permanent Failure Probability Results

With the information on transient failures above, the study of the hardware failure probability distribution can be extended to include the case for transient failures. We now consider the computer system described in the last chapter is used in a satellite application. The orbit considered here is a Highly Elliptical Orbit (HEO) with an inclination of 27 degrees around the earth (with an apogee of 35449 km and perigee of 3997 km). The period of the orbit is 11 hr 39 min. The environment model corresponds to a solar quiet condition with long-term average SEU rates. Assume the shielding thickness for the satellite is 100 mils (which equals to 0.25cm). The range of radiation elements includes all possible heavy ions (atomic number from 2 to 92) and protons. We assume that a nuclear reactor will not be used for the satellite orbit, which means there will be no neutron in this orbit environment.

To calculate the SEU rates for different radiation sources, the parameters in Figure 5.2 and Figure 5.4 have to be obtained. The environment and mission orbit related parameters can be chosen based on the above description. The IC storage elements related parameters should be extracted from radiation testing. Since we do not have such testing data available for the flip-flops in the vtvtlib25 cell library, the radiation testing data of similar technology is used, assuming they have similar SEU rates. Ground based radiation tests have been performed on the registers of the PPC750 microprocessor by IBM and Motorola [82], The PPC750 microprocessor is also based on the CMOS 0.25μm technology. The Weibull fitting parameters are extracted from the test data of the PPC750 registers. All the parameters needed are summarized in Table 5.1.

Table 5.1 Parameters for the calculation of SEU rates for the HEO orbit profile

Parameters	Heavy ions	Protons
z	2μm	
L_o	3.6 MeV-cm^2/mg	
W	30.82 MeV-cm^2/mg	88.98 MeV
S	0.92	0.95
CS_m	10 μm^2	0.1×10^{-12} cm^2
$R_{elements}$	atomic number 2 to 92	
EM	long-term average fluxes	
$\theta_{shielding}$	100 mils	
O	Highly Elliptical Orbit	
E_o		8.0 MeV

The average SEU rates induced by heavy ions and protons for each orbit segment are calculated and illustrated in Figure 5.9. The Y axis denotes the SEU rate in the unit of SEUs/Bit/Hour. The X axis denotes the time along an orbit period. The total SEU rates for the orbit as a function of time are shown in Figure 5.10.

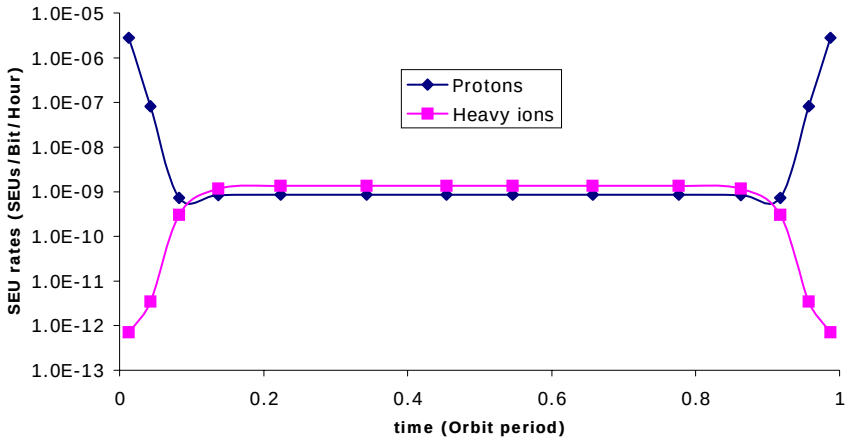


Figure 5.9 Heavy ions and protons induced SEU rates along the HEO orbit as a function of time

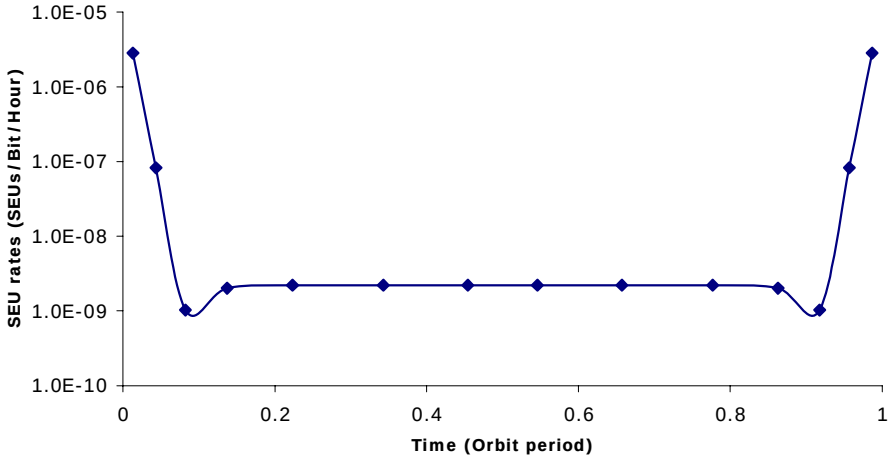


Figure 5.10 Total SEU rates along the HEO orbit as a function of time

High SEU rates appear when the satellite is close to Earth, where trapped protons are mainly responsible for the transient failures. When the satellite is far from Earth, the SEU rates induced by heavy ions and protons are about the same order.

Figure 5.11 shows a comparison of transient and permanent failure probabilities for several register bits along the orbiting period, assuming the CPU is executing the division program described in Section 4.4 repetitively. The transient failure probability increases rapidly during the time when the satellite is close to Earth due to the high SEU rates. On the other hand, the permanent failure probabilities increase in a stable style regardless of the satellite's possition. In this case, the SEU is the dominant failure mechanism in the satellite orbit environment.

