Modelling and Mitigation of Soft-Errors in CMOS Processors

Alireza Rohani

Modelling and Mitigation of Soft-Errors in CMOS Processors

Alireza Rohani

Members of the dissertation committee:

Prof. dr. ir.	G.J.M. Smit	University of Twente (promoter)
Dr. ir.	H.G. Kerkhoff	University of Twente (co-promoter)
Prof. dr. ir.	B.R.H.M Haverkort	University of Twente
Prof. dr. ir.	J.C. van de Pol	University of Twente
Prof. dr. ir.	K.L.M. Bertels	Delft University of Technology
Prof. dr.	H.S. Wunderlich	University of Stuttgart (Germany)
Dr.	D. Alexandrescu	iRoC Technologies (France)
Prof. dr.	P.M.G Apers	University of Twente (chairman and secretary)

TOETS

This work has been carried out as part of the Catrene project “TOETS” [CT302] and supported by the Netherlands Enterprise Agency.

CTIT

CTIT PhD. Thesis Series No. 978-90-365-3807-7

Center of Telematic and Information Technology
University of Twente, P.O. Box 217, NL-7500 AE, Enschede, The Netherlands

This thesis was printed by Gildeprint, the Netherlands.

ISBN 978-90-365-3807-7
DOI 10.3990/1.9789036538077

MODELLING AND MITIGATION OF SOFT-ERRORS IN CMOS PROCESSORS

DISSERTATION

to obtain
the degree of doctor at the University of Twente,
on the authority of the rector magnificus
Prof. dr. H. Brinksma,
on account of the decision of the graduation committee
to be publicly defended
on Friday, 12th of December 2014 at 16:45

by

Alireza Rohani

born on 13th July 1983,
in Damghan, Iran

This dissertation is approved by:

Prof. dr. ir.	G.J.M. Smit	University of Twente (promoter)
Dr. ir.	H.G. Kerkhoff	University of Twente (co-promoter)

Abstract

The topic of this thesis is about soft-errors in digital systems. Different aspects of soft-errors have been addressed here, including an accurate model to simulate soft-errors in a gate-level net list, a simulation framework to study the impact of soft-errors in a VHDL design and an efficient architecture to minimize the effect of soft-errors in a DSP.

The first two chapters of this thesis introduce the background knowledge with regard to soft-errors. Chapter three introduces a simulation framework to study the impact of soft-errors in complex digital systems modelled in the VHDL language. This framework has been introduced to resolve the enormous CPU time typically required in simulation-based soft-error experiments.

Chapter four introduces two realistic models that can simulate the impact of soft-errors in a 45-nm CMOS technology node at gate level. One of the approaches has been extracted from radiation testing along with using a transistor-level soft-error analysis tool. Another approach has been developed by analysing the behaviour of soft-errors in a 45-nm CMOS technology node.

In chapter 5, some unique features of DSPs have been exploited to introduce low-overhead soft-error mitigation architectures to minimize the impact of soft-errors in a DSP processor. This mitigation technique concerns irregular parts of a processor (such as the control unit and data path). The unique features of DSP processors are the existence of several functional units, a limited number of different opcodes in each functional unit and also a highly-repetitive instruction flow in a DSP workload. Moreover, the mitigation method which has been developed for a single core has been applied to a multi-core environment in chapter 6 to propose a soft-error mitigation technique for multi-core architectures.

As a conclusion, based on simulated data and experiments, this thesis proposes a methodology to investigate the impact of soft-errors during the design phase of a digital system.

This page intentionally left blank.

Nederlandse samenvatting

Het onderwerp van dit proefschrift betreft sporadische fouten in digitale systemen. Deze sporadische fouten worden veelal aangeduid als *soft errors*. Verschillende aspecten van soft errors worden belicht in dit proefschrijft, waaronder een accuraat simulatiemodel om soft errors op poort-niveau te emuleren, een simulatieraamwerk om de gevolgen van soft errors in een VHDL-ontwerp te bestuderen en een efficiënte architectuur om de effecten van soft errors in DSP's te minimaliseren.

De eerste twee hoofdstukken van dit proefschrift behandelen de achtergrondkennis met betrekking tot soft errors. Hoofdstuk drie introduceert een simulatieraamwerk om de gevolgen van soft errors in complexe, in VHDL beschreven digitale systemen te onderzoeken. Het raamwerk wordt geïntroduceerd om extreem lange rekentijden, die normaliter gepaard gaan met simulatiegebaseerde soft error-experimenten, te voorkomen.

Hoofdstuk vier introduceert twee realistische modellen die de effecten van soft errors op poort-niveau emuleren in 45-nm CMOS-technologie. De eerste methode is gebaseerd op stralingsmetingen tezamen met een soft error analyse-applicatie op transistorniveau. De tweede methode is ontwikkeld op basis van de analyse van de fysieke gevolgen van soft errors in 45-nm CMOS-technologie.

In hoofdstuk 5 wordt een architectuur met lage complexiteit geïntroduceerd waarmee de effecten van soft errors in DSP's teniet worden gedaan door gebruik te maken van enkele speciale eigenschappen van DSP's. Deze methode werkt op de onregelmatige onderdelen van de processor (zoals de regeleenheid en het datapad). De speciale eigenschappen van DSP's betreffen 1) het bestaan van verschillende functie-eenheden, 2) een beperkt aantal opcodes in elke functie-eenheid en 3) programma's met veel herhaaldelijk uitgevoerde instructies. Daarnaast kan de methode, hoewel deze ontwikkeld is om soft errors in single-core systemen te verhelpen, ook toegepast worden in een multicore context, zoals beschreven in hoofdstuk 6.

Tot slot, is er een methode ontwikkeld op basis van simulatieresultaten en experimenten om al tijdens de ontwerpfase rekening te houden met soft errors en de gevolgen daarvan te minimaliseren.

This page intentionally left blank.

Acknowledgements

Looking back to my life reminds me of many great people who have influenced me to become a better human being. To mention a few, I would like to thank Mr. Abolfazl Khalilnejad, my first English teacher back in high school who was not only one of the greatest teachers I have ever had, but also a symbol of responsibility and discipline to me. I would also like to give my appreciation to Dr. Hamid Reza Zarandi, my supervisor during my master degrees at Amirkabir University of Technology.

Starting my PhD in the Netherlands back in 2010 went smoother with having wonderful people around me. To name few, I would like to thank Masi Amirpour, Pouria Zand, Marziyeh Malekinajad, Siavash Aflaki, Mitra Baratchi, Sina Behfard, Alireza Masum, Zahra Taghikhani, Amirhossein Ghamarian, Wim Korevaar and Majid Bahrepour.

I would like to appreciate my promoter, Prof. Gerard Smit, for giving me the opportunity to carry out my PhD in CAES group. I would like to give my greatest appreciation to my daily supervisor, Dr. Hans Kerkhoff. He has not only helped me regarding my PhD research, but I learned responsibility, dedication and morality from him. I could not think of any better supervisor than Hans.

I would like to thank people of the CAES group, especially Muhammad Aamir Khan, Ahmed Ibrahim, Hassan Ebrahimi, Andreina Zambrano, Jinbo Wan, Yong Zhao, Wim Korevaar, Robert de Groote, Koen Blom, Marco Gerards, Philip Hölzenspies and Bert Molenkamp. I especially would like to thank Marlous Weghorst, Thelma Nordholt – Prenger, Nicole Baveld, and Bert Helthuis that made the CAES group a more pleasant environment to work.

I would also like to thank my paranymphs, Anja Kolesnichenko and Amir Meshkat for helping me during my defence. I am thankful of Wim Korevaar how helped me to translate the summary of this thesis to Dutch.

I would like to thank my parents, Masoume Amirahmadi and Nematollah Rohani who taught me self-devotion. I believe being raised by those made me a person who is eager of pursuing his dreams. Also, thanks to my two lovely and amazing sisters, Aida and Mitra. There were many moments in my life when I missed them here in the Netherlands.

And special thanks to my lovely and beautiful wife, Mahroo Zandrahimi. I met Mahroo during my studies in 2009 and she made my academic life special as

well. She always understood my work situation, especially during this last year when I was travelling between Enschede and Delft. She means an endless source of kindness, love and support to me. I also like to thank my father-in-law, Dr. Morteza Zandrahimi for his support.
Alireza Rohani, November 2014

Contents

List of Acronyms

AC	Accumulator
ADIRUs	Air Data Inertial Reference Units
ALU	Arithmetic Logic Unit
ATPG	Automatic Test Pattern Generation
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CR	Checkpoint and Recovery
DAC	Duplication And Comparison
DMR	Modular Redundancy
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processing
DUE	Detected-Unrecoverable-Error
DWC	Duplication With Comparison
EDA	Electronic Design Automation
EDAC	Error Detection and Correction Codes
EMI	Electro Migration Interference
ESA	European Space agency
FIR	Finite Impulse Response
FIS	Fault Injector Signal
FIT	Failure In Time
FIUs	Fault Injection Units
FPGA	Field Programmable Gate Arrays
GLN	Gate Level Net-list
HBFI	Hardware-Based Fault Injection
HDL	Hardware Description Language
ICs	Integrated Circuits
LCU	Local Control Unit
LSB	Least Significant Bits
LUT	Look-Up Table
MEU	Multiple Memory Upset
MeV	Mega electron Volt
MOS	Metal Oxide Semiconductor
NASA	Aeronautics and Space Administration
NoC	Network on Chip
PC	Program Counter
QoS	Quality of Service

RIIF	Reliability Information Interchange Format
RISC	Reduced Instruction Set Computer
RMT	Redundant Multi-Threading
ROM	Read only Memory
RTL	Register Transfer Level
SDC	Silent Data Corruption
SDF	Standard Delay Format
SEE	Single Event Effect
SEM	Soft Error Mitigation
SER	Soft Error Rate
SET	Single Event Transient
SEU	Single Event Upset
SoC	System on Chip
SPARC	Scalable Processor Architecture
SRAM	Static Random Access Memory
STEM	Soft and Timing Error Mitigation
TMR	Triple Modular Redundancy
VLIW	Very long Instruction Word
VLSI	Very Large Scale Integration
VPI	Verilog Programming Interface

CHAPTER 1

Introduction

1.1 Introduction

The unprecedented progress of CMOS technology has enabled digital systems to be emerged ubiquitously in every aspect of our life. Nowadays it is difficult to imagine a task in which digital computing is not involved. This includes portable electronic systems like laptop computers, cellular phones, and music players up to different embedded computing systems in the medical, automotive and avionics industry. The sharp rate of growth in CMOS technology has been sustained by shrinking the minimum technology sizes of transistors to smaller and smaller dimensions along with the continuous reduction in the operating and threshold voltages [Hir02]. While this technology scaling has provided modern VLSI systems with a higher performance and lower power consumption, their sensitivity to certain types of faults has dramatically increased. As a result, the reliability of a system which are implemented in a modern CMOS process node is a key concern [Cao09].

The required level of reliability of a device depends on different parameters. For example, a very brief momentary malfunction in an audio device embedded in a car might cause no harm other than inconvenience and a slight reduction of Quality of Service (QoS). However, even a slight temporary malfunction in the lane-detection system of a modern car might lead to the loss of human life.

As a real example, the sudden dive of a Qantas flight, back in 2008 [Wik08] will be briefly discussed. The airplane had to carry out an emergency landing due to an inflight accident featuring a pair of sudden un-commanded pitch-down manoeuvres that resulted in serious injuries to many of the passengers. The final report issued in 2011 concluded that the accident occurred due to a failure mode affecting one of the aircraft's three air-data inertial reference units (ADIRUs). The failure mode was further tracked down to design limitations, in which in a very rare and specific situation, multiple spikes were formed in one of the ADIRUs which in turn could command the aircraft to pitch down.

A primary source of momentary malfunction of advanced CMOS computing is known as soft-errors [Nic11]. A soft-error, also referred to as

Single Event Effect (SEE), can occur when an energetic particle from extra-terrestrial space or from impurities in packaging material hits the surface of a CMOS transistor. As a consequence of this collision, a current glitch might be generated in the transistor channel, which subsequently results into a voltage glitch at a circuit node. This voltage glitch has the potential to propagate into the subsequent logic gates of the system and can even cause a functional failure of the system. Soft-errors can occur in any internal node of a circuit, at random times. Depending on the timing of the clock, glitches can propagate to higher hierarchical levels and load a wrong value into a latch or flip-flop. For example, in Figure 1.1, a glitch has been generated in logic gate1 at time T1. This glitch has reached the positive edge-triggered flip-flop-1 at time T2. Because the positive clock-edge for flip-flop-1 is occurring at time T2, an erroneous value which is now 1 instead of 0, will be stored in the flip-flop. However, this erroneous value will not reside permanently in the flip-flop; so when a new value reaches the positive edge-triggered flip-flop in the next clock-cycle, the flip-flop stores the new value. Hence, the output of the flip-flop will be high for one clock-cycle.

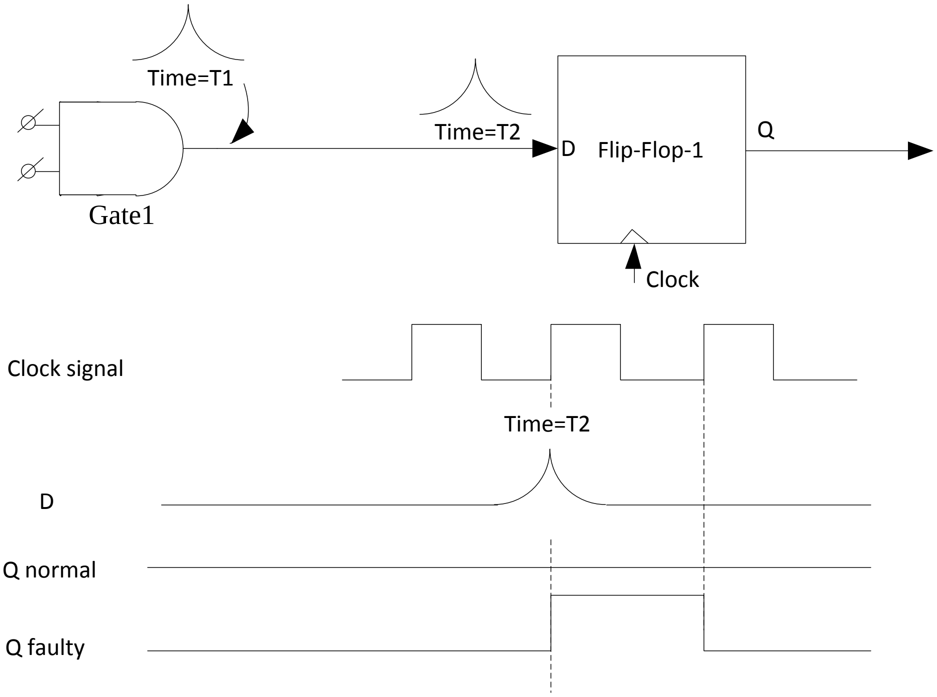


Figure 1.1. Loading an erroneous value in a flip-flop due to a glitch in a circuit.