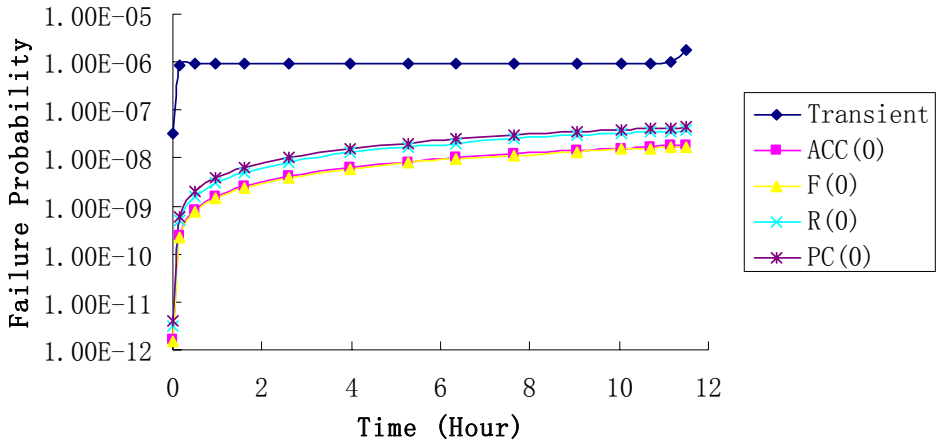


Figure 5.11 Transient and permanent failure probabilities along the orbit as a function of time

However, the transient SEU rate can be reduced through radiation hardening techniques. Ground-based tests show that the SEU rate of a radiation hardened microprocessor, such as the RAD750 developed by BAE system based on the PPC750 microprocessor, can be reduced by up to six orders of magnitude [83, 84]. If the registers in the Z80 microprocessor are hardened using such techniques, the SEU dominance could disappear. Assuming the SEU rate can be reduced by two orders of

magnitude, the probabilities for transient and permanent failures along the satellite orbit would be of the same order, as illustrated in Figure 5.12. If this is the case, both transient and permanent failures should be considered to study their impact on software reliability.

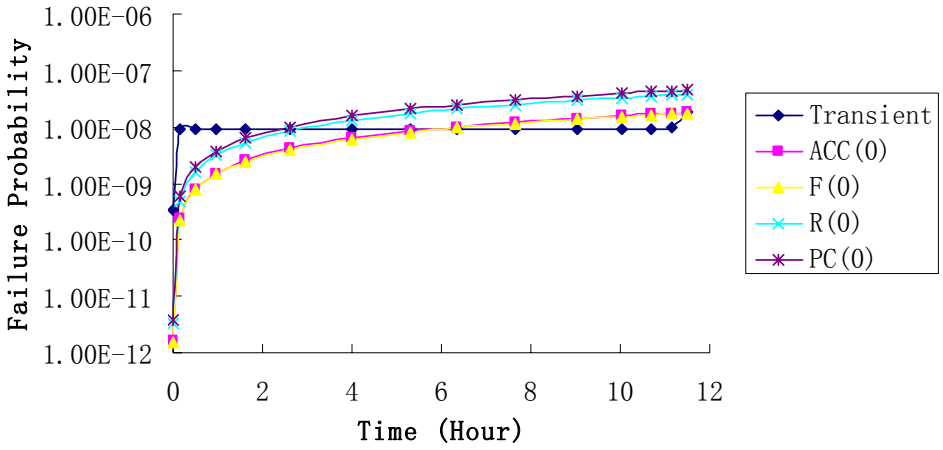


Figure 5.12 Transient and permanent failure probabilities along the orbit as a function of time, with radiation hardening techniques applied

Chapter 6 Summary and Future Research

6.1 Summary

In this dissertation, we have developed a methodology for the reliability analysis of the manifestations of permanent hardware failures in the hardware devices of computer systems operating under a particular execution profile of application software. An analysis of the different types of manifestations of permanent failure in semiconductor devices was performed at different hardware levels (physical, logic and register transfer levels). We have focused on intrinsic failures (HCI, EM, TDDB), which propagate to higher hardware levels in the form of signal delays, changes of circuit functionality, and signals stuck at a logic value (0 or 1). We then proposed a methodology for the analysis of the manifestations of permanent hardware failures on software reliability. The methodology is divided into three parts: (i) analysis of the manifestations of permanent failures on circuit elements (logic gates, flip-flops, etc.), (ii) development of reliability models as functions of the software execution, and (iii) calculation of failure probability distributions of the hardware devices of a computer system under a particular software execution.

In the first part of this methodology (analysis of failure manifestations), SPICE simulation is performed to investigate the behavior of the circuit elements (logic gates and flip-flops) under study with a set of generic input stimuli, which covers all possible combinations of logic levels and transitions of the input signals. This allows for calculating the failure rates of different circuit elements. A set of Failure equivalent circuit models for different failure mechanisms, including HCI, EM, and

TDDB is used to study the circuit failure manifestations in the presence of hardware failures. The main outcome of this phase consists of the set of manifestations of the permanent failures observed in the circuits' output signals (e.g., signal delays, functionality changes or stuck-at failures).

In the second part of the methodology (development of reliability models), a set of reliability models are built that allow for calculating the occurrence rate of each failure manifestation of a circuit as a function of the software execution profile of a computer system. The models are based not only on existing expressions for the constant stress failure rate of permanent failures, but also on specifically developed models that account for the operational conditions of circuits (e.g., current and voltages) and for the usage of the computer hardware devices as a consequence of the software execution. Different structures and notations are proposed for the reliability models in order to process huge numbers of failure manifestations into reduced and practical sets of expressions.

In the third part of the methodology (calculation of failure probability distributions) the reliability models developed in the previous phase are applied to a particular computer platform. The usage of the hardware devices is obtained through VHDL simulations of the computer system under the execution of the software program of interest. This allows for solving the reliability models and calculating the failure probability distributions (per failure manifestation) of the various hardware devices of the computer platform (e.g., ALU gates, CPU registers, memories, etc.).

We have then extended the methodology to the consideration of transient failures, also known as Single Event Upsets (SEUs). We studied the causes and

manifestations of transient failures in semiconductor devices, and developed reliability models that integrate into the same framework well-known analytical models for the failure rate calculation of Single Event Upsets (SEUs). These take into account SEUs induced by cosmic ray particles (heavy ions and protons), neutrons present in the atmosphere, as well as neutrons emitted by nuclear reactors such as the ones that will be used in the spacecrafts of the future nuclear powered space missions from NASA. The models use design and technology parameters of the IC hardware devices, the operational environment characteristics (radiation particle fluxes) as well as the specifications of the system and mission (e.g., spacecraft shielding and orbit). The case study was then extended to the consideration of transient failures, by calculating the failure probability distributions due to SEUs of the hardware devices of the Z80 based computer system.

6.2 Conclusions

The contribution of this dissertation is to propose a simulation-based method to determine the software-specific hardware usage profile and failure probability profile that can be used to determine the likelihood, location, and the time of hardware failures in computer systems in operation. The main features and contributions of the methodology are summarized hereafter:

- It takes into account the influence of the software execution, the operational environment and the semiconductor design and technology in the creation and activation phenomena of hardware failures.
- It includes the whole spectrum of hardware failures that can arise during the system operation, i.e. not only Single Event Upsets (SEUs), but also

permanent semiconductor defects due to Hot Carrier Injection, Electromigration, and Time Dependent Dielectric Breakdown.