Historically speaking, the first concern of soft-errors emerged during the nineties when several studies repeatedly showed that the majority of system failures in modern digital circuits can be categorized as soft-errors, rather than traditional manufacturing errors or permanent faults [Gre94]. Recent VLSI technology trends such as shrinking the transistor features has helped the design of transistors for higher integration density, higher performance and lower power consumption. Higher integration densities, increase in operating frequencies along with reduction of operating supply voltage all have considerably increased the soft-error vulnerability of current digital systems [Cao09]. Moreover, the increased use of wireless technology, such as Wi-Fi and mobile phone transceivers has increased the hostility of our environment as a threat from soft-errors.

The amount of erroneous glitches in a transistor depends on many parameters, being the speed of the circuit, the environment where the system is being used, altitude, etc. While the soft-error rate of individual transistors are projected to increase with every new generation of VLSI, incorporating more and more transistors into a device even exacerbates the soft-error problem. Taking into account all the above-mentioned consequences of technology scaling, it has been consistently proven that soft-errors are a major threat of circuit/system reliability for the sub-100nm technology [Kar04]. Figure 1.2 shows the rate of soft-errors for a matured technology as well as the projected soft-error rate for the 16nm process node. As can be seen in this Figure, for a technology node larger than 100nm, the soft-error rate was not a concern at all. However, if the technology shrinks to 45nm, a typical Intel processor chip can experience 20 failures in its life time. This number will increase exponentially with shrinking dimensions in technology.

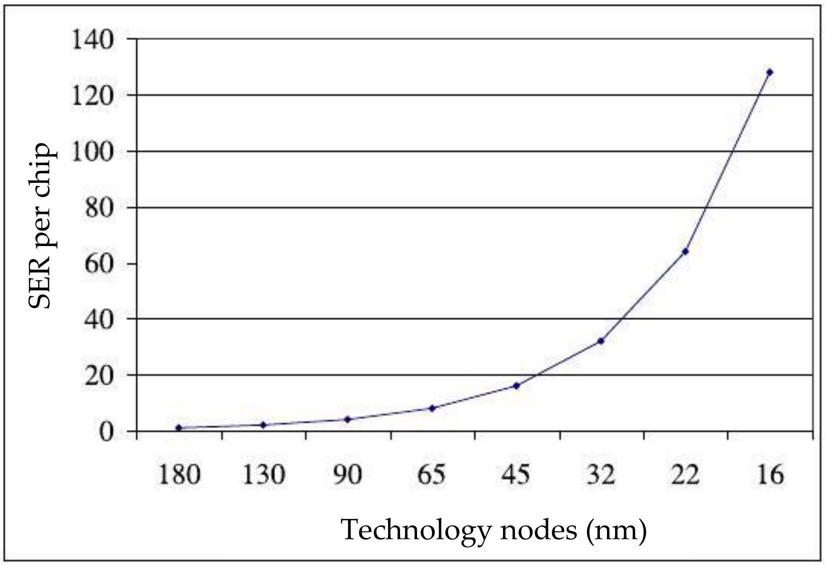


Figure 1.2. Soft-error rate in recent process technology nodes [Kar01].

Historically, soft-errors have been mainly of concern to those systems designed to be used in safety-critical systems, or systems that were going to be used in hostile environments, such as satellites, spaceships and aircrafts. Those particular applications could benefit from expensive fabrication technology and complex fault-tolerant solutions to reduce the impact of soft-errors. However, those expensive advances in developing fault-tolerant designs will not be cost-effective for mass-produced consumer products. Furthermore, emerging issues like process variations have introduced additional sources of soft-errors [Xfu09] which exacerbate the sensitivity of present computer systems to soft-errors.

As a conclusion, the concerns of soft-errors for current embedded systems are not limited to space applications anymore, since device scaling accompanied by supply power reduction has caused reliability issues for embedded system manufactured in sub 100nm process nodes.

1.2 Motivation and problem statement

In the TOETS (Towards One European Test Solution) project, developing new methods to deal with failures occur in sub 100nm technology nodes are investigated. Our special concern in this thesis is to develop a soft-error hardened system to be used in automotive industry. At the time of writing of this thesis (2014), full-hybrid and X-by-wire cars are already driving in the streets (such as Tesla [Tes14] and Nissan [Nis14]). Moreover, the first auto-drive car has been authorized to be emerged on the streets of the USA (Google project) [Goo14].

So, it is no longer possible to consider the automotive industry as a low-critical domain regarding soft-errors. For example, Toyota had one of the biggest recalls of the automotive industry across the globe in 2010 to fix the electronic systems of its cars. The problem was claimed to be related to the very sensitive parts of the car with regard to soft-errors [Men12, Fin13a, Fin13b]. It was shown that a glitch in the electronic system of the car could influence the functionality of its acceleration system.

The other important concern regarding the automotive industry is the total cost, which limits the usage of expensive soft-error mitigation solutions. As a result, the digital architect has to develop an electronic device that has an acceptable vulnerability level concerning soft-errors, while its final cost/performance is acceptable to be used in a car.

Since safety-critical applications in a car are more towards DSP applications, such as lane detection or distance prediction, our main goal in this work is to develop a soft-error hardened architecture for DSP processors which satisfies the performance criteria.

This thesis addresses the soft-error problems occurring in DSP processors fabricated in a 45nm technology node. Several aspects of soft-errors, from an architectural soft-error model to proposing light-weight architectural solutions for detection and correction of soft-errors in single and multicore DSP systems will be studied throughout this thesis. Specifically, the problem statement can be stated as follows:

- A soft-error analysis framework to assess the effect of soft-errors in complex processors needs to be investigated. Traditional simulation-based fault-injection frameworks are slow and not practical to conduct soft-error analysis on complex DSP processors. So, accelerated frameworks are essential for soft-error analysis on complex digital processors.
- An efficient model to emulate the impact of soft-errors in sub 100nm technology nodes needs to be developed. As the CMOS technology of implementation shrinks beyond 45nm technology nodes, already developed fault models are not practical anymore. A realistic and accurate simulation model of soft-errors in a 45nm and beyond technology nodes is essential in order to study the impact of soft-errors in complex digital processors.
- While there are many general soft-error mitigation mechanisms in digital processors, we are especially interested to use the unique characteristics of DSP processors, such as existence of identical resources, to develop an efficient fault-tolerant mechanism. Moreover, we want to investigate unstructured parts of a processor, such as the data-path or control-logic, since these two units cannot be protected by conventional fault-tolerance methods.
- Since the increasing usage of multicore architectures in modern digital systems, we also want to develop a fault-tolerant architecture customized for multicore architectures consisting of DSP cores. Moreover, the existence of several identical cores in multicore architecture might be very useful for soft-error mitigation mechanisms.

1.3 Outline of the thesis

The remainder of this thesis has been organized as follows:

Chapter 2 describes the basic terminology of soft-errors, including the origin of soft-errors and a survey of the state-of-the-art methods dealing with detection and correction of soft-errors in processors.

The details of our simulation-based fault-injection framework will be discussed in chapter 3. This framework is able to inject conventional logic gate-level fault models, like a fixed-duration glitch, into a Hardware-Description-Language (HDL)-based design. In chapter 4, a realistic simulation model for soft-errors in 45nm process nodes will be proposed. Two unique techniques to detect and correct soft-errors in DSP processors are described in chapter 5. The framework provided in chapter 3 along with the realistic fault model described in chapter 4 form the basis of two advanced methods being developed to harden a DSP processor with respect to soft-errors. In chapter 6, the architecture of a multi-core design will be used to develop a detection and correction method. Since chapter 6 combines the fault-tolerant architecture of a single core from chapter 5, this chapter must be read before reading chapter 6. Finally, in chapter 7, conclusions are given and some suggestions for future work are provided.

References

[Cao09] Y. Cao, P. Bose, J. Tschanz, “Reliability challenges in Nano-CMOS design,” IEEE Design and Test of Computers, pp. 6-7, 2009.

[Fin13a] Financial Times Press, www.sddt.com, 2013.

[Fin13b] Financial Times Press, www.eetimes.com, 2013.

[Goo14] Google Self-Driving Car Project, www.GoogleSelfDrivingCars.com, 2014.

[Gre94] L. Gregory, S. Gwan, K. Ravishankar, “Device-level transient fault modeling,” in International Symposium on Fault-Tolerant Computing, pp. 86-94, 1994.

[Hir02] M. Hirose, “Challenge for future semiconductor development,” in Microprocessors and Nanotechnology Conference, pp. 2-3, 2002.

[Kar01] T. Karnik, B. Bloechel, K. Soumyanath, “Scaling trends of cosmic ray induced soft-errors in static latches beyond 180nm,” in International Symposium on VLSI Circuits, pp. 61-62, 2001.

[Kar04] T. Karnik, P. Hazucha, J. Patel, “Characterization of soft-errors caused by Single-Event-Upset in CMOS processes,” in IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 2, pp. 128-143, 2004.

[Men12] Report by MentorGraphics, www.chipdesignmag.com, 2012.

[Nic11] M. Nicolaidis, “Soft-errors in Modern Electronic Systems,” in Frontiers in Electronic Testing, ISBN 978-1-4419-6993-4, 2011.

[Nis14] www.nissanusa.com/electric-cars/leaf, 2014.

[Tes14] www.teslamotors.com, 2014.

[Xfu09] X. Fu, T. Li, J. A. B. Fortes, “Soft-error vulnerability aware process variation mitigation,” in International Symposium on High Performance Computer Architecture, pp. 93-104, 2009.

[Wik08] WikiPedia, http://en.wikipedia.org/wiki/Qantas_Flight_72, 2008.

CHAPTER 2

Sources, Terminology and Evaluation Methods of Soft-Errors

ABSTRACT- This chapter will cover the terminology of soft-errors, discuss the sources of soft-errors and also different evaluation methods to assess the vulnerability of a system with regard to soft-errors will be explained. Moreover, the details of our case study the Xentium processor, will be presented at the end of this chapter. It will serve later on as a test bench in developing a fault-injection framework, a new model for soft-errors and also its architecture will be modified to develop a reliable and low overhead DSP architecture to mitigate soft-errors.

2.1 Introduction

Until a decade ago, there was no consistency on whether it would make sense to invest in the mitigation of soft-errors in digital circuits or not. In general, a soft-error does not concern ordinary and low-critical applications. For example the cell-phone or audio industry is not concerned about soft-errors at all. However, if a correct and timely operation of a system is critical, especially in harsh environments, soft-errors will be an issue for sure. Some examples of critical systems are: the break system in modern electrical cars (drive-by-wire cars), electronic systems of an airplane or the communication backbone of a satellite. In these systems, the correct functionality of the system can be lost, temporarily or permanently, by the effect of soft-errors. If the impact of soft-errors is momentary, then a short malfunction will appear in the device. If the error manifests in the system, it might be required to reset the system completely, which can be sometimes very costly in terms of performance loss. This because the entire workload needs to be executed again.

Since the nature of these temporary malfunctions are quite random, it is very hard to trace a failure which has been caused by a soft-error. These soft-error induced failures are even more harder to tackle when new information has already been loaded into the logic that has been affected by soft-errors.

Another concern which makes tackling soft-error induced failures very hard, is the limitation of traditional test methods, such as Automatic Test Pattern Generation (ATPG). Because soft-errors appear and disappear in a very brief period of time, a permanent isolation of an affected net or logic gate is not practical in dealing with soft-errors.

As a result, all the methods that deal with soft-errors should be built based on an online detection and correction mechanism to mask the effect of soft-errors as soon as possible. On the other hand, a failure which has been induced by a soft-error is not reproducible, since it is random in nature; hence the online soft-error mechanism should be able to stop the propagation of a soft-error as soon as possible. One of the solutions which can be used to prove that a soft-error has caused a failure in a system, is to log every status of a system and then trace the root of the problem. However, it is generally too costly to log the status of all the components of a design at every instance of time.

After the emerging of soft-error induced failures in modern digital systems during the nineties, different industrial sectors started research programs to address the problem of soft-errors. To name a few: Intel, IBM and Fujitsu in the semiconductor sector, Boeing, Airbus, Ericsson-Saab Avionics in the avionics sector, and the European Space Agency (ESA) and National Aeronautics and Space Administration (NASA) in space applications. As a real case of a soft-error induced failure, some random failures were found in a computer on a commercial aircraft in 1993 [Ols93, Yuh11]. The circuit which was affected by the random malfunctioning was a 256 kilo-bit SRAM which showed failures at a rate of one error per eighty days. Moreover, there were some reports by IBM and Boeing in which a strong correlation between the rate of random malfunctioning and the altitude above sea level of the aircraft electronic system was recorded [Tab93]. Apart from these two well-known examples of soft-errors in digital systems, some other examples induced by soft-errors in the semiconductor industry have highlighted the importance of soft-error measurements in the electronic design industry. Some examples have been shortly listed in the next paragraph based on examples from [Yuh11].

A phenomenon which is known as the Hera problem has been reported by IBM [Zie96]. During those years, IBM observed an increase in the rates of failures in Large Scale ICs (LSI) memories manufactured in the USA. Surprisingly, identical memories which were produced in Europe did not have this problem. The problem was traced back to the radiation which was emitted from the packaging material of a ceramic package. The problem was further traced back to impurities inside the ceramic packaging which emit radioactive rays and caused the memory cells to toggle their values randomly in time.

The second example is a problem being observed in a data server line, the Enterprise of Sun [For00]. The server occasionally crashed for a brief amount of time. The rate of failures was as high as four times in one month and they were induced by high sensitivity of memory cells with regard to soft-errors.

Another example concerned Cisco systems [Cis03]; some routers showed random failures caused by radiation-induced soft-errors. After Error Detection and Correction Codes (EDAC) [Nic11] were implemented in the memories, the rate of soft-errors diminished.

The rest of this chapter serves as an introduction to soft-errors. First, the terminology of soft-errors will be discussed. Then, the origin of soft-errors will be covered. Different methods to evaluate the vulnerability of systems against soft-errors will be discussed. Finally, the details of our case study, which is the Xentium processor [Rec11] will be provided. This processor will be used to analyse the impact of soft-errors in a complex digital system and also for the development of efficient methods to mitigate soft-errors.

2.2 Terminology

This section provides the common terminology which is being used by the soft-error community [Nic11, Sha11].

The main cause of soft-errors in integrated circuits are high-energy particles coming from extra-terrestrial sources or from inside chip packaging materials. In case an energetic particle hits a CMOS transistor, it has the potential to produce a localized ionization which is able to change the data which has been latched in a flip-flop or a latch. If a particle has sufficient energy to change the charge content of a memory from 0 to 1, or vice versa, this phenomenon is called Single Event Upset, known as SEU [Bau02, Sha11]. However, this change in the content of the memory is not a permanent one, such as errors caused by stuck-at-0 or stuck-at-1 faults

[Cro99]. So if the affected latch or flip-flop is loaded with new data, the impact of the SEU will be masked. However, in many situations the erroneous value has the potential to propagate into the system before overwriting of data occurs. In this case, the SEU has the potential to modify the entire functionality of a system. These kind of errors are called soft since the actual hardware of the circuits is not permanently damaged. Hence, if the system is reset or is reloaded with the proper state, the system can operate correctly again.

Figure 2.1 shows the moment when a high-energy particle hits a CMOS transistor. If the high-energy particle has sufficient energy, which is more than 1 Mega-electron-Volt (MeV), it has the potential to deposit a dense track of electron-hole pairs as they pass through a p-n junction [Shi02]. Some of the deposited charge will be absorbed by the gate of the transistor and form a short duration pulse of current at the internal circuit node. This short current pulse is depicted in Figure 2.2. This figure shows that a current pulse with maximum amplitude of 600μA has been produced by the particle. The duration and amplitude of this momentary pulse depends on the technology of implementation of the transistor, which can be 45nm, 22nm, etc., the type and energy of high-energy particle as well as the temperature.

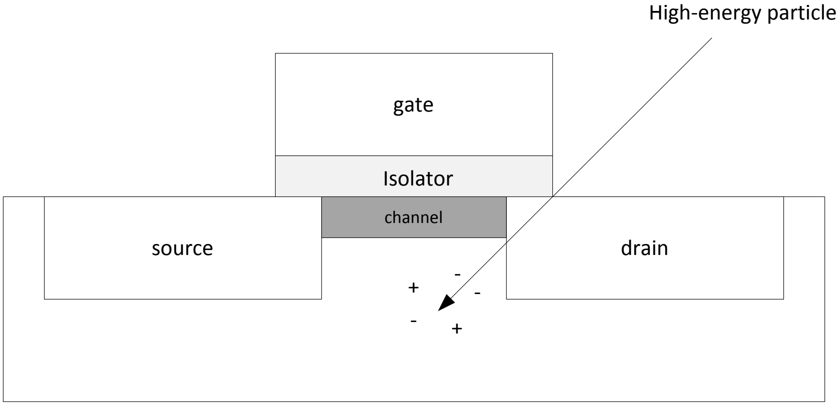


Figure 2.1. Striking a transistor by a high-energy particle.

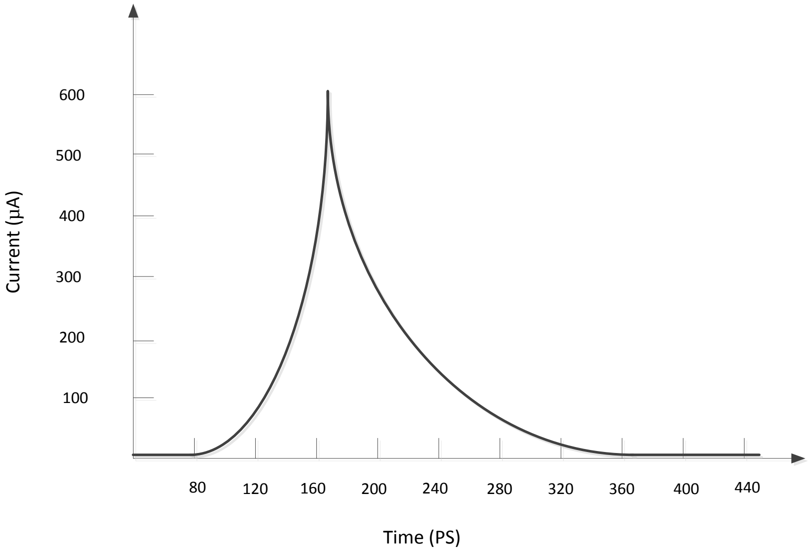
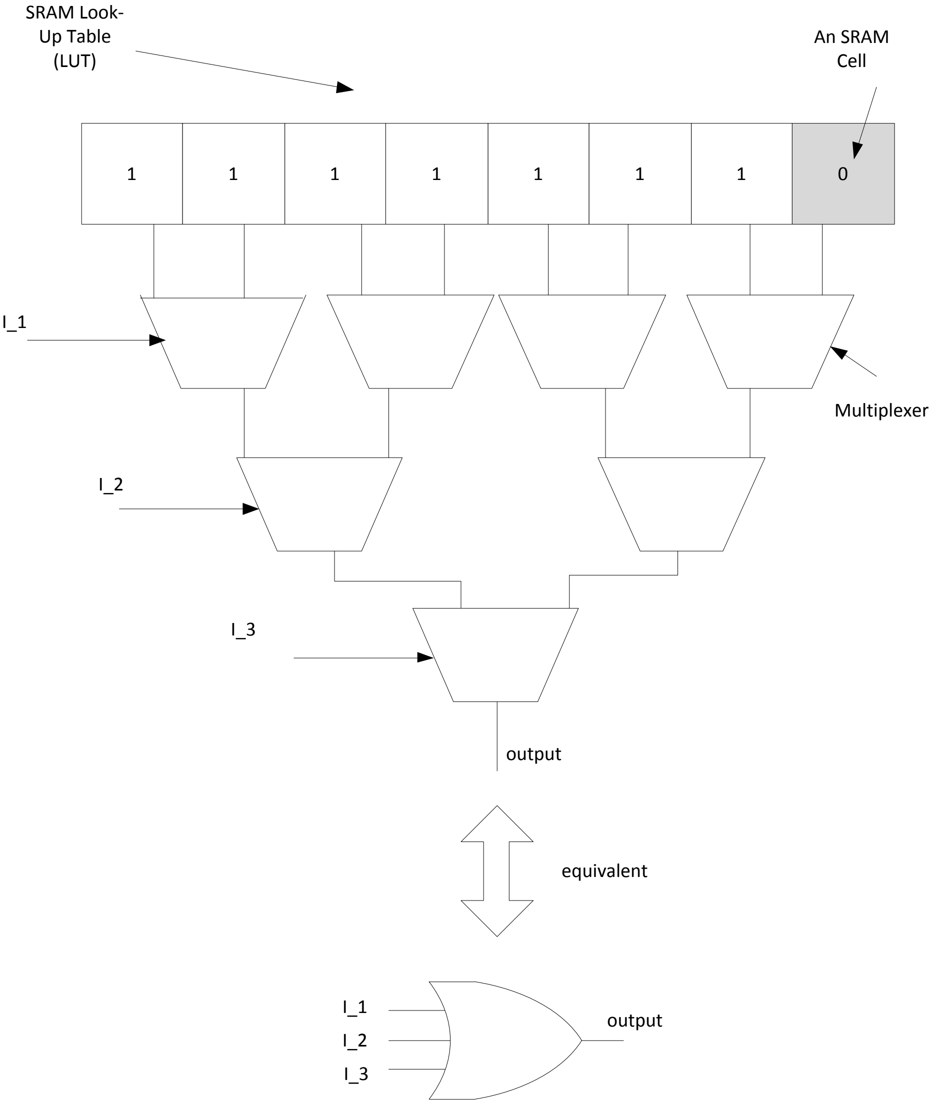
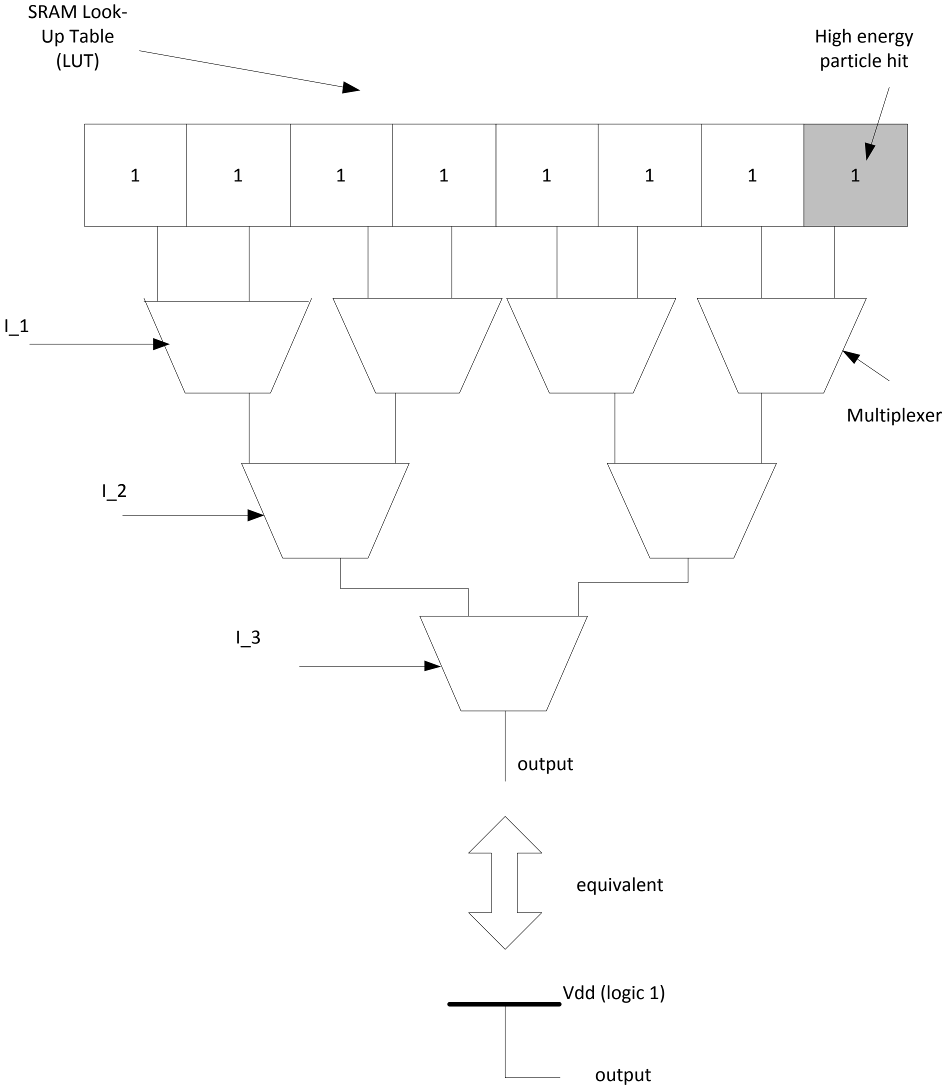


Figure 2.2. The produced perturbation caused by a high-energy particle.

Figure 2.3a shows a sequence of SRAM cells which have been configured as a Look-Up Table (LUT) in order to implement a logic OR function. Suppose that a radiation particle has hit the last SRAM cell (Figure 2.3b) and changed the stored value from 0 to 1. In this situation, the logic which will be implemented by the new configuration is a permanent stuck-at-1 value connected to Vdd; this has been shown in the equivalent logic gate in Figure 2.3b. It will be shown later on that error detection and correction codes are a powerful mechanism to mitigate this kind of errors.


SRAM Look-Up Table (LUT)
An SRAM Cell
1
1
1
1
1
1
1
0
I_1
Multiplexer
I_2
I_3
output
equivalent
I_1
I_2
I_3
output

a)



b)

Figure 2.3. A soft-error in a Look-Up Table. a). the correct operation of the Look-Up Table. b). the erroneous operation of the Look-Up Table.

Another phenomenon caused by soft-errors is the Single Event Transient (SET) which occurs if a momentary pulse (glitch) is generated at the output of a logic gate. This glitch has the potential to traverse through other combinational logic gates and reach a flip-flop or logic gate input in the succeeding hierarchy. If the clock-edge occurs at the same time when the glitch reaches a flip-flop input, the erroneous value will be latched into the flip-flop and the status of the circuit will be changed.

Figure 2.4 shows the propagation of a SET in several logic gates and reaching a memory cell. As can be seen in this figure, in the normal situation the value of 0 should be stored in the flip-flop, but as a result of a particle hit, the erroneous value of 1 has been latched in the flip-flop. This phenomenon is different from the SEU, since the value of the flip-flop has not been changed directly, but a wrong value has been produced by the combinational logic and then captured by the flip-flop. This type of error is very difficult to handle.

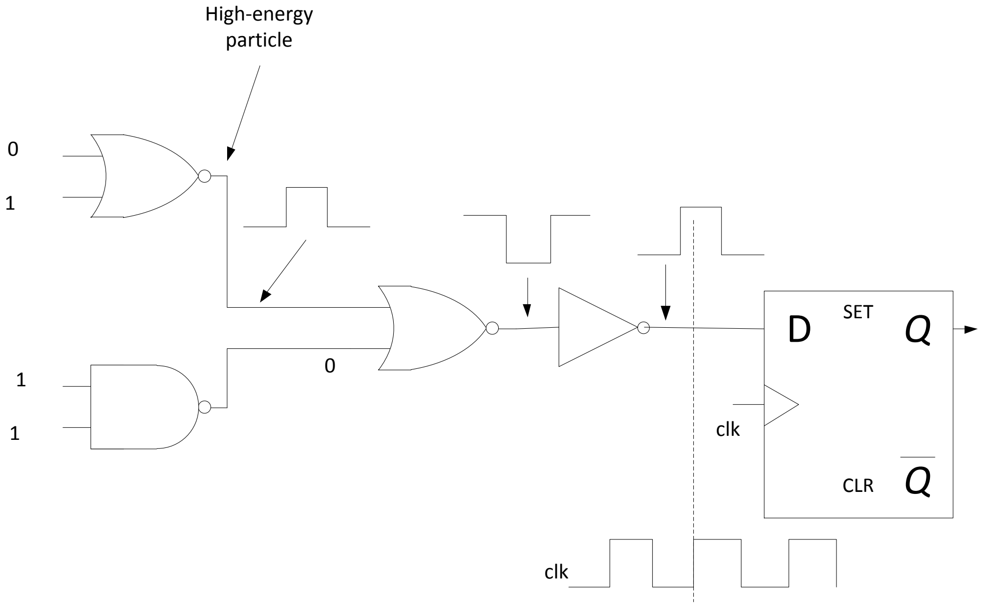


Figure 2.4. Propagation of a SET in the combinatorial part of a circuit.

A metric which is used to refer to soft-errors is the frequency of occurrence of errors. This metric is commonly referred to as the soft-error rate or SER. The SER depends on many factors including altitude above sea level and temperature.

In the following section, the origin of soft-errors and their occurrence rate will be discussed.

2.3 The sources of soft-errors

There are multiple physical phenomena that induce soft-errors in a MOS digital circuit, the two dominant ones being neutron and alpha particles. The effect of these two sources are quite different from each other and they will be discussed in different subsections.

2.3.1 Neutrons

High-energy neutrons are one of the most dominant sources of soft-errors [Wan07]. Close to the orbit of planet Earth, the prime source of neutrons is cosmic radiation. The cosmic rays are radiation fluxes which consist of high-energy particles originating from outer space. There are two main types of cosmic radiation that induce soft-errors: solar cosmic rays and galactic cosmic rays [Anc03].

Solar cosmic rays originate from the sun and are primarily composed of proton and helium particles. Protons dominate the solar cosmic ray flux and are typically low energy particles. Galactic cosmic rays are high-energy particles that penetrate into the orbit of planet Earth from the outside of our solar system. In general, galactic cosmic rays typically have a very large energy and are the cause of most of the soft-errors in satellite and aerospace avionics.