- It considers all the possible locations for the hardware failures, i.e. not only sequential logic circuits (registers, memory cells, caches, etc.) but also combinational logic circuits (logic gates).
- It analyzes the propagation of failures under particular operational conditions (including the software execution) and precisely determines the form under which each hardware failure manifests (stuck-at-1, stuck-at-0, bit-flip, circuit delay, change of functionality, etc.).
- It takes into account the usage of the hardware circuit elements due to the software execution during the operational life of the system and provides the failure probability distributions of the circuit elements. This information can facilitate both software and hardware reliability engineers to improve the system reliability more efficiently by focusing on the most failure-prone circuit elements.
- It can be used to extend the use of the fault injection technique to software reliability prediction under hardware failures and allows for precisely defining representative fault models that can be used in fault injection techniques and tools. It also sets the basis to develop testbeds based on software implemented fault injection (SWIFI) to calculate the final failure probability of the software application. As far as we know, this is the first time that such an extension has been proposed.

6.3 Future Work

We have focused on permanent and transient failures that directly impact the storage elements and logic gates in a computer system. One of the future directions of this work will be to extend the approach to account for the impact of permanent and transient failures that propagate from combinational logic circuit elements of a microprocessor to its storage elements. We also plan to extend the proposed reliability models to account for the effect of the hardware detection and recovery mechanisms (e.g., error detection and correction codes) used in most modern computer systems. Finally, the results of this research will be integrated into PRA frameworks and will be used for the calculation by fault injection of the software reliability of software-intensive safety critical systems.

Appendix A Failure Manifestations for Logic Gates

Table A.1 Z80 ALU logic gates – Results of SPICE simulations for HCI stress

	Mi	$\lvert I(i,hci)\rvert / I$	F
INV_1	M0	1/2	$Delay - fall$
NOR2_1	M0	1/8	$Delay - fall - \uparrow 0$
	M1	1/8	$Delay - fall - 0 \uparrow$
OR2_1	M0	1/8	$Delay - rise - \uparrow 0$
	M1	1/8	$Delay - rise - \uparrow 1$
	M2	2/8	$Delay - fall$
NAND2_1	M0	1/8	$Delay - fall$
	M1	1/8	$Delay - fall$
XOR2_1	M0	2/8	$Delay - rise - \downarrow 1 - 0 \uparrow$
	M3	2/8	$Delay - rise - \uparrow 0$
	M5	1/8	$Delay - fall - 1 \uparrow - \uparrow 1$
	M7	1/8	$Delay - fall - 1 \uparrow - \uparrow 1$
	M9	1/8	$Delay - fall - 0 \downarrow - \downarrow 0$
	M11	1/8	$Delay - fall - 0 \downarrow - \downarrow 0$
NOR3_1	M0	1/16	$Delay - fall - \uparrow 00$
	M3	1/16	$Delay - fall - 0 \uparrow 0$
	M5	1/16	$Delay - fall - 00 \uparrow$
MUX2_1	M0	4/8	$Delay - fall - 10 \downarrow$
	M3	1/8	$Delay - rise - \uparrow 01 - 1X \uparrow$
	M5	2/8	$Delay - rise - \uparrow 01 - 1X \uparrow$
	M7	2/8	$Delay - rise - 0 \uparrow X - \downarrow 10$
	M9	1/8	$Delay - rise - 0 \uparrow X - \downarrow 10$
	M11	6/8	$Delay - fall$
ABorC	M0	1/16	$Delay - rise - 1 \uparrow 0 - 11 \uparrow$
	M3	1/16	$Delay - rise - 1 \uparrow 0 - 11 \uparrow$
	M5	3/16	$Delay - rise - XX \uparrow$
	M7	5/16	$Delay - fall$
NAND3_1	M0	1/16	$Delay - fall$
	M1	1/16	$Delay - fall$
	M2	1/16	$Delay - fall$
XNOR2_1	M0	2/8	$Delay - fall - 0 \uparrow$
	M3	2/8	$Delay - fall - \uparrow 0$
	M5	1/8	$Delay - rise - 1 \uparrow - \uparrow 1$
	M7	1/8	$Delay - rise - 1 \uparrow - \uparrow 1$
	M9	1/8	$Delay - rise - 0 \downarrow - \downarrow 0$
	M11	1/8	$Delay - rise - 0 \downarrow - \downarrow 0$
	M13	4/8	$Delay - fall$

	Mi	$\lvert I(i,hci)\rvert / I$	F
NAND4_1	M0	3/32	$Delay - fall$
	M1	1/32	$Delay - fall$
	M2	2/32	$Delay - fall$
	M3	1/32	$Delay - fall$
NOR4_1	M0	1/32	$Delay - fall - \uparrow 000$
	M3	1/32	$Delay - fall - 0 \uparrow 00$
	M5	1/32	$Delay - fall - 00 \uparrow 0$
	M7	1/32	$Delay - fall - 000 \uparrow$
OR4_1	M0	1/32	$Delay - rise - \uparrow 000$
	M3	1/32	$Delay - rise - 0 \uparrow 00$
	M5	1/32	$Delay - rise - 00 \uparrow 0$
	M7	1/32	$Delay - rise - 000 \uparrow$
	M9	4/32	$Delay - fall$
NOT (ABorCorD)	M0	1/32	$Delay - fall - 1 \uparrow 00 - \uparrow 100$
	M3	1/32	$Delay - fall - 1 \uparrow 00 - \uparrow 100$
	M5	2/32	$Delay - fall - X0 \uparrow 0 - 01 \uparrow 0$
	M7	3/32	$Delay - fall - X00 \uparrow -010 \uparrow$
ABorCorD	M0	1/32	$Delay - rise - 1 \uparrow 00 - \uparrow 100$
	M3	1/32	$Delay - rise - 1 \uparrow 00 - \uparrow 100$
	M5	2/32	$Delay - rise - X0 \uparrow 0 - 01 \uparrow 0$
	M7	3/32	$Delay - rise - X00 \uparrow -010 \uparrow$
	M9	8/32	$Delay - fall$
AND2_1	M0	1/8	$Delay - rise$
	M1	1/8	$Delay - rise$
	M2	2/8	$Delay - fall$