When the galactic cosmic radiation reaches ground sea level, the flux of particles is primarily composed of muon, proton, neutron, and pion particles [Zie81]. Neutrons are the most likely particles to cause a soft-error in a circuit since they have the highest energy.

As a result of the interaction with the atmosphere, the radiation flux depends on the altitude. For example, there is about a 10 times difference in flux between the sea level and an altitude of 10000 feet [Zie81]. Thus, computers operating at a high altitude, for example in aircrafts, can experience soft-error rates in excess of an order of magnitude than they would have at sea level [Wan07].

The influence of neutron particles can be reduced to negligible levels with very strong physical shielding. For example, each 33 centimetres of concrete can reduce the neutron flux by approximately 1.4 times [Dir03]. As a consequence, shielding is an impractical soft-error mitigation solution in many computing installations where reliability is demanding, such as in embedded systems.

2.3.2 Alpha radiation

Another dominant source of soft-errors is considered to be alpha particle radiation [Wan07]. An alpha particle is composed of two protons and two neutrons. Alpha particles have a very high-energy as well as a large mass, and can be easily shielded by simple materials. Even a piece of paper is sufficient to shield alpha-particle radiation. Moreover, alpha particles can travel only a few centimetres in the air. Consequently, alpha particles should originate from a source very close to the circuit to be able to cause a soft-error.

The discovery of alpha particles to produce soft-errors goes back to the nineties when the Intel corporation experienced some random behavior in its 16-Kbyte DRAM memories caused by packaging [Bau05]. Intel then tracked the origins of suspected radioactive impurities, and they found that a new LSI ceramic package was used for these chips. The package used uranium materials and consequently the level of radiation emitted to the chips was higher than normal.

Nowadays, even very low alpha-particle rates can cause a malfunction in 45nm CMOS circuits and below. Packaging materials should therefore be selected carefully to reduce the amount of emission regarding alpha particles. Moreover, it turned out to be possible to shield the emission of alpha particles with shielding materials during packaging even if the technology was still less sensitive to alpha particles [Adv05].

Regarding the contribution of these particles to cause a soft-error, the neutron soft-error rate is the dominant one. However, shrinking technology dimensions along with reducing supply voltages has made the alpha particle the second dominant source of soft-errors [Adv05].

2.4 Soft-error vulnerability analysis

Despite the fact that detection and isolation of hard errors (permanent errors) in modern digital circuits are mature, it is very challenging to detect the occurrence of a failure caused by soft-errors in a system. A measure of vulnerability with regard to soft-errors should be available to evaluate circuits that are going to be used in a safety-critical environment. Soft-error sensitivity analysis has since long been used to assess the vulnerability of different parts of a design in the presence of different sources of soft-errors. The process of soft-error analysis is based on stressing the system under test with soft-errors.

Fault-injection has been used for many years as a method of soft-error analysis [Dav09]. Fault-injection works by injection of a predefined model of soft-errors in different parts of a design, for different applications. The fault-injection further determines the functional response of a circuit with regard to the injected soft-errors. Fault-injection is generally a very time-consuming and complex procedure since it requires to inject soft-errors in different logic states of a system (or at least the majority of states).

Fault-injection provides several advantages [Zia04]. To name a few, one can mention that the designer is able to understand the effects of soft-errors in a system under test. Moreover, if a protection mechanism is used in a system, fault-injection can be used to assess the efficiency of those mechanisms. Fault-injection can also be used to discover faulty behaviour of a system which is hidden during the normal tests. Finally, fault-injection needs to be carried out if a processor system is in operation. So it can be used to explore the behaviour of different benchmarks with regard to soft-errors.

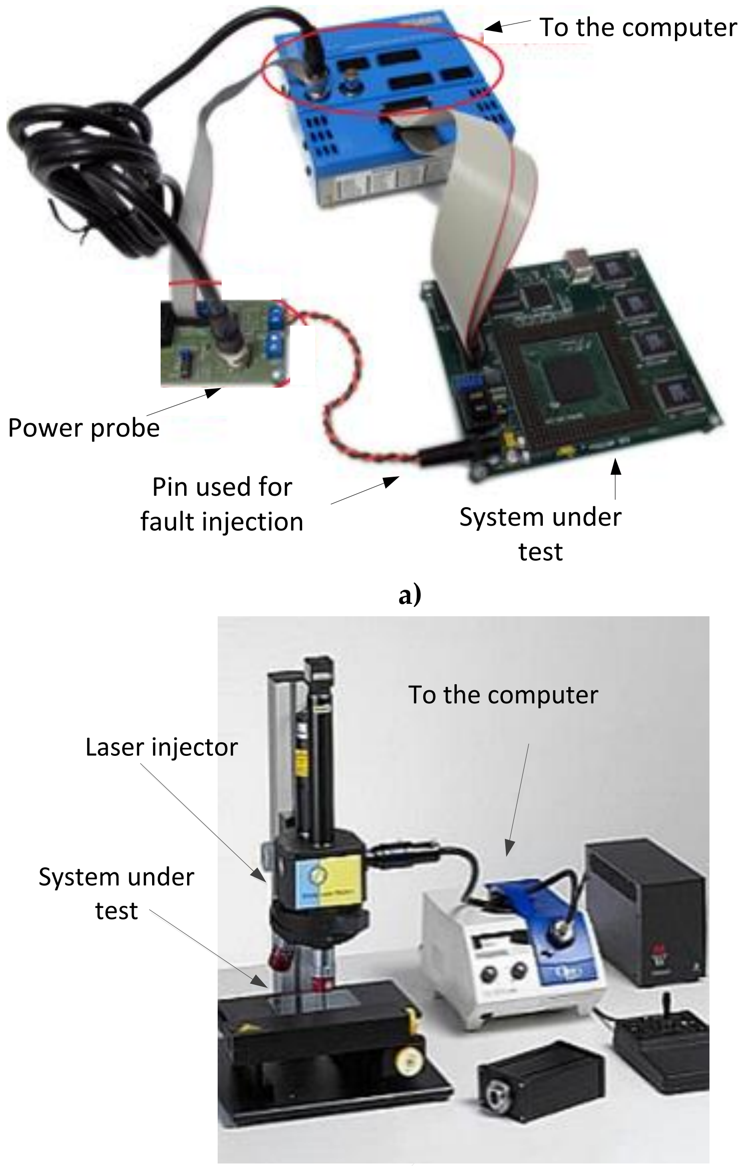
Fault-injection can be carried out at different levels of abstraction. In general, there are four categories of fault-injection, including Hardware-Based Fault-injection, Software-Based Fault-injection, Simulation-Based Fault-injection and Emulation-based Fault-injection. The following paragraphs will briefly explain the different categories. The main focus is to list the benefits and drawbacks of each method [Zia4, Zha07, Dav09].

2.4.1 Hardware-based fault-injection techniques

Hardware-Based Fault-Injection (HBFI) techniques are conducted by stressing the actual hardware with real environmental sources which are responsible for soft-errors. Those environmental parameters can be laser-based radiation [Pou00], power-supply disturbance [Hut09], and Electro-Migration Interference (EMI) [Var05]. HBFI techniques can be further categorized into [Zia04]:

HBFI techniques with contacts; in this category the fault injector is in direct physical contact with the system under test. The injector produces voltage or current changes externally to the target chip. Figure 2.5a shows a power-supply injector which is being used for fault-injection at the chip pins. This power supply (blue box) generates a disturbance and this disturbance will be consequently injected in the chip by a power probe.

In the case of HBFI without contact the injector has no direct physical contact with the system under test; an external source produces some physical activity, such as heavy-ion radiation to evoke a predefined disturbance of soft-errors in the circuit. Figure 2.5b shows a laser-based fault-injection which injects a very accurate laser beam into a system. The laser beam is used to modify the contents of a chip, while the white box provides the proper characteristics of the laser that is being injected to the chip. This method of fault-injection needs to be highly accurate in positioning especially with the current trend of shrinking chip technology dimensions.



a)

b)

Figure 2.5. a) Fault-injection at chip pins. b) Laser-based fault-injection (both pictures are a courtesy of [Opt12]).

Even though conducting hardware-based fault-injection techniques is very complex and costly, they are very close to the real physical nature of a soft-error. The benefits of hardware-based fault-injection can be summarized as [Zia04]:

The HBFI methods can access locations that cannot be accessed by other fault-injection methods. For example, laser-based fault-injection can inject faults into all the flip-flops (after removing any protective layers) and registers which are simply not accessible by I/O pins or software.

A physical analysis by injection of physical faults into a prototype is sometimes the only practical way to estimate the behaviour of a circuit with regard to soft-errors. This is the case if the source code of the system is not available or there is no simulation model of the predefined soft-error model to conduct fault-injection. Furthermore, there is no need to modify the architecture of the system under test to conduct fault-injection. This is desirable if the system is only available as a prototype.

Meanwhile there are different drawbacks for HBFI methods. Among them is limited observability, which means it is very hard to track an injected fault in the system. Moreover, HBFI techniques require special-purpose hardware in order to perform the fault-injection experiments.

In this thesis, the results of hardware-based fault-injection from others will be used to develop a simulation model for Single Event Transients (SETs) which can be incorporated in simulation-based fault-injection techniques.

2.4.2 Software-based fault-injection techniques

Traditionally, software-based fault-injection techniques modify the software being executed under the operating system. Different sorts of faults can be injected at this level, varying from register and memory faults to faulty network packets. Software fault-injections are more focused on the aspects of a system which are accessible by a software developer, for example the operating system. Software simulations are normally non-intrusive, i.e. the hardware of the system will not be changed. The benefits of software-based fault-injection techniques are that these techniques can be carried out on the basis of operating systems, which are difficult to conduct using hardware-based fault-injection approaches. Furthermore, experiments can be executed almost in real-time, depending on whether the timing of the system under test is a target of fault-injection or not. This allows running of a large number of fault-injection experiments within a reasonable amount of time. The same amount of time needs to be executed without a fault. Finally, software-based fault-injection techniques do not require any special hardware, and in addition conducting fault-injection experiments by software modification has a low complexity and hence a low development and implementation cost.

However, there are also a number of drawbacks; for example the fault-injection process needs to be executed at assembly language level. Therefore, the flexibility to model different soft-errors are limited. Furthermore, soft-errors cannot be injected into locations that are inaccessible by the software, such as an internal register file. Last but not least, it requires a modification of the source code to carry out fault-injection. As a result, the source code that will be executed for fault-injection will not be the same as the one that will run on the system under normal operational situations.

2.4.3 Simulation-based fault-injection techniques

Simulation-based fault-injections [Jen93] involves the construction of a simulation model of the system under analysis, including a detailed simulation model of the circuit which is being used for fault-injection. Moreover, the perturbation should be modelled at the same level as the circuit that has been modelled. The operational failure of the simulated system can occur according to a predetermined distribution of perturbations in order to accelerate the injection of soft-errors. This predetermination helps in terms of a more effective propagation of faults in the system, such as an overlap of an erroneous pulse with the positive clock edge of a flip-flop. First, the simulation model of the system under test is developed using a hardware description language such as VHDL or its American counterpart Verilog. Faults that have been modelled based on VHDL or Verilog are subsequently injected into the VHDL model of the system. The details of simulation-based fault-injection techniques will be explained in the next chapter. However, as the benefits and drawbacks of this class of fault-injection techniques the following comments can be made:

As a benefit, simulated-based fault-injection techniques can support almost all abstraction levels, from the transistor level up to the architectural level. The only requirement is that a simulation model of the system under test as well as the soft-error should exist at the same hierarchical level. In addition, it is possible to carry out this fault-injection method while the system is still under development. Another advantage is that there is full controllability over when and where a fault is injected into the system. This feature is very important in fault-injection analysis since the hardware-based fault-injection approaches cannot provide this degree of controllability.

Furthermore, the cost of computer infrastructure is low, in terms of special-purpose hardware. It also provides timely feedback to system design engineers because all the results of the simulation can be logged in the simulation computer for further investigation. In addition, during simulation-based fault-injection methods, a fault-injection is performed using the same software that will run in the field.

One of the most beneficiary features of simulation–based fault-injection methods is the degree of observability and controllability. In another words, any signal or register in the design can be accessed and modified. The result of this modification can be traced clock-by-clock in a simulation program.

As drawbacks of simulation-based fault-injection techniques, the following issues can be mentioned:

Fault-injection using simulation-based techniques needs a large development effort as the soft-errors should be modelled at the same hierarchical level as the system under test. Furthermore, conducting this type of fault-injection is very time consuming with regard to the experiment length; this is because carrying out simulation-based fault-injection is employing the simulation of the system in its fault-free version as well as in the presence of possible faults. This fact can cause the experimental length of these experiments to take several days while the simulation computer needs to run the fault-injection experiments.

2.4.4 Emulation-based fault-injection techniques

In recent years, a new category has been added to the fault-injection methods, known as emulation-based fault-injection techniques. This method injects faults in a circuit description implemented in an FPGA [Civ02, Por07]. This approach combines the efficiency of hardware-based fault-injection techniques and the flexibility of simulation-based fault-injection techniques in one framework. Experimental results have shown that a significant speed-up can be achieved as compared to simulation-based fault-injection techniques. However emulation-based fault-injections are generally only feasible for permanent faults, e.g. stuck-at faults. Moreover, the final circuit should be synthesizable and therefore the usage of test-benches in the fault-injection process is not possible.

The benefits of emulation-based fault-injection techniques are that the injection time is much shorter as compared to simulation-based techniques. This capability allows the designer to have a quick evaluation.

There are also drawbacks of this method, as the initial VHDL description must be synthesizable and optimized to avoid the requirement of a large and costly emulator; in addition a reduction of total running time can be accomplished. This fact limits the usage of test-benches in a circuit. Other disadvantages are that the implementation cost concerns the general hardware emulation system and the implementation of an FPGA-based emulation board. Furthermore, the algorithmic description of a circuit is not yet widely accepted by synthesis tools, and therefore emulation-based fault-injection approaches can often only be applied at the Register-Transfer-Level (RTL) of a system. Finally, it is necessary to have a high-speed communication link between the host computer and the emulation FPGA board which is a critical factor in the emulation set-up.

As a summary of different fault-injection methods, hardware-based methods provide the fastest fault-injection in terms of the required time to carry out experiments; however, conducting such experiments is very costly and complex to control. On the other hand, simulation-based fault-injections provide a high level of controllability to conduct perturbations; however, the required time to conduct such experiments is very long.

2.5 Architecture of our target processor

This section provides the baseline architecture of our case study, the Xentium processor®, from Recore Systems [Rec11]. As mentioned before, the goal of this thesis is to investigate the impact of soft-errors on digital processors. This includes the development of a model for soft-errors, assess the impact of soft-errors in a digital processors and also increasing the robustness of digital processors with regard to soft-errors. In order to assess these different criteria we have selected a Digital Signal Processor (DSP), the Xentium processor [Car11, Ker10] from Recore Systems [Rec11]. The Xentium processor is an ultra-low power DSP processor designed for high performance digital signal-based workloads.