Table A.2 Z80 ALU logic gates – Results of SPICE simulations for EM stress

	Ni	$\lvert I(i,em)\rvert / I$	F
INV_1	N1	2/2	$Delay - rise$
	N2	2/2	$Delay - fall$
	N3	2/2	$Delay - fall$
NOR2_1	N1	4/8	$Delay - rise$
	N2	4/8	$Delay - rise$
	N3	2/8	$Delay - fall - \uparrow 0$
	N4	2/8	$Delay - fall - 0 \uparrow$
OR2_1	N1	4/8	$Delay - fall$
	N2	4/8	$Delay - fall$
	N3	0	-
	N4	2/8	$Delay - rise - \uparrow 0$
	N5	2/8	$Delay - fall$

	Ni	$\lvert I(i,em)\rvert / I$	F
NAND2_1	N1	2/8	$Delay - rise - \downarrow 1$
	N2	2/8	$Delay - rise - 1 \downarrow$
	N3	4/8	$Delay - fall$
	N4	4/8	$Delay - fall$
XOR2_1	N1	4/8	$Delay - fall - 0 \downarrow$
	N2	6/8	$Delay - rise - \uparrow 0 - 1 \downarrow$
	N3	2/8	$Delay - rise - \downarrow 1 - 0 \uparrow$
	N4	4/8	$Delay - fall - 0 \downarrow$
	N5	4/8	$Delay - fall - \downarrow 0$
	N6	4/8	$Delay - fall$
	N7	4/8	$Delay - rise - 0 \uparrow$
	N8	6/8	$Delay - fall - 1 \uparrow - \uparrow 1$
	N9	0	-
NOR3_1	N1	6/16	$Delay - rise$
	N2	6/16	$Delay - fall - \uparrow 00 - 0 \uparrow 0$
	N3	2/16	$Delay - fall - \uparrow 00$
	N4	4/16	$Delay - fall$
MUX2_1	N1	12/8	$Delay - fall - \uparrow 10 - 1X \downarrow$
	N2	6/8	$Delay - fall - \downarrow 01 - 0 \downarrow X$
	N3	0	-
	N4	8/8	$Delay - rise - 0 \uparrow 1$
	N5	10/8	$Delay - fall$
	N6	0	-
	N7	12/8	$Delay - rise - \uparrow 01 - 10 \uparrow$
	N8	4/8	$Delay - rise - 0 \uparrow X - \downarrow 10$
	N9	0	-
ABorC	N1	6/16	$Delay - fall - 01 \downarrow - \downarrow 10$
	N2	6/16	$Delay - fall - 1 \downarrow 0 - 10 \downarrow$
	N3	10/16	$Delay - fall$
	N4	0	-
	N5	10/16	$Delay - rise$
	N6	0	-
	N7	10/16	$Delay - rise$
	N8	0	-
NAND3_1	N1	2/16	$Delay - rise - \downarrow 11$
	N2	4/16	$Delay - rise - 11 \downarrow -1 \downarrow 1$
	N3	6/16	$Delay - fall$
	N4	6/16	$Delay - fall$
XNOR2_1	N1	4/8	$Delay - rise - 0 \downarrow$
	N2	6/8	$Delay - fall - \uparrow 0 - 1 \downarrow$
	N3	2/8	$Delay - fall - \downarrow 1 - 0 \uparrow$
	N4	0	-
	N5	4/8	$Delay - rise - \downarrow 0$
	N6	4/8	$Delay - rise - \downarrow 0$
	N7	4/8	$Delay - rise$

	Ni	$\lvert I(i,em)\rvert / I$	F
	N8	0	-
	N9	4/8	$Delay - fall - 0 \uparrow$
	N10	6/8	$Delay - rise - 1 \uparrow - \uparrow 1$
	N11	0	-
	N12	0	-
NAND4_1	N1	2/32	$Delay - rise - \downarrow 111$
	N2	4/32	$Delay - rise - 11 \downarrow 1 - 1 \downarrow 11$
	N3	2/32	$Delay - rise - 111 \downarrow$
	N4	8/32	$Delay - fall$
	N5	8/32	$Delay - fall$
NOR4_1	N1	8/32	$Delay - rise$
	N2	8/32	$Delay - fall - \uparrow 000 - 00 \uparrow 0 - 0 \uparrow 00$
	N3	2/32	$Delay - fall - \uparrow 000$
	N4	4/32	$Delay - fall - 00 \uparrow 0 - 0 \uparrow 00$
	N5	2/32	$Delay - fall - 000 \uparrow$
OR4_1	N1	8/32	$Delay - fall$
	N2	8/32	$Delay - fall$
	N3	0	-
	N4	2/32	$Delay - rise - \uparrow 000$
	N5	4/32	$Delay - rise - 0 \uparrow 00$
	N6	2/32	$Delay - rise - 000 \uparrow$
NOT (ABorCorD)	N1	10/32	$Delay - rise - 010 \downarrow -11 \downarrow 0 - 01 \downarrow 0$
	N2	10/32	$Delay - rise - 1 \downarrow 00 - 100 \downarrow -10 \downarrow 0$
	N3	16/32	$Delay - rise$
	N4	16/32	$Delay - fall$
	N5	4/32	$Delay - fall - 1 \uparrow 00 - \uparrow 100$
	N6	6/32	$Delay - fall - X0 \uparrow 0 - 01 \uparrow 0$
	N7	5/32	$Delay - fall - X00 \uparrow -010 \uparrow$
ABorCorD	N1	10/32	$Delay - fall - 010 \downarrow - \downarrow 100 - 01 \downarrow 0$
	N2	10/32	$Delay - fall - 1 \downarrow 00 - 100 \downarrow -10 \downarrow 0$
	N3	16/32	$Delay - fall$
	N4	0	-
	N5	16/32	$Delay - rise$
	N6	0	-
	N7	4/32	$Delay - rise - 1 \uparrow 00 - \uparrow 100$
	N8	6/32	$Delay - rise - X0 \uparrow 0 - 01 \uparrow 0$
	N9	5/32	$Delay - rise - X00 \uparrow -010 \uparrow$
	N10	0	-
AND2_1	N1	2/8	$Delay - fall - \downarrow 1$
	N2	2/8	$Delay - rise$
	N3	4/8	$Delay - rise$
	N4	0	-
	N5	4/8	$Delay - fall$

Table A.3 Z80 ALU logic gates – Results of SPICE simulations for TDDB stress

	Mi	$\lvert C(i,tddb)\rvert$	F
INV_1	M0	0.5	$Stuck-at-1$
	M1	0.5	$Stuck-at-0$
NOR2_1	M0	0.5	$DiffFunc \rightarrow \bar{b}$
	M1	0.5	$DiffFunc \rightarrow \bar{a}$
	M2	0.5	$Stuck-at-0$
	M3	0.25	$Stuck-at-0$
OR2_1	M0	0.5	$DiffFunc \rightarrow b$
	M1	0.5	$DiffFunc \rightarrow a$
	M2	0.25	$Stuck-at-1$
	M3	0.25	$Stuck-at-1$
	M4	0.5	$Stuck-at-1$
	M5	0.75	$Stuck-at-0$
NAND2_1	M0	0.5	$Stuck-at-1$
	M1	0.25	$Stuck-at-1$
	M2	0.5	$DiffFunc \rightarrow \bar{b}$
	M3	0.5	$DiffFunc \rightarrow \bar{a}$
XOR2_1	M0	0.5	$DiffFunc \rightarrow a\bar{b}$
	M1	0.5	$DiffFunc \rightarrow \overline{ab}$
	M2	0.5	$DiffFunc \rightarrow \overline{a\bar{b}}$
	M3	0.5	$DiffFunc \rightarrow \bar{a}b$
	M4	0.5	$DiffFunc \rightarrow \bar{a}b$
	M5	0.5	$DiffFunc \rightarrow a+b$
	M6	0.25	$DiffFunc \rightarrow \bar{a}b$
	M7	0.25	$DiffFunc \rightarrow a+b$
	M8	0.25	$DiffFunc \rightarrow a\bar{b}$
	M9	0.25	$DiffFunc \rightarrow \overline{ab}$
	M10	0.5	$DiffFunc \rightarrow a\bar{b}$
	M11	0.5	$DiffFunc \rightarrow \overline{ab}$
NOR3_1	M0	0.5	$DiffFunc \rightarrow \bar{b}\bar{c}$
	M1	0.5	$Stuck-at-0$
	M2	0.25	$Stuck-at-0$
	M3	0.5	$DiffFunc \rightarrow \bar{a}\bar{c}$
	M4	0.125	$Stuck-at-0$
	M5	0.5	$DiffFunc \rightarrow \bar{a}\bar{b}$

	Mi	$\lvert C(i,tddb)\rvert$	F
MUX2_1	M0	0.5	$DiffFunc \rightarrow \bar{a}b + ac + ab\bar{c}$
	M1	0.5	$DiffFunc \rightarrow ac + bc$
	M2	0.5	$DiffFunc \rightarrow a + b$
	M3	0.5	$DiffFunc \rightarrow \bar{a}b$
	M4	0.5	$DiffFunc \rightarrow a + b + \bar{c}$
	M5	0.5	$DiffFunc \rightarrow \bar{a}b\bar{c}$
	M6	0.5	$DiffFunc \rightarrow \bar{a} + \bar{b} + c$
	M7	0.5	$DiffFunc \rightarrow a\bar{b}c$
	M8	0.5	$DiffFunc \rightarrow \bar{a} + c$
	M9	0.5	$DiffFunc \rightarrow ac$
	M10	0.5	$Stuck - at - 0$
	M11	0.5	$Stuck - at - 1$
ABorC	M0	0.375	$DiffFunc \rightarrow \bar{a}c + bc$
	M1	0.5	$DiffFunc \rightarrow b + c$
	M2	0.5	$DiffFunc \rightarrow a + c$
	M3	0.5	$DiffFunc \rightarrow c$
	M4	0.375	$DiffFunc \rightarrow a + b + c$
	M5	0.5	$DiffFunc \rightarrow ab$
	M6	0.625	$Stuck - at - 0$
	M7	0.375	$Stuck - at - 1$
NAND3_1	M0	0.5	$Stuck - at - 1$
	M1	0.25	$Stuck - at - 1$
	M2	0.125	$Stuck - at - 1$
	M3	0.5	$DiffFunc \rightarrow \overline{bc}$
	M4	0.5	$DiffFunc \rightarrow \overline{ac}$
	M5	0.5	$DiffFunc \rightarrow \overline{ab}$
XNOR2_1	M0	0.5	$DiffFunc \rightarrow \bar{a} + b$
	M1	0.5	$DiffFunc \rightarrow ab$
	M2	0.5	$DiffFunc \rightarrow ab$
	M3	0.5	$DiffFunc \rightarrow a + \bar{b}$
	M4	0.5	$DiffFunc \rightarrow a + \bar{b}$
	M5	0.5	$DiffFunc \rightarrow \bar{a}\bar{b}$
	M6	0.25	$DiffFunc \rightarrow a + \bar{b}$
	M7	0.25	$DiffFunc \rightarrow \bar{a}\bar{b}$
	M8	0.25	$DiffFunc \rightarrow \bar{a} + b$
	M9	0.25	$DiffFunc \rightarrow ab$
	M10	0.5	$DiffFunc \rightarrow \bar{a} + b$
	M11	0.5	$DiffFunc \rightarrow ab$
	M12	0.5	$Stuck - at - 0$
	M13	0.5	$Stuck - at - 1$
NAND4_1	M0	0.5	$Stuck - at - 1$
	M1	0.25	$Stuck - at - 1$
	M2	0.125	$Stuck - at - 1$
	M3	0.0625	$Stuck - at - 1$
	M4	0.5	$DiffFunc \rightarrow bcd$