The default architecture of the Xentium core including a data-path, a control unit, an instruction cache, a network interface and memory banks is shown in Figure 2.6. The memory banks are static RAMs that are communicating with the data-path in parallel to increase parallelism. A detailed architecture of the data-path is shown in Figure 2.7. The data-path has been designed based on a Very Large Instruction Word (VLIW) architecture that consists of ten functional units and five register files. Each functional unit is responsible for a certain class of instructions. For example, E units (E0 and E1) perform load/store instructions, M units (M0 and M1) are multipliers that are useful for accumulation operations. P and C units (P0 and C0) are used in those operations where the Program Counter (PC) is involved. Finally A (A0 and A1) and S units (S0 and S1) perform arithmetic and logical operations. All functional units can access five register files (RFA, RFB, RFC, RFD and RFE) in parallel. An actual implementation of the Xentium processor is based on 90nm CMOS technology leading to a silicon area of 1.2mm^2 and running on a clock frequency of 200MHz.

This processor has been developed as part of a multi-core System-on-Chip (SoC) system as depicted in Figure 2.8. This chip contains nine Xentium cores, interconnected by a NoC. Each of the single cores are able to connect to the adjacent routers, while the routers are connected to a Network-on-Chip (NoC). The NoC can be connected to more conventional bus architectures to communicate with other peripherals, if required.

Different parts of the Xentium processor will be elaborated in different chapters of this thesis. More details of each part of the processor will be discussed in the most appropriate chapter concerned.

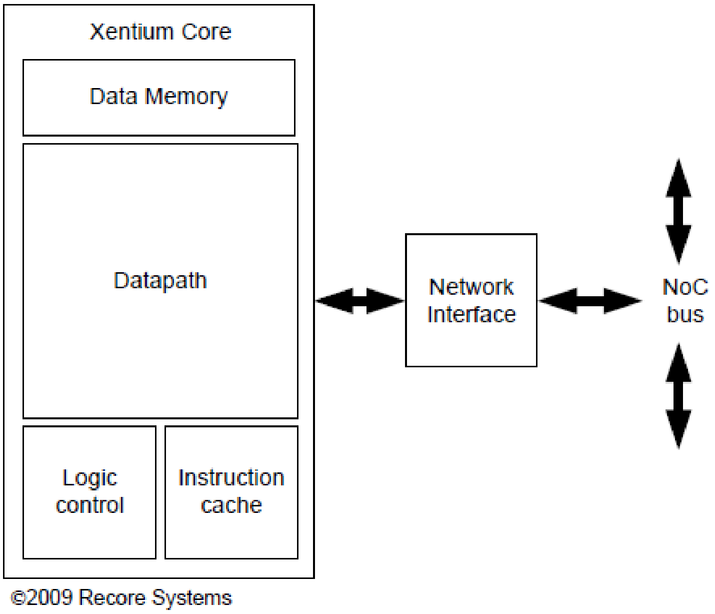


Figure 2.6. Xentium processor with memory and network interface [Rec11].

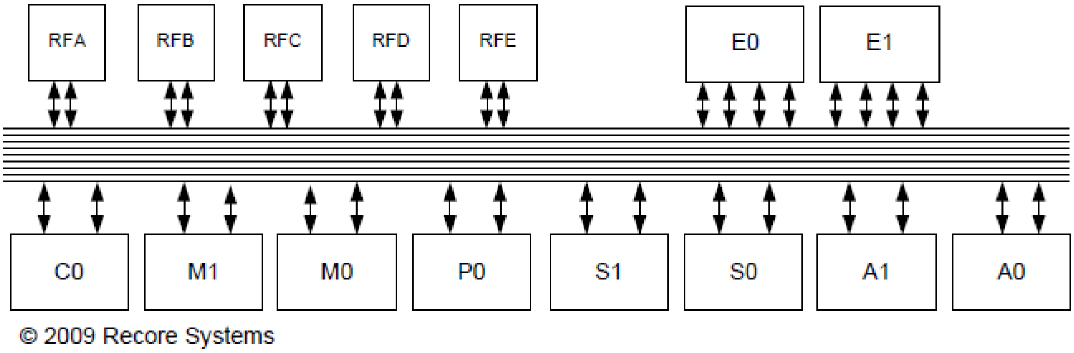


Figure 2.7. The Xentium data-path architecture [Rec11].

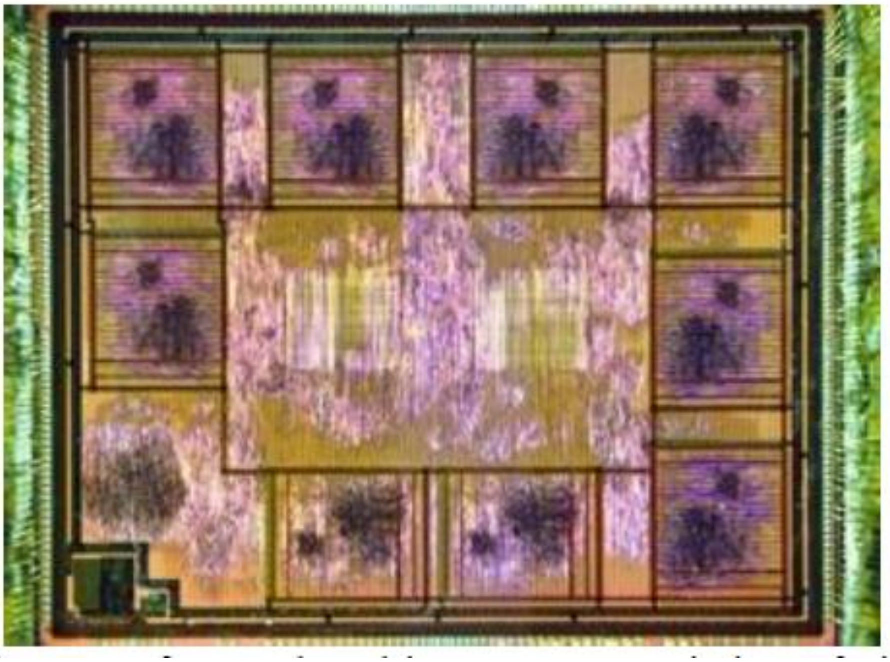


Figure 2.8 Photomicrograph of the multicore SoC consisting of nine Xentium core processors [Rec11].

2.6 Conclusions

This chapter provides the basic background with regard to soft-errors. The sources of soft-errors were discussed and also the terminology of soft-errors was provided. Different evaluation methods with regard to the effect of soft-errors in a digital system, including hardware, software, emulation and simulation–based fault-injections were covered in this chapter. Furthermore, the basic architecture of our case study has been introduced, the Xentium processor. The Xentium processor will be used later on in the evaluation of our proposed fault-injection method; also its architecture will be modified to develop a reliable DSP architecture to mitigate the effect of soft-errors.

References

[Adv05] S. Adve, P. Sanda, “Reliability aware microarchitecture,” in the IEEE/ACM International Symposium on Microarchitecture, Vol. 25, No. 6, pp. 8–9, 2005.

[Anc03] L. Anchordoqui, T. Paul, S. Reucroft et al. “Ultra-high energy cosmic rays: The state of the art before the auger observatory,” in International Journal of Modern Physics, Vol. 18, pp. 2229–2366, 2003.

[Bau02] R. Baumann, “Soft-errors in Commercial Semiconductor Technology: Overview and Scaling Trends,” in IEEE Reliability Physics Tutorial Notes, Reliability Fundamentals, pp. 1–14, 2002.

[Bau05] R. Baumann, “Radiation-induced soft-errors in advanced semiconductor technologies,” in IEEE Transactions on Device and Materials Reliability, Vol. 5, No. 3, pp. 305–316, 2005.

[Car11] J. Cardoso, M. Hubner, “Reconfigurable computing, from FPGAs to hardware/software co-design,” Springer, ISBN 978-1-4614-0061-5, 2011.

[Cis03] Cisco 12000 Single Event Upset Failures Overview and Work Around Summary, http://www.cisco.com/en/US/ts/fn/200/fn25994.html, 2003.

[Civ02] P. Civera, M. Macchiarula, “An FPGA-Based approach for speeding-up fault-injection campaigns on safety-critical circuit,” in Journal of Electronic Testing: theory and applications (JETTA), Vol. 18, No. 3, pp. 261-271, 2002.

[Cro99] A. Crouch, “Design-for-test for digital IC's and embedded core systems,” Prentice Hall, ISBN 978-0130848277, 1999.

[Dav09] J. M. Daveau, A. Blampey, G. Gasiot et al., “An industrial fault-injection platform for soft-error dependability analysis and hardening of complex system-on-a-chip,” in the Proceedings of IEEE International Reliability Physics Symposium (IRPS), pp. 212-220, 2009.

[Dir03] J. D. Dirk, M. E. Nelson, J. F. Ziegler, et al., “Terrestrial thermal neutrons,” in IEEE Transactions on Nuclear Science, Vol. 50, No. 6, pp. 2060–2064, 2003.

[For00] D. Lyons, “Sun Screen,” in Forbes Magazine, http://members.forbes.com/global/2000/1113/0323026a.html, 2000.

[Hut09] M. Hutter, J. M. Schmidt, T. Plos, “Contact-Based fault-injections and power analysis on RFID tags,” in European Conference on Circuit Theory and Design, pp. 409-412, 2009.

[Jen93] E. Jenn, M. Rimen, J. Ohlsson et al., “Design guidelines of a VHDL-Based simulation tool for the validation of fault tolerance,” in Proceedings of Open Workshop LAAS/CNRS, pp. 461-483, 1993.

[Ker10] H. G. Kerkhoff, X. Zhang, “Design of an infrastructural IP dependability manager for a dependable reconfigurable many-core processor,” in IEEE

International Symposium on Electronic Design, Test and Applications (DELTA), pp. 270-275, 2010.

[Nic11] M. Nicolaidis, "Soft-errors in modern electronic systems," Springer, ISBN 978-1-4419-6993-4, 2011.

[Ols93] J. Olsen, P. E. Becher, P. B. Fynbo, et al., "Neutron induced Single Event Upsets (SEUs) in Static RAMs observed at 10km flight altitude," in IEEE Transactions on Nuclear Science, Vol. 40, pp. 120-126, 1993.

[Opt12] www.opto.de, 2012.

[Pou00] V. Pouget, D. Lewis, P. Fouillat, "Time-resolved scanning of integrated circuits with a pulsed laser: application to transient fault-injection in an ADC," in IEEE Transactions on Instrumentation and Measurement, Vol. 53, No. 4, pp. 1227-1231, 2000.

[Pou07] M. Portela-Garcia, L. O. Celia, M. Garcia-Valderas et al., "A rapid fault-injection approach for measuring SEU sensitivity in complex processors," in IEEE International On-Line Testing Symposium, pp. 101-106, 2007.

[Rec11] Recore-systems, http://www.recoresystems.com/, 2011.

[Sha11] S. Z. Shazli, "High level modeling and mitigation of transient errors in nano-scale systems," PhD Thesis, ISBN 3443832, Northeastern University, 2011.

[Shi02] P. Shivakumar, M. Kistler, S. W. Keckler, et al., "Modelling the effect of technology trends on the soft-error rate of combinational logic," in the Proceedings of International Conference on Dependable Systems and Networks (DSN), pp. 1-10, 2002.

[Tab93] A. Taber and E. Normand, "Single Event Upset in avionics," in IEEE Transactions on Nuclear Science, Vol. 40, pp. 120-126,1993.

[Var05] F. Vargas, D. L. Cavalcante, E. Gatti, et al., "On the proposition of an EMI-Based fault-injection approach," in IEEE International On-Line Testing Symposium (IOLTS), pp. 207-208, 2005.

[Wan07] N. J. Wang, "Cost effective soft-error mitigation in microcontrollers," PhD Thesis, ISBN 978-1-4114-8598-5, University of Illinois at Urbana-Champaign, 2007.

[Yuh11] H. Yu, "Low-cost highly-efficient fault tolerant processor design for mitigating the reliability issues in nano-metric technologies," PhD Thesis, ISBN 978-1-1275-3245-1, TIMA Lab., 2011.

[Zha07] W. Zhang, X. Fu, T. Li, et al., "An analysis of microarchitecture vulnerability to soft-errors on simultaneous multithreaded architectures," in IEEE International Symposium on Performance Analysis of Systems and Software (PASS), pp. 169-178, 2007.

[Zia04] H. Ziade, R. Ayoubi and R. Velazco, "A survey on fault-injection techniques," in the International Arab Journal of Information Technology, Vol. 1, pp. 171-186, 2004.

[Zie81] J. F. Ziegler and W. A. Lanford, "The effect of sea level cosmic rays on electronic devices," in the Journal of Applied Physics, Vol. 52, pp. 4305–4312, 1981.

[Zie96] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld et al., "IBM experiments in soft fails in computer electronics," in IBM Journal, Vol. 40, pp. 3-18, 1994.

CHAPTER 3

A Framework for Accelerating Soft-Error Analysis in HDL Designs

Parts of this chapter have been published in the papers titled "A technique for accelerating injection of transient faults in complex SoCs" in the IEEE Euro-micro conference on digital system design in 2011, "Study of the effects of SET induced faults on sub-micron technologies" in the IEEE/IFIP international conference on dependable systems and networks in 2011 and "Rapid transient fault insertion in large digital systems" in the Elsevier journal of microprocessors and microsystems in 2013.

ABSTRACT - This chapter introduces two contributions in terms of simulation-based fault-injection in HDL designs. The first contribution concerns acceleration of soft-error injection in HDL designs, with regard to the elapsed CPU time, which is the real time to conduct fault-injections. The second contribution is dealing with conventional challenges in conducting simulation-based fault-injection, i.e. the importance of timing information in the net list on the accuracy of fault-injection results, as well as reaching the point of convergence in fault-injection results. The latter observation assures the designer that fault-injection results are not dependent on the number of fault-injections any longer. The introduced fault-injection framework is capable of simulating various fault models in a comparable elapsed CPU time, as compared to other conventional simulation-based fault-injection frameworks. The enhanced speed up has been assessed by conducting numerous simulation-based fault-injections on a DSP processor and comparing the elapsed CPU time to some conventional fault-injection tools. These experiments showed that the developed framework is capable of reducing the elapsed CPU time by a factor ranging from 27% to 67% as compared to conventional simulation-based fault-injection tools, and by a factor of 10% compared to available accelerated simulation-based frameworks.

3.1 Introduction

This chapter introduces a simulation framework to conduct simulation-based soft-error studies, as the first approach to deal with soft-errors.

As discussed in Chapter 2, simulation-based fault-injections are being used as a very detailed and accurate experimental method to assess the sensitivity of a system with regard to soft-errors, in the academic community as well as in the industrial world [Pec13]. Simulation-based fault-injection uses a simulation model of the system to evoke predefined fault models into different parts of a system. The simulation model of the system can be developed using any hardware description language, such as VHDL, Verilog or SystemC. The predefined fault models can also be described in any hardware language due to the availability of several integrated simulators which are able to simulate a design which is consisting of several types of HDL languages.