	Mi	$\lvert C(i,tddb)\rvert$	F
	M5	0.5	$DiffFunc \rightarrow acd$
	M6	0.5	$DiffFunc \rightarrow abd$
	M7	0.5	$DiffFunc \rightarrow abc$
NOR4_1	M0	0.5	$DiffFunc \rightarrow \bar{b}\bar{c}\bar{d}$
	M1	0.5	$Stuck-at-0$
	M2	0.25	$Stuck-at-0$
	M3	0.5	$DiffFunc \rightarrow \bar{a}\bar{c}\bar{d}$
	M4	0.125	$Stuck-at-0$
	M5	0.5	$DiffFunc \rightarrow \bar{a}\bar{b}\bar{d}$
	M6	0.0625	$Stuck-at-0$
	M7	0.5	$DiffFunc \rightarrow \bar{a}\bar{b}\bar{c}$
OR4_1	M0	0.5	$DiffFunc \rightarrow b+c+d$
	M1	0.0625	$Stuck-at-1$
	M2	0.125	$Stuck-at-1$
	M3	0.5	$DiffFunc \rightarrow a+c+d$
	M4	0.25	$Stuck-at-1$
	M5	0.5	$DiffFunc \rightarrow a+b+d$
	M6	0.5	$Stuck-at-1$
	M7	0.5	$DiffFunc \rightarrow a+b+c$
	M8	0.9375	$Stuck-at-0$
	M9	0.0625	$Stuck-at-1$
NOT (ABorCorD)	M0	0.5	$DiffFunc \rightarrow \bar{c}\bar{d}$
	M1	0.5	$DiffFunc \rightarrow \bar{b}\bar{c}\bar{d}$
	M2	0.5	$DiffFunc \rightarrow \bar{a}\bar{c}\bar{d}$
	M3	0.4375	$DiffFunc \rightarrow \bar{a}\bar{b}\left(\bar{c}+\bar{d}\right)+\bar{c}\bar{d}$
	M4	0.375	$DiffFunc \rightarrow \bar{a}\bar{b}\bar{c}\bar{d}$
	M5	0.5	$DiffFunc \rightarrow \bar{a}\bar{d}+\bar{b}\bar{d}$
	M6	0.1875	$DiffFunc \rightarrow \bar{a}\bar{b}\bar{c}\bar{d}$
	M7	0.5	$DiffFunc \rightarrow \bar{a}\bar{c}+\bar{b}\bar{c}$
ABorCorD	M0	0.5	$DiffFunc \rightarrow c+d$
	M1	0.5	$DiffFunc \rightarrow b+c+d$
	M2	0.5	$DiffFunc \rightarrow a+c+d$
	M3	0.4375	$DiffFunc \rightarrow \left(a+\bar{b}\right)\left(c+d\right)+cd$
	M4	0.375	$DiffFunc \rightarrow a+b+c+d$
	M5	0.5	$DiffFunc \rightarrow \left(a+d\right)\left(b+d\right)$
	M6	0.1875	$DiffFunc \rightarrow a+b+c+d$
	M7	0.5	$DiffFunc \rightarrow \left(a+c\right)\left(b+c\right)$
	M8	0.8125	$Stuck-at-0$
	M9	0.1875	$Stuck-at-1$
AND2_1	M0	0.25	$Stuck-at-0$
	M1	0.5	$Stuck-at-0$
	M2	0.75	$Stuck-at-1$
	M3	0.5	$DiffFunc-b$
	M4	0.5	$DiffFunc-a$
	M5	0.25	$Stuck-at-0$

Glossary

ALU	Arithmetic Logic Unit
CS	Cross Section
EM	Electromigration
EOS	Electrical Overstress
ESD	Electrostatic Discharge
FIT	Failure In Time
HCI	Hot Carrier Injection
HEO	Highly Elliptical Orbit
HISREM	Hot Carrier Induced Series Resistance Enhancement Model
IC	Integrated Circuit
IRPP	Integral Rectangular Parallelpiped
LET	Linear Energy Transfer
MaCRO	Maryland Circuit Reliability-Oriented SPICE simulation method
MOSFET	Metal-Oxide Semiconductor Field-Effect Transistor
MTTF	Mean Time To Failure
RTL	Register Transfer Level
RPP	Rectangular Parallelpiped
SOFR	Sum-of-failure-rates
SPICE	Simulation Program Integrated Circuits Emphasis
SV	Sensitive Volume
SWIFI	Software Implemented Fault Injection
TDDB	Time Dependent Dielectric Breakdown

VHDL	Very High Speed Integrated Circuit Hardware Description Language

Bibliography

[1] N. G. Levenson, "Role of Software in Spacecraft Accidents," *Journal of spacecraft and Rockets*, vol. 41, 2004.

[2] M. Li, Y. Wei, D. Desovski, H. Najad, S. Ghose, B. Cukic, and C. Smidts, "Validation of a Methodology for Assessing Software Reliability," presented at the 15th IEEE International Symposium on Software Reliability Engineering, 2004.

[3] M. Li and C. Smidts, "Validation of a Methodology for Assessing Software Quality," Nuclear Regulatory Commission, Office of Nuclear Regulatory Research, Washington DC NUREG/CR-6848, 2004.

[4] F. R. Shapiro, "Entomology of the Computer Bug: History and Folklore," *American Speech*, vol. 62, pp. 376-378, 1987.

[5] R. K. Iyer and P. Velardi, "Hardware-Related Software Errors: Measurement and Analysis," *IEEE Transactions on Software Engineering*, vol. SE-11, pp. 223-231, 1985.

[6] J. Laprie, "Dependable Computing and Fault Tolerance: Concepts and Terminology," in *IEEE FTCS-15*. Ann Arbor, Michigan, 1985.

[7] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, and J. Laprie, "Fault Injection for Dependability Validation: A methodology and Some Applications," *IEEE Transactions on Software Engineering*, vol. 16, pp. 166-182, 1990.

[8] J. Karlsson, P. Liden, P. Dahlgren, R. Johansson, and U. Gunneflo, "Using Heavy-Ion Radiation to Validate Fault-Handling Mechanisms," *IEEE Micro*, vol. 14, pp. 8-23, 1994.

[9] M. C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault Injection Techniques and Tools," *Computer*, vol. 30, pp. 75-82, 2001.

[10] DBench, "Fault Representativeness," deliverable from Dependability Benchmarking (DBench) European IST project (project IST-2000-25425), 2002.

[11] G. A. Kanawati, N. A. Kanawati, and J. A. Abraham, "FERRARI: A Flexible Software-Based Fault and Error Injection System," *IEEE Transactions on Computers*, vol. 44, pp. 248-260, 1995.

[12] K. K. Goswami and R. K. Iyer, "Simulation of Software Behavior under Hardware Faults," in *IEEE FTCS-23*, 1993.

[13] T. A. Delong, "A Fault Injection Technique for VHDL Behavioral-Level Models," *IEEE Design and Test of Computers*, vol. 13, pp. 24-33, 1996.

[14] A. M. Amendola, "Fault Behavior Observation of a Microprocessor System through a VHDL Simulation-Based Fault Injection Experiment," in *EURO-DAC'96*, 1996.

[15] G. S. Choi, "FOCUS: An experimental environment for Validation of Fault Tolerance Systems: A Case Study of a Jet Engine Controller," in *IEEE International Conference on Computer Design*. Hamburg, Germany, 1989.

[16] D. Gil, "Fault Injection into VHDL Models: Analysis of the Error Syndrome of a Microcomputer System," in *24th Euromicro Conference*, 1998.

[17] J. Musa, "The operational profile in software reliability engineering: an overview," presented at the 3rd International Symposium on Software Reliability Engineering, 1992.

[18] R. Shukla, D. Carrington, and P. Strooper, "Systematic Operational Profile Development for Software Components," presented at 11th Asia-Pacific Software Engineering Conference, 2004.