Simulation-based fault-injections provide various advantages which make them very popular for soft-error analysis [Bar05]. Issues are a high controllability over where and when a fault should be evoked, as well as a high observability in terms of the propagation of faults. Very important is the fact that the designer is able to conduct soft-error analysis even before the system is actually implemented. However, there are a number of downsides regarding some facets of simulation-based fault-injections. The first concern is that simulation-based fault-injections require an extensive period of Central-Processing-Unit (CPU) time of the host computer to conduct fault-injection experiments, or elapsed CPU time. This phenomenon is known as the CPU intensiveness [Zia04]. A long elapsed CPU time is induced by the fact that the simulation time is several orders of magnitude longer compared to the real time. Hence a comprehensive simulation-based fault-injection might take several days to be accomplished.

The second concern is due to the fact that the accuracy of fault-injection results strongly depends on the level of hierarchy in which simulation-based fault-injections are conducted [Nic11]. This means the results of fault-injections will lead to different results if fault-injection experiments are carried out on a front-end HDL model (Register Transfer Level, RTL) versus a back-end HDL model (such as post-synthesized logic gate-level net list, including timing information). This issue will become more important as a number of emerging soft-error standards, such as the Reliability Information Interchange Format, RIIF [Ava12], focus on the RTL hierarchy level; this provides a universal soft-error library regardless of the final library in which a circuit will be implemented. The results of this chapter will show that fault-injection results can be interpreted differently if the timing information in a net list (which is represented at the logic gate-level net list) is disregarded.

In this chapter, the CPU intensiveness of simulation-based fault-injections is addressed by developing a framework to speed-up injection of conventional models of soft-errors in a HDL design. Simulation-based fault analysis is composed of three different phases, set-up, fault-injection and evaluation phases. Our developed framework accelerates the whole simulation-based fault analysis by speeding up the fault-injection phase, while the set-up and evaluation phases are identical to other conventional fault-injection methods. It is also important to mention that the framework in this chapter has been developed to inject conventional models of soft-errors, i.e. the bit-flip model for Single-Event-Upsets (SEUs) and the momentary rectangular pulse for Single-Event-Transients (SETs) [Kar04], as discussed in Chapter 2.

Another subject of this chapter is the contribution on the level of granularity of the system under analysis in producing an accurate fault-injection. This issue will be addressed by conducting identical simulation-based fault analysis on a Digital Signal Processing (DSP) processor at two levels of hierarchies, a post placed-and-routed gate-level net list (including timing information), and a pre-placed-and-routed RTL net list. It will be shown that taking the timing information in a net list into account contributes to a faster convergence of fault-injection results. The latter issue is very important since reaching a point of convergence in simulation-based fault analysis is a metric which indicates that the fault-injection results are no longer dependent on the number of simulations.

The framework which will be presented in this chapter, will serve as a preliminary step in conducting soft-error evaluation studies. The outcome of this framework helps to distinguish the sensitivity of gates/nets of a system, with regard to soft-errors. Consequently, these sensitive parts will be enhanced with soft-error mitigation methods to decrease the level of soft-error vulnerability.

The remainder of this chapter is organized as follows: section 3.2 discusses state-of-the-art simulation-based fault analysis as well as the accelerated ones. Section 3.3 discusses the details of the developed framework. The achievements in terms of CPU intensiveness will be presented in Section 3.4 while the importance of hierarchical levels will be treated in Section 3.5. Finally, Section 3.6 will conclude this chapter.

3.2 Simulation-based fault analysis

The first step to conduct a simulation-based fault analysis is to represent the circuit under analysis in one of the HDL languages (VHDL, Verilog or SystemC). The next step involves perturbation of registers or nets according to a predetermined perturbation model, referred to as the fault model. This latter step is known as the fault-injection phase. An elementary simulation-based fault-injection experiment corresponds to one simulation execution during which one predefined fault model is injected into the simulation environment [Zia04]. A series of such simulations constitutes a simulation-based fault-injection campaign. A simulation-based fault-injection campaign might be composed of thousands of simulation-based fault-injection experiments. Finally, the logged results of the fault-injection campaign need to be interpreted to establish the sensitivity of the circuit under analysis or parts of it with regard to the injected fault model. This last step is formally known as the evaluation phase.

In order to discuss the development of our framework, first the state-of-the-art techniques that have been used in the fault-injection phase will be briefly presented. Then, the integration of two different approaches into one platform will be discussed in order to use their benefits in accomplishing an accelerated simulation-based fault-injection.

3.2.1 State-of-the-art simulation-based fault-injection

In general, implementing the fault-injection phase of a simulation-based fault analysis can be divided into two categories [Bar04, Zia04, Gra10]:

- using built-in commands of the simulator program, which approach is known as "built-in commands".
- using code-modifications techniques, which can be further divided into saboteur and mutant methods.

3.2.1.1 Built-in commands

Built-in commands are based on using, at simulation time, built-in commands of the HDL simulator in order to modify the value/timing of a net or register. This approach normally provides the fastest performance with regard to the total elapsed CPU time, since it does not modify any part of the representation of the circuit under analysis. However, the applicability of this technique strongly depends on the functionality of the built-in commands of the simulator program [Lee09]. For example, whether a momentary change in a value of a net is feasible or not depends on whether the force command has been embedded in a simulator kernel or not.

One of the most widely-used techniques in the built-in commands category is to disconnect a particular signal (target signal for fault-injection) from its input(s) at a certain point of time (so-called 'time instance'); then force it to a new value for a brief period of time (so-called 'fault duration').

For example, to inject a temporary pulse on a signal named 'V' at a time-instant 'T$_1$', the sequence of the following pseudo-commands should be executed [Gaw10] as shown in Figure 3.1.:

```
1-        simulate-until @ (T1) ns
2-        force-signal (V) to (faulty-value)
3-        simulate until @ ('fault duration') ns
4-        release-signal (V)
5-        simulation-continue
6-        result-logged
```

Figure 3.1. Pseudo-commands of built-in simulation fault-injection.

Several existing frameworks exploit the built-in commands for the fault-injection phase. MEFISTO [Jen94] was one of the first tools that employed built-in commands in the fault-injection phase. Another example is GSTF [Bar00] which is a VHDL-based fault-injection tool, was built based on the built-in commands of a commercial VHDL simulator (V-System®). It is capable of injecting soft-errors into different levels of hierarchy, which can be logic gate-, RTL- and chip-level.

With respect to the implementation cost, the built-in commands tools are the easiest and lightest simulation-based frameworks to set up, since no parts of the circuit under analysis need to be modified. Furthermore, no re-compilation (even partial) of any parts of the HDL code is necessary. Finally, built-in commands methods are known as the fastest methods with regard to the elapsed CPU time [Zia04, Gra10]. However, since these methods strongly depend on the functionality of the built-in commands of the simulator, the possibilities of representing various fault models are rather limited. For instance, they are not able to inject fault models representative for buses, such as intermittent-short or intermittent-delay faults [Zia04, Gra10].

3.2.1.2 Code-modification techniques

Another category of simulation-based fault-injections is the code-modification technique, where the HDL code of the circuit under analysis needs to be modified [Zia04]. Code-modification techniques are further categorized into saboteur and mutant techniques [Nic11].

In the saboteur technique, a component called saboteur is added to the HDL description of the original component for the sole purpose of fault-injection. The saboteur will be idle during normal system operation, while it modifies the value of one or more signals when it is active, i.e. at the moment a fault is being injected. For example suppose of a saboteur inserted at the input of an 'OR' gate. It is '0' when inactive (so no interference with the gate), but is '1' during activation. Saboteurs are inserted, in series or in parallel, either interactively at the schematic editor level or manually/automatically directly into the HDL source code. Serial insertion, in its simplest form, consists of breaking up the signal path between an input and its corresponding net and placing the saboteur in between. It is important to emphasize that the majority of fault models, including delay, stuck-at, SET and SEU can be implemented via the saboteur technique [Gil08]. However, saboteurs require a number of control signals to the original description of their target, such as gates, in order to indicate the type of the perturbation. Consequently, the additional control signals must be initialized for the top-level components, which consequently increases the complexity of the entire system [Ben03, Gil08]. A simple example of this increased complexity is shown in Figure 3.2. Here, a saboteur has been inserted in the Half-adder-0 (the grey gate). The control mechanism of this saboteur is that if its value is '1', the output of the gate 'OR' (the grey gate) will be '1', otherwise the saboteur will not interfere with the normal workflow of this half-adder. However, this control signal (indicated by dotted signal) needs to be initialized at all higher levels, including the full-adder, the Arithmetic-Logic-Unit (ALU) and finally in the microprocessor.

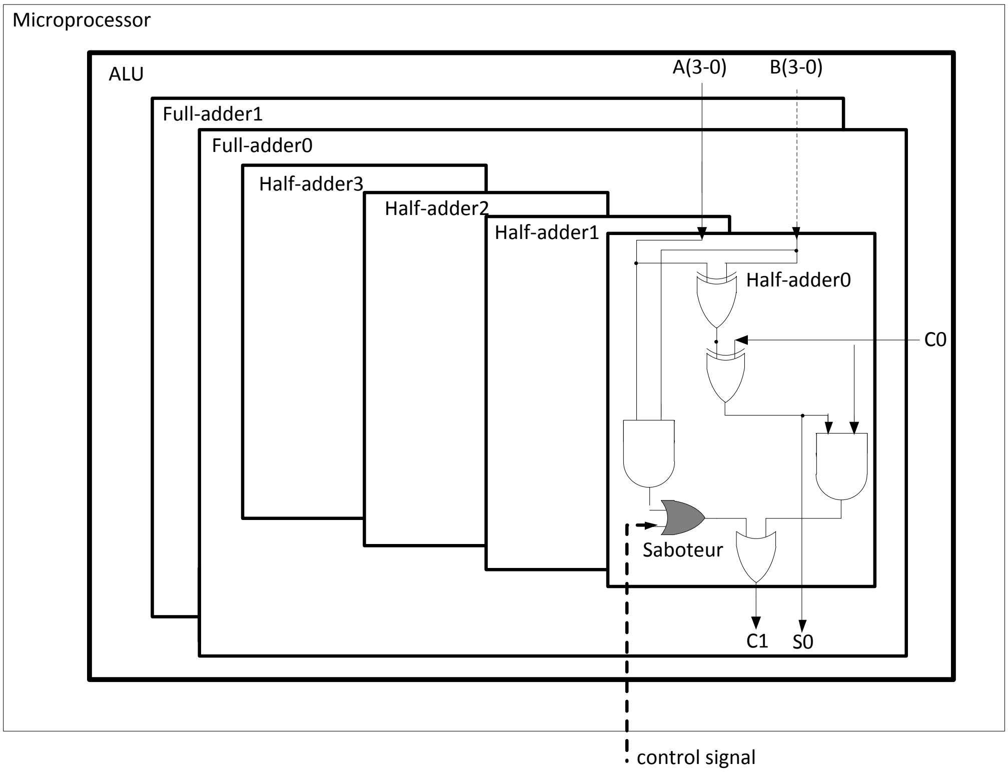


Figure 3.2. Interference of the control signal (dotted signal) of a saboteur with higher hierarchical levels.

A well-known framework which uses the saboteur technique is VFIT (VHDL-based Fault-Injection Tool) [Gil03]. One of the main features of VFIT is its automatic implementation of saboteurs.

Mutants techniques are another category of code-modification techniques which contain dormant code blocks within the normal net list description. These blocks of code are activated by injecting faults, altering the operation of the logic device, for example an 'AND' gate. Because the fault response is generated internally within the model, fault-injection can be carried out at any level of abstraction for various fault models. However, the usage of mutants requires the original model of a component by the new mutant model. With regard to the cost of the set-up phase, as well as the CPU intensiveness in the fault-injection phase, the mutant technique is the slowest simulation-based fault-analysis technique. Furthermore, a partial recompilation for every fault-injection might be required, which inherently results in an increased elapsed CPU time.

The main advantage of the mutant technique is its independence of the adopted simulator, which as a result provides the opportunity of representing very complex fault models. A well-known framework which employs the mutant technique for fault-injection can be found in [SUN11].

In this chapter, a simulation-based fault analysis approach will be developed which is based on the simultaneous usage of built-in commands (in order to accelerate the fault-injection phase) and the saboteurs (in order to enhance the possibility of representing various fault models). The main achievement of the developed framework is to accelerate the fault-injection phase while having the capability of representing a larger fault model repository, as compared to the built-in commands technique.

3.2.2 Accelerated simulation-based fault-injection framework

Since the main contribution of this chapter is to develop a simulation-based fault-injection framework which accelerates the injection of diverse models of soft-errors, this subsection briefly surveys recent accelerated simulation-based fault analysis approaches presented in the literature. It is important to mention that only simulation-based methods have been covered in this subsection, while other methods which use a combination of simulation-based methods and other categories of fault-injections, e.g. Field-Programmable-Gate-Arrays (FPGAs), emulation-based or software-based methods, are out of the scope of this section.

The authors in [Ber02] proposed two methods in order to speed-up the fault-injection phase in simulation-based fault analysis. The first method relies on setting checkpoints on simulator commands and the second method employs the well-known fault collapsing technique to shrink the list of equivalent faults. The first method, checkpoints on the simulator-commands, is based on restarting the simulator from the last known state for each fault-injection, rather than restarting from the beginning of simulations.

In the case of the second method, fault collapsing, a number of faults will be removed from the fault list based on the well-known fault collapsing techniques [Ben98]. Even though these two methods can decrease the elapsed CPU time of a simulation-based fault analysis, the check-point method is beneficiary only for rather simple designs. This because the more complex a design is, the more time is devoted to store/retrieve and manage the checkpoints. The authors in [Lee09] and [Mis07] convert a HDL design to a SystemC equivalent and subsequently employ capabilities of the C language, such as parallel computing, to speed up the simulations. Another approach transforms the soft-error analysis problem into the equivalent Boolean problem and subsequently exploits optimization techniques to speed-up fault analysis [Sha12]. A method which has been documented in [Lev05] relies on the generation of the mutants along with formal property checking. The technique is able to inject all models of soft-errors in a competitive elapsed CPU time. As examples of industrial platforms, one can mention [Reb00] and [Ber02], where acceleration of simulation time is achieved by re-arranging the simulator commands of the simulation program. De-rating factors of a net list, being masking factors of a gate, i.e. a '0' in an 'OR' gate and a '1' in an 'AND' gate, have been used in [Ale11] to minimize the number of fault-injections.

An accelerated fault-injection by engaging the RTL net list along with the logic gate-level net list is proposed in [Gar12]. A differential fault simulation approach, documented in [Ale12], has been developed based on conventional simulation tools and a novel parallel, soft-error optimized simulation tool. This method benefits from various optimization techniques targeting a smaller elapsed CPU time while preserving the accuracy of the results. As an example of engaging mathematical optimization methods in order to speed up the fault-injection phase, one can mention [Asa12], where an analytical soft-error reliability modelling technique has been employed in order to reduce the fault-injection time while achieving a higher accuracy.