[19] K. Chruscielski and J. Tian, "An operational profile for the Cartridge Support Software," presented at the 8th International Symposium on Software Reliability Engineering, 1997.

[20] S. Elbaum and S. Narla, "A Methodology for Operational Profile Refinement," presented at Annual Reliability and Maintainability Symposium, 2001.

[21] M. Gittens, H. Lutfiyya, and M. Bauer, "An Extended Operational Profile Model," presented at the 15th International Symposium on Software Reliability Engineering, 2004.

[22] R. V. Sandfoss and S. A. Meyer, "Input Requirementsneeded to Produce an Operational Profile for a New Telecommunications System," presented at the 8th International Symposium on Software Reliability Engineering, 1997.

[23] C. Dunn and J. McPherson, "Temperature Cycling Acceleration Factors in VLSI Applications," presented at IEEE International Reliability Physics Symposium Proceedings, 1990.

[24] R. Blish, "Temperature Cycling and Thermal Shock Failure Rate Modeling," presented at IEEE International Reliability Physics Symposium Proceedings, 1997.

[25] J. Gunn, R. CAMENGA, and S. MALIK, "Rapdi Assessment of the Humidity Dependence of IC Failure Modes by Use of HAST," presented at IEEE International Reliability Physics Symposium Proceedings, 1983.

[26] D. Peck, "Comprehensive Model for Humidity Testing Correlation," presented at IEEE International Reliability Physics Symposium Proceedings, 1986.

[27] J. Yue, *ULSI Technology*: McGraw-Hill, 1996.

[28] A. Acovic, G. Rosa, and Y. Sun, "A Review of Hot-Carrier Degradation Mechanism in MOSFETs," *Microelectron. Relib.*, vol. 36, pp. 845-869, 1996.

[29] E. Snyder, D. Campbell, S. Swanson, and D. Pierce, "Novel self-stressing test structures for realistic high-frequency reliability characterization," presented at IEEE International Reliability Physics Symposium Proceedings, 1993.

[30] J. Wang-Ratkovic, R. Lacoe, K. MacWilliams, S. Miryeong, S. Brown, and G. Yabiku, "New understanding of LDD CMOS hot-carrier degradation and device lifetime at cryogenic temperatures," presented at IEEE International Reliability Physics Symposium Proceedings, 1997.

[31] E. Takeda, C. Y. Yang, and A. Miura-Hamada, *Hot-Carrier Effects in MOS Devices*: Academic Press, 1995.

[32] M. Ohring, *Reliability and Failure of Electronic Materials and Devices*: Academic Press, 1998.

[33] D. G. Pierce and P. G. Brusius, "Electromigration: A Review," *Microelectron. Reliab.*, vol. 37, pp. 1053-1-72, 1997.

[34] A. Scorzoni, B. Neri, C. Caprile, and F. Fantini, "Electromigration in Thin-Film Interconnection Lines: Models, Methods and Results," *Materials Science Reports*, vol. 7, pp. 143-220, 1991.

[35] D. Young and A. Christou, "Failure Mechanism Models for Electromigration," *IEEE Transactions on Reliability*, vol. 43, pp. 186-192, 1994.

[36] J. R. Lloyd, "Reliability Modeling For Electromigration Faulire," *Quality and Reliability Engineering International*, vol. 10, pp. 303-308, 1999.

[37] J. F. Verweij and J. H. Klootwijk, "Dielectric Breakdown I: A Review of Oxide Breakdown," *Microelectronics*, vol. 27, 1996.

[38] J. S. Suehle, "Ultrathin Gate Oxide Reliability: Physical Models, Statistics, and Characterization," *IEEE Trans. Electron Devices*, vol. 49, pp. 958-971, 2002.

[39] R. Degraeve, B. Kaczer, and G. Groesenken, "Degradation and Breakdown in Thin Oxide Layers: Mechanisms, Models and Reliability Prediction," *Microelectron. Reliab.*, vol. 39, pp. 1445-1460, 1999.

[40] E. Y. Wu, E. J. Nowak, A. Vayshenker, W. L. Lai, and D. L. Harmon, "CMOS Scaling Beyond the 100-nm Node with Silicon-Dioxide-Based Gate Dielectrics," *IBM J. RES & DEV*, vol. 46, 2002.

[41] E. Wu, J. Sune, W. Lai, E. Nowark, L. McKenna, A. Vayshenker, and D. Harmon, "Interplay of Voltage and Temperature Acceleration of Oxide

Breakdown for Ultra-Thin Gate Oxides," *Solid-State Electronics*, vol. 46, pp. 1787-1798, 2002.

[42] E. Wu and J. Sune, "Power-Law Voltage Acceleration: A Key Element for Ultra-thin Gate Oxide Reliability," *Invited paper in special issue of Microelectronics Reliability*, 2005.

[43] J. D. Walter, "Methods to Account for Accelerated Semiconductor Device Wearout in Longlife Aerospace Applications," University of Maryland, 2003.

[44] W. Nelson, *Accelerated Testing: Statistical Models, Test Plans, and Data Analyses*. New York: John Wiley & Sons, 1990.

[45] H. Yang, J. S. Yuan, Y. Liu, and E. Xiao, "Effect of Gate Oxide Breakdown on RF Performance," *IEEE Transactions on Device and Materials Reliability*, vol. 3, pp. 93-97, 2003.

[46] R. H. Tu, E. Rosenbaum, W. Y. Chan, C. C. Li, E. Minami, K. Quader, P. K. Ko, and C. Hu, "Berkeley Reliability Tools - BERT," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, pp. 1524-1534, 1993.

[47] Y. Leblebici and S. M. Kang, "A One-Dimensional MOSFET Model for Simulation of Hot-Carrier Induced Device and Circuit Degradation," presented at IEEE International Symposium on Circuits and Systems, 1990.

[48] N. Hwang and L. Forbes, "Hot-Carrier Induced Series Resistance Enhancement Model (HISREM) of nMOSTFET's for Circuit Simulation and Reliability Projections," *Microelectron. Reliab.*, vol. 35, pp. 225-239, 1995.

[49] J. H. Stathis, R. Rodriguez, and B. P. Linder, "Circuit Implications of Gate Oxide Breakdown," *Microelectron. Reliab.*, vol. 43, pp. 1193-1197, 2003.

[50] B. Kaczer, R. Degraeve, M. Rasras, A. De Keersgieter, K. Van De Mieroop, and G. Groeseneken, "Analysis and Modeling of a Digital CMOS Circuit Operation and Reliability after Gate Oxide Breakdown: a Case Study," *Microelectron. Reliab.*, vol. 42, pp. 555-564, 2002.

[51] X. Li, J. Qin, B. Huang, X. Zhang, and J. B. Bernstein, "A New SPICE Reliability Simulation Method for Deep Submicron CMOS VLSI Circuits," *IEEE Transactions on Device and Materials Reliability*, vol. 6, 2006.

[52] P. C. Li, G. I. Stamoulis, and I. N. Hajj, "iProbe-d: a Hot-Carrier and Oxide Reliability Simulator," presented at IEEE Proceedings of International Reliability Physics Symposium, 1994.

[53] J. Segura, C. De Benito, A. Rubio, and C. F. Hawkins, "A Detailed Analysis of GOS Defects in MOS Transistors: Testing Implications at Circuit Level," presented at IEEE International Test Conference, 1995.

[54] J. Srinivasan, S. V. Adve, P. Bose, J. A. Rivers, and C. K. Hu, "RAMP: A Model for Reliability Aware MicroProcessor Design," IBM Research Report RC23048, 2003.

[55] JEDEC, "Failure Mechanisms and Models for Semiconductor Devices," JEDEC Publication No. 122B 2003.

[56] M. Brox, E. Wohlrab, and W. Weber, "A Physical Lifetime Prediction Method for Hot-Carrier Stressed P-MOS Transistors," presented at IEDM'91 Tech. Dig, 1991.

[57] R. R. Fritzemeier, H. T. Nagle, and C. F. Hawkins, "Fundamentals of Testability – A Tutorial," *IEEE Transactions on Industrial Electronics*, vol. 36, pp. 117-128, 1989.

[58] J. B. Sulistyo and D. S. Ha, "A New Characterization Method for Delay and Power Dissipation of Standard Library Cells," *VLSI Design*, vol. 15, pp. 667-678, 2002.

[59] J. B. Sulistyo, J. Perry, and D. S. Ha, "Developing Standard Cells for TSMC 0.25um Technology under MOSIS DEEP Rules," Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA Technical Report VISC-2003-01, 2003.

[60] "Reliability in CMOS IC Design: Physical Failure Mechanisms and their Modeling," MOSIS Technical Notes, http://www.mosis.org/support/technical-notes.html.

[61] "VCS MX Reference Guide," Synopsys Inc. 2006.

[62] "Synopsys Design Analyzer Reference Manual," Synopsys Inc. 2002.

[63] "IEEE Standard VHDL Language Reference Manual," IEEE Std 1076-1993 1993.

[64] B. Huang, X. Li, M. Li, J. B. Bernstein, and C. S. Smidts, "Study of the Impact of Hardware Fault on Software Reliability," presented at IEEE International Symposium on Software Reliability Engineering, 2005.

[65] Zilog, "Z80 Family CPU User Manual."

[66] D. Wallner: http://www.opencores.org/projects.cgi/web/t80/overview, 2002.

[67] "SDCC compiler," http://sdcc.sourceforge.net/.

[68] B. Huang, X. Li, M. Li, J. B. Bernstein, and C. S. Smidts, "Software-Specific Hardware Failure Profile," presented at AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, 2005.

[69] T. Karnik, P. Hazucha, and J. Patel, "Characterization of Soft Errors Caused by Single Event Upsets in CMOS Processes," *IEEE Transactions On Dependable and Secure Computing*, vol. 1, 2004.

[70] T. C. May and M. H. Woods, "Alpha-Particle-Induced Soft Error in Dynamic Memories," *IEEE Trans. Electron Devices*, vol. ED-26, pp. 2-9, 1979.

[71] T. Juknke and H. Klar, "Calculation of the Soft Error Rate of Submicron CMOS Logic Circuits," *IEEE Journal of Solid-State Circuits*, vol. 30, pp. 830-834, 1995.

[72] J. Ziegler, "Terrestrial Cosmic Rays," *IBM Journal of Research and Development*, vol. 42, pp. 117-139, 1998.

[73] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi, "Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic," presented at International Conference of Dependable Systems and Networks, 2002.

[74] E. L. Peterson, J. B. Langworthy, and S. E. Diehl, "Suggested Single Event Upset Figure of Merit," *IEEE Transactions on Nuclear Science*, vol. NS-30, pp. 448, 1983.

[75] E. L. Peterson, J. C. Pickel, J. H. Adams, Jr., and E. C. Smith, "Rate Prediction for Single Event Effects –A Critique," *IEEE Transactions on Nuclear Science*, vol. 39, pp. 1577, 1992.

[76] J. C. Pickel, "Single-Event Effects Rate Prediction," *IEEE Transactions on Nuclear Science*, vol. 43, pp. 483, 1996.

[77] "CREME96," https://creme96.nrl.navy.mil.

[78] A. J. Tylka, J. H. Adams, P. R. Boberg, B. Brownstein, and W. F. Dietrich, "CREME96: A Revision of the Cosmic Ray Effects on Micro-Electronics Code," *IEEE Transactions on Nuclear Science*, vol. 44, pp. 2150-2160, 1997.

[79] W. L. Bendel and E. L. Petersen, "Proton upsets in orbit," *IEEE Transactions on Nuclear Science*, vol. NS-30, pp. 4481, 1983.

[80] J. R. Letaw and E. Normand, "Guidelines for Predicting Single-Event Upsets in Neutron Environments," *IEEE Transactions on Nuclear Science*, vol. 38, 1991.

[81] E722-94, "Standard Practice for Characterizing Neutron Energy Spectra in terms of an Equivalent Monoenergetic Neutron Fluence for Rediation-Hardness Testing of Electronics," *Journal of ASTM International*, vol. 12.02, 2003.

[82] G. M. Swift, F. F. Farmanesh, S. M. Guertin, F. Irom, and D. G. Millward, "Single-Event Upset in the PowerPC750 Microprocessor," *IEEE Transactions on Nuclear Science*, vol. 48, pp. 1822, 2001.

[83] R. W. Berger, D. Bayles, R. Brown, S. Doyle, A. Kazemzadeh, K. Knowles, D. Moser, J. Rodgers, B. Saari, D. Stanley, and B. Grant, "The RAD750 - a Radiation Hardened PowerPC Processor for High Performance Spaceborne Applications," presented at IEEE Aerospace Conference, 2001.

[84] D. Rea, D. Bayles, P. Kapcio, S. Doyle, and D. Stanley, "PowerPC RAD750 - A Microprocessor for Now and the Future," presented at IEEE Aerospace Conference, 2005.