The presented framework in this chapter has been developed based on simultaneous usage of built-in commands, provided by commercial HDL simulators, along with the saboteurs technique. The latter is used in order to accelerate the fault-injection phase while preserving the possibility of imitating different fault models. The simulation-commands will be used to directly modify the imposed control signals of each saboteur. As a result, it is not required to modify any component of a design, including the component which has been targeted for fault-injection as well as the top-level modules. This is a key factor in our approach.

In the next section, the details of our developed framework will be discussed.

3.3 The developed fault-injection framework

This section discusses the details of the developed fault-injection framework. The development phase will be explained in a bottom-up approach. First, the finest granularity modules, the Fault-Injection-Units (FIUs) will be explained, and subsequently the function of FIUs in the fault-injection process will be explained.

3.3.1 Fault-injection units

The developed fault-injection framework of this chapter is able to inject the majority of existing models of soft-errors in a HDL design. To be more specific, it can imitate all fault models which are supported by the saboteurs technique, including Single-Event-Transient (SET), Single-Event-Upset (SEU) and delay faults. Furthermore, this framework is able to support different levels of hierarchy in order to conduct soft-error analysis. We will explore different levels of hierarchy to show the capabilities of our framework as compared to conventional fault-injection tools.

A basic element in our framework is the introduction of Fault-Injection-Units (FIU). FIUs are units which are added to the target components (which can be a net or register, in a hardware description language) in order to inject a fault of the intended fault model. FIUs work almost the same as saboteurs, i.e. they are inactive during the normal system operation. They alter the value or timing characteristics of a net or register during fault-injection when they are active. However, the control mechanism of FIUs is completely different as compared to the saboteurs. If a FIU is attached to a net or register, it is not required to change any hierarchical levels of a design. The insertion of FIUs can be done either manually (for a small number of FIUs) or automatically (for a very large number of FIUs) in a system. As the main goal of this chapter is to assess the degree of acceleration achieved by the proposed framework in the fault-injection phase, FIUs are inserted manually into the net list of our design. However, all FIUs can be automatically inserted using the well-known automatic saboteur insertion techniques [Gri11].

In order to control a FIU, a Fault-Injector-Signal (FIS) is incorporated into each FIU; it is directly controlled by built-in commands. Whenever the FIS takes value '1', its corresponding FIU will inject the associated fault model into the target component, which can be a net or register. However, a FIU will not interfere in the normal operation of the component if its corresponding FIS holds the value '0'.

All FIUs have to be inserted into the HDL net list prior to the compilation process. Characteristics of each FIU in terms of 'time instance' (i.e. time of occurrence) and 'fault duration' (i.e. the duration that a fault manifests itself in the system) are determined at the instantiation time of each FIU, during the set-up phase. Therefore, it is not required to recompile any part of the design during the fault-injection phase.

In the following paragraphs, different models of FIUs will be introduced and discussed. The first FIU model has been developed to inject a SET in combinational-logic blocks. The second FIU model injects a SEU in sequential-logic blocks, and the last model injects a delay fault in a net in the critical path. Figures 3.3a and 3.3b show the first FIU model which injects a SET into a target net indicated by 'V'. As can be seen in these figures, if the FIS signal is '0', the 'V' signal will be derived from its normal signal value (input). However, if the FIS signal is '1', a glitch will be injected in the 'V' signal. The characteristics of the perturbation in the 'V' signal, which are 'time instance' and 'fault duration', are completely bounded to the FIS signal. It is important to mention that FIS signals are directly controlled by the built-in commands of the simulator program; hence no parts of the top-level components need to be modified. Figure 3.3a forces a positive transition (0-to-1) in the 'V' signal while Figure 3.3b shows a FIU injected as a negative transition (1-to-0) in the 'V' signal.

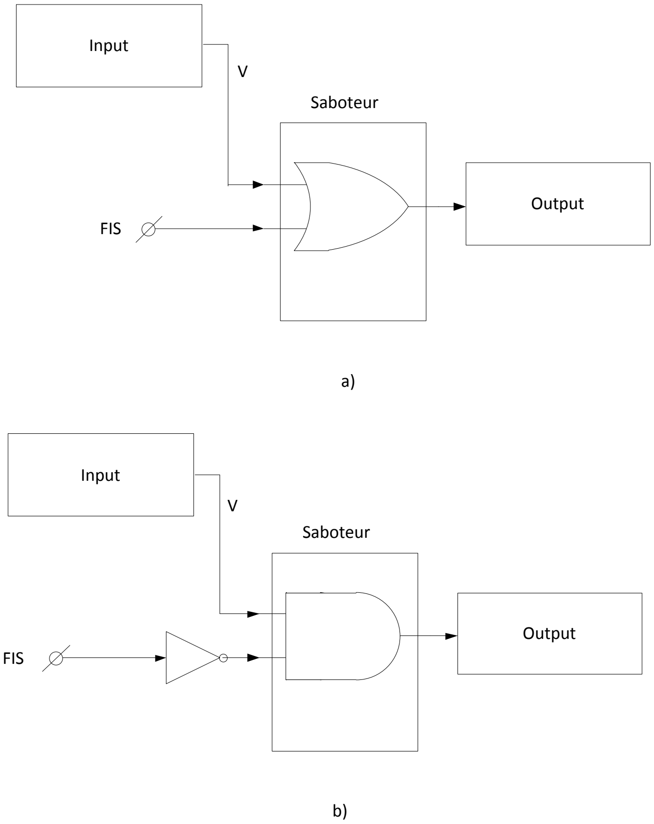


Figure 3.3. FIU insertion for a SET. a) positive transition (0-to-1). b) negative transition (1-to-0).

Figures 3.4a and 3.4b show two FIUs models in order to inject delay and SEU fault models in a nominal component, respectively. The delay fault model is represented by a buffer in Figure 3.4a. In this figure, when the FIS signal is ‘1’, a delayed value of the ‘V’ signal (via the buffer) will be passed to the output which is consistent with the delay-fault model. A FIU model to inject a SEU is shown in Figure 3.4b. If the signal FIS is ‘1’ in this figure, the inverted value of the input will be passed to the output, otherwise the normal input will go to the output gate. As can be seen in these figures, if the signal FIS is ‘0’, the ‘V’ signals in all FIUs will take their fault-free value. However, if the FIS signal is ‘1’, the V signal will get an erroneous value.

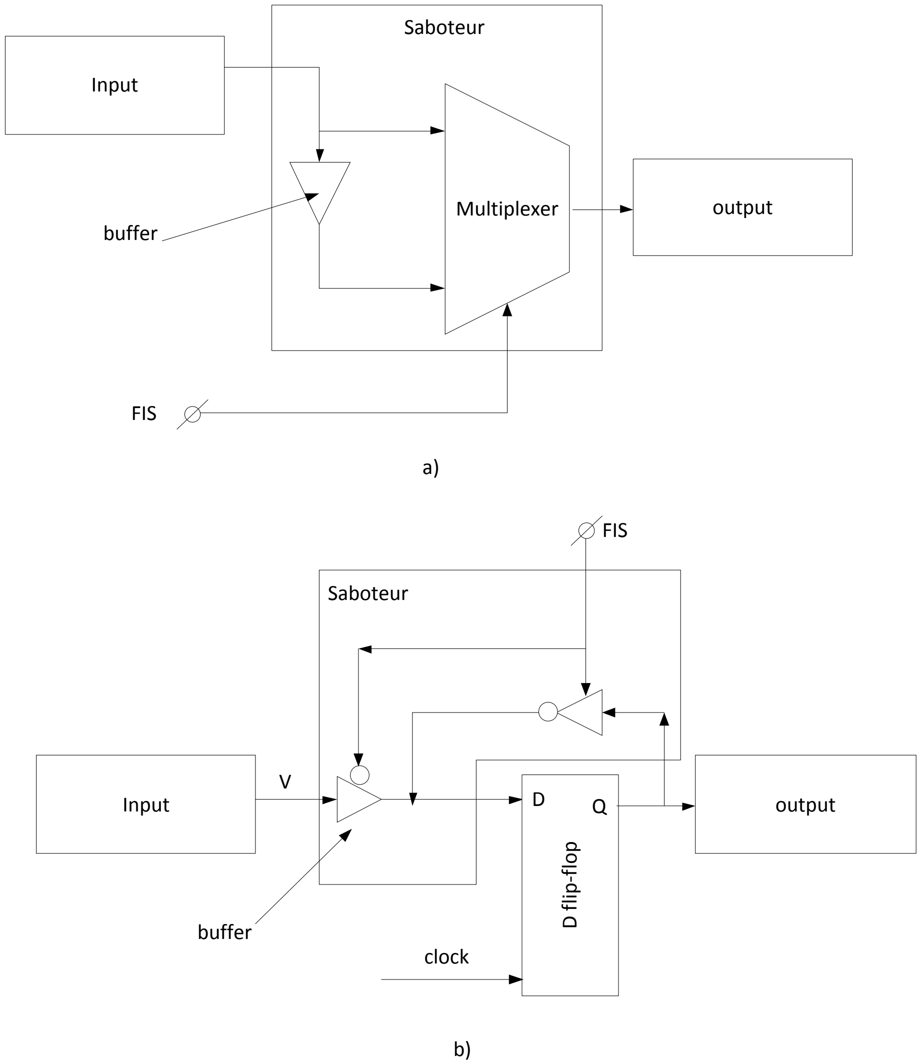


Figure 3.4. FIU insertion for delay and SEU faults. a) FIU insertion for a delay fault and b) the SEU fault model.

One of the limitations of FIUs is that all key characteristics of the injected perturbation, such as 'time instance' and 'fault duration' are directly related to the FIS signal. In the conventional saboteur technique, all the calculations to perform an effective fault-injection (such as overlapping the duration of a perturbation with the positive clock edge) are dynamically carried out by observing the component of interest by saboteurs during run-time operation [Arl03]. This dynamic assignment of fault durations during the run time makes saboteurs capable of injecting an effective fault. However, in our framework, all characteristics of a perturbation have been embedded in the FIS signals which need to be defined before run-time of a system. It will be shown that selection of a specific mathematical distribution function for the duration of the signal FIS with regard to the system clock period (T_{clk}) causes most of the changes of FIS signals cover a positive clock edge of the system clock; which means that a change in the FIS signal will be captured by the system clock and hence will be propagated into the higher levels of hierarchy. As a result, all FIS signals can be statically scheduled at the beginning of the fault-injection phase. This issue will be elaborated later on in this chapter.

3.3.2 Embedding FIUs in the fault-injection phase

The developed framework of this chapter is an integrated fault-injection and evaluation framework. The main achievement of this framework is to provide a rapid soft-error injection in a HDL-based system. After the insertion of FIUs into the predefined nets/registers of a system (either manually or automatically), the user needs to specify some characteristics of the system under analysis. More specific the system clock period (T_{clk}) and the overall execution time of the workload ($T_{execution}$) for the circuit under analysis. Several mathematical distribution functions have been investigated in order to provide the possibility of statically determine fault durations with regard to T_{clk}. The detailed explanation of these assessments will be given in the section 3.4. As a result of those experiments, it has been established that if durations of FIS signals form an exponential distribution function with an arithmetic mean value μ which equals to T_{clk}, about 90% of the perturbations will overlap with the positive clock edge of the system. In this case the internal state of the circuit under analysis will be potentially affected.

To determine the exponential distribution function for fault durations, any mathematical tool can be used. In our framework, a MATLAB routine determines fault durations of each FIS signal such that all the different fault durations compose an exponential distribution function with the mean value μ equal to the system clock period. As a result, each FIS signal has a different sample with the same time instance but different fault durations. This concept is shown in Figure 3.5. Here, four different examples of a glitch signal are shown. All the samples have started from the same time instance (6ns) but the duration of each sample (fault duration) is different. In this example, the last two perturbations have an overlap with the positive clock edge of the system clock and so are able to propagate into the system.

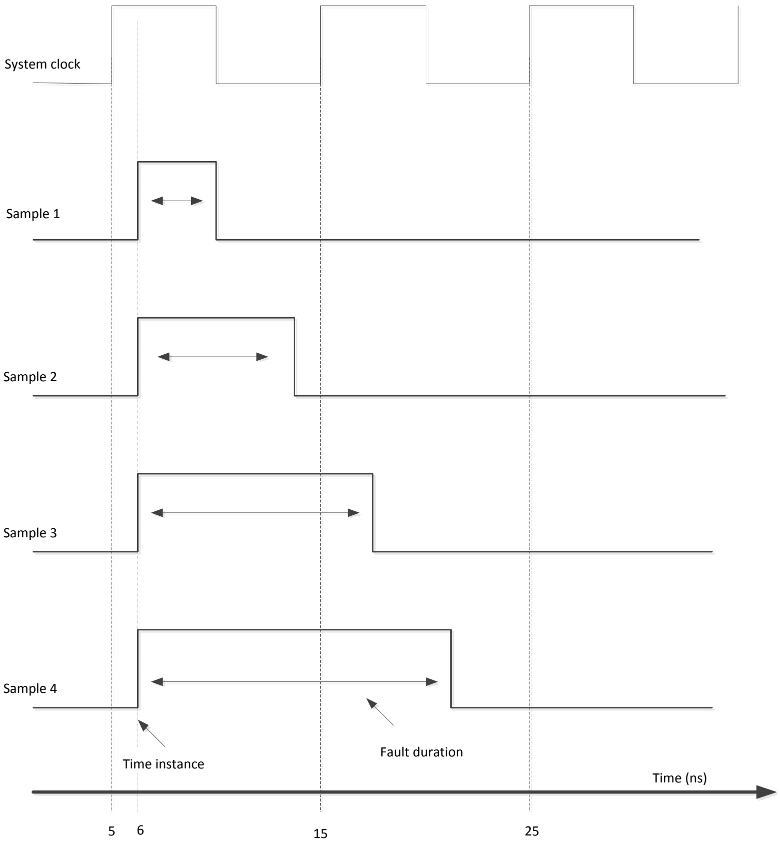


Figure 3.5. Different fault durations for a perturbation.

Finally, the framework combines all the prerequisite information, including the number of samples for each FIS (which will be determined by the user), time instance (which is a random time) and fault duration (which will be determined by a MATLAB routine). To be able to automate the execution of fault-injection, all the mentioned information is stored in a macro file. This macro file is directly executable by a commercial simulator program, in our case QuestaSim® from Mentor Graphics [Men10]. Moreover, our framework observes the logged result of the simulator to see whether a fault was able to affect the circuit under analysis or not, and the proportion of affected faults.

Figure 3.6 shows an overview of our proposed framework. As can be seen in this figure, it is composed out of three main phases: set-up phase, fault-injection phase and evaluation phase. During the set-up phase, the user incorporates FIUs into the HDL representation of the system. As mentioned before, this step can be carried out automatically [Gri11]. The FIU incorporation is a partial code-modification that imports FIUs into the predefined nets/registers inside the HDL net list. The predefined nets/registers are selected randomly. The user also needs to define the specifications of the system under test for the framework. These specifications include the system clock period T_{clk}, the total execution time $T_{execution}$ of the workload, and the total number $N_{fault\text{-}injections}$ of fault-injection experiments. This information will be used by different parts of the framework in order to define a distribution of fault durations for each FIS. As mentioned before, a software routine developed in MATLAB has been used to specify the distribution of fault durations, in such a way that the mean value μ of all the samples of each FIS will be equal to the T_{clk}. This is because the maximum overlap between a perturbation and a positive clock edge of the system can be achieved.

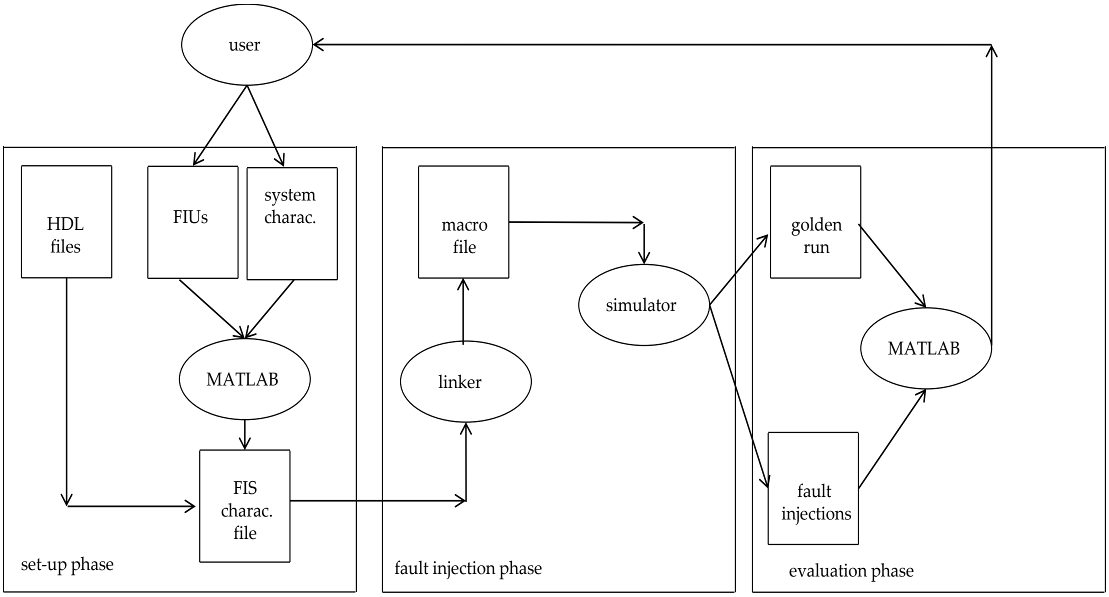


Figure 3.6. Overview of our fault-injection framework.

Each fault-injection is carried out during one execution of a workload. The time when the processor starts executing a workload is known as the reset time T_{reset}, while the time when the processor finishes the workload is known as the execution time $T_{execution}$. The time instance of each FIS is completely random in time, with the following marginal constraint. Its minimum limit equals to ten clock-cycles after the reset time, $T_{reset} + 10*T_{clk}$, while its maximum limit is equal to ten clock-cycles before the end of the workload execution-time, $T_{execution} - 10*T_{clk}$. These boundaries can be represented as:

$$T_{reset} + 10 * T_{clk} < Fault\ duration < T_{execution} - 10 * T_{clk} \qquad (3.1)$$

These constraints have been selected in this way to allow the injected faults to propagate during the active execution time of the system [Wan11]. Hence, there is potentially sufficient time for an injected fault to alter the status of the system.

The fault duration of each FIS signal is calculated based on a specific distribution function, which is an exponential distribution function in our case. The ‘time instance’ and ‘fault duration’ of each FIS is stored into a text file named FIS characteristics file (as is shown in Figure 3.6).

In the fault-injection phase, a software program named ‘linker’, which has been implemented by a C++ program, combines the characteristics of each FIS signal, i.e. ‘time instance’ and ‘fault duration’ in order to generate the macro file. The macro file can be read by a commercial simulator, which is QuestaSim [Men10] in our framework. Figure 3.7 shows the pseudo-code for the macro file.

```
Do experiment j
  {
          Restart the design
          Simulation top-module
          Run @’time instance’
          Force FIS(j) to faulty-value
          Run @’fault duration’
          Force FIS(j) to fault-free value
}
```

Figure 3.7. Pseudo-code of the generated macro file.

The key information within this macro file include:

- top module: the module that must be simulated during the fault-injection phase.
- time instance: the ‘time instance’ that the specific FIS signal FIS(j) will be activated.
- force: direct access to each FIS via built-in commands of the simulator program. This direct access is the key part to accelerate the fault-injection phase.
- fault duration: the duration of each FIS signal, FIS(j), that has been calculated by a MATLAB program during the set-up phase and has already been saved in the FIS characteristic file.

- Number of Experiments: the total number of fault-injection experiments that has been determined by the user during the set-up phase. This number is started from an initial value (which is random) and will be increased until a convergence point is determined in the fault-injection results. The determination of the convergence point is discussed in detail in the next section. After execution of the macro file, the simulator program logs the simulation data. The data include fault-free results (golden run) and the fault-injection results, obtained from the fault-injection phase.

Constructing the macro file is a key issue in our approach, since it integrates all the information which has been produced by the MATLAB routine along with the specific requirements of each FIU in order to provide built-in commands a direct access to the FIS signals. When the macro file is applied to the simulator program, all fault-injections will be automatically carried out and the output information will be subsequently logged into a text file in order to be interpreted during the evaluation phase. This latter phase is accomplished by tracing differences between the golden-run and the data gathered during the fault-injection phase. The vulnerability of each net/register will be calculated by considering the proportion of faulty outputs over the total number of fault-injections in that net/register.

In Section 3.4, the developed framework will be employed to carry out fault-injections in two processors, an AVR microprocessor and a DSP Xentium processor. Fault-injection experiments on the AVR microprocessor have been used to assess the degree of speed-up achieved by the proposed framework. The results of fault-injection in the Xentium processor has been used to address some other facets of simulation-based fault-injections. We mention the importance of hierarchy levels of simulation-based fault-injection and the final number of fault-injections (number of experiments) in a fault-injection process. The architecture of the Xentium processor has already been explained in Chapter 2. All the following data have been achieved by applying the developed framework to the original architecture of the mentioned processors.

3.4 Time acceleration results

The main contribution of the developed framework is to integrate the possibilities of built-in commands techniques along with the saboteur technique to speed-up simulation-based fault-injections. Furthermore, it is very important to have the ability of modelling various fault models in the developed framework. As the dominant fault models for soft-errors are SET and SEU [Wan11], there is a particular interest in emulating these two fault models in our framework. A SET fault is modelled by the logic depicted in Figure 3.3a and a SEU will be modelled by the logic as shown in Figure 3.4b.

Table 3.1 summarizes possible fault models for different simulation-based fault-injection techniques. Since the representation of fault models in our framework is based on the saboteur technique, the capability of modelling faults for the saboteur technique and our developed methods is identical. However, fault-injection experiments will show that the elapsed CPU time in our developed framework is much lower (up to 67%) as compared to mutant techniques. It is important to mention that even though the mutant technique has the best capability to model different faults, conducting fault-injection experiments with this technique requires a considerable amount of elapsed CPU time. The mutant technique is known as the slowest simulation-based fault-injection method.

Table 3.1. Possible fault models in different simulation-based fault-injection methods.

Injection technique	Possible fault models
Built-in commands	SET, delay fault
Mutant	SET, SEU, delay fault, MEU [1]
Saboteur	SET, SEU, delay fault
Check-point on simulator [Rod02]	SET, SEU, delay fault
Fault collapsing [Rod02]	SET
Our developed framework	SET, SEU, delay fault

[1] Multiple Event Upset

To assess the degree of acceleration gained by the developed framework, our fault-injection results on an AVR microprocessor have been used. During the fault-injection phase, the AVR microprocessor executes two benchmarks from the Mibench benchmark suite [Gut01]: a bit-count and a quick-sort program, both from the automotive and industrial control domain of the Mibench benchmark repository. The bit-count program tests the bit-manipulation abilities of a processor by counting the number of bits that are set to '1' in an array of integers. The quick-sort program sorts one hundred integer numbers. The motivation to use these two benchmarks is to provide a fair comparison between different fault-injection frameworks of the AVR microprocessor; this because other simulation-based methods also used these two benchmarks to conduct fault-injection results with respect to the AVR microprocessor.

The framework described in the Section 3.3 has been used to inject SEUs on sequential components (registers and flip-flips) of an AVR microprocessor (flip-flops, SRAM) as well as SET faults into its combinational components (nets inside the combinatorial logic). Furthermore, the challenge of static determination of fault durations is addressed in the following.

As mentioned before, saboteur and mutant techniques can gather the timing information of a net/register during run-time and then inject a fault at a positive clock edge. However, this is not feasible in our framework since all the information about faults need to be stored in a macro file before starting a fault-injection experiment. In other words, all fault characteristics should be statically scheduled prior to a fault-injection experiment.

Two essential parameters for each injected fault are 'time instance' and 'fault duration'. While 'time instances' are assigned randomly, 'fault durations' need to be pre-calculated to cause SETs to appear during a positive clock edge. In order to statically schedule fault durations, several distribution functions have been evaluated to assess their degree of effectiveness with regard to fault-injection. As a rule of thumb, the more fault-injections occur during the positive edge of the system clock, the more it propagates faults into a system. Therefore the goal is to define a distribution function of the fault duration which can cause maximum overlap between fault durations and a positive edge of the system clock (a so-called effective fault). Figure 3.8 shows the concept of effective and ineffective faults. An effective faults occurs during the positive edge of a system clock and has a potential to be captured by a flip-flop or register; while an ineffective fault occurs outside of the time window of being captured by sequential logic. Suppose that a distribution function 'Distribution_a' causes 10 effective faults out of 100 samples of a SET while distribution function 'Distribution_b' causes 50 effective faults out of 100 samples, then distribution function 'Distribution_b' is a better candidate to define the duration of fault signals.

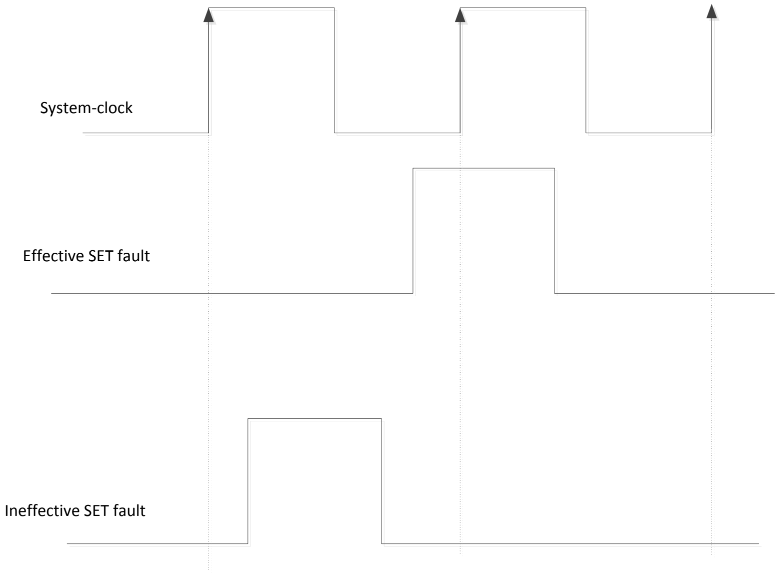


Figure 3.8. Effective and ineffective SET faults.

Table 3.2 shows three distribution functions which have been investigated to define the duration of FIS signals in our experiments. This table shows three different distribution functions, which are 'Normal', 'Poisson' and 'Exponential' distribution functions. The second column of this table shows three different mean values μ which have been considered to calculate duration of faults. The third column of this table is the percentage of injected faults which overlap with the positive edge of the system clock.

As can be seen in Table 3.2, in the case of an exponential distribution with mean value μ equals to the system clock period T_{clk}, 90% of the injected FIS signals will overlap with a positive edge of the clock. This result implies that by defining an exponential distribution with a mean value equal to the system clock period T_{clk}, is a feasible solution to assign the duration of FIS signals at the beginning of the fault-injection experiments in the set-up phase.

Table 3.2. Fault duration classifications for the AVR microprocessor.

Distribution function	Mean value μ	Percentage of effected faults (%)
Normal	T_{clk}	54
	$0.5*T_{clk}$	62
	$2*T_{clk}$	38
Poisson	T_{clk}	40
	$0.5*T_{clk}$	68
	$2*T_{clk}$	53
Exponential	T_{clk}	90
	$0.5*T_{clk}$	76
	$2*T_{clk}$	78

To conduct fault-injection experiments in the case of the AVR microprocessor, 100 FIUs were inserted into different random parts of the data path of the AVR and then 200 samples were applied to each FIS signal, resulting in 20,000 fault-injections in total. The duration of the samples of each FIS were assigned by an exponential distribution with a mean value μ equal to the system clock period. The 'time instances' of all FIS signals have been selected randomly, with the boundary conditions of Equation (3.1). The density of the injected FIS signals in the net/registers of a component is proportional with the occupied silicon area of that component. For instance if the Program-Counter (PC) occupies two times more area as compared to the Accumulator (AC), the number of FIS signals on the PC should be two times larger as compared to the AC.

The elapsed CPU time (real time) for our developed framework as well as that of some conventional simulation-based methods have been summarized in Table 3.3. The numbers in this table have been achieved by implementing all three methods, including saboteur, mutant and our developed method with identical parameters with regard to the AVR microprocessor. The last column of this table shows the achieved improvement for our framework as compared to saboteur and mutant techniques. The 'reference point' in this table means that the elapsed CPU time for the saboteur and mutant techniques has been compared to the elapsed CPU time of our proposed framework. It shows that using our framework instead of the saboteur technique leads to 23% reduction in elapsed CPU time while using our framework instead of the mutant technique leads to 66% reduction.

Table 3.3. Elapsed CPU time for conventional simulation-based methods.

Fault-injection technique	**Elapsed CPU time (hours)**	**Elapsed CPU time improvements if our presented framework was used (%)**
Saboteur [Nic11]	5.2	23%
Mutant [Nic11]	12	66%
Our presented framework	4	Reference point

As mentioned before, all experiments listed in Table 3.3 have been carried out with similar parameters, in terms of the total number of fault-injections (20,000), fault models, the circuit under analysis (AVR microprocessor) and the infrastructure which has been used to carry out simulations. Table 3.3 shows that the elapsed CPU time decreases from 23% (as compared to the saboteur technique) up to 66% (as compared to the mutant technique). It is noted that part of the longer elapsed CPU time for the mutant technique is caused by the fact that the circuit under analysis needs to be partially recompiled if a different FIS signal is selected for fault-injection. This means 200 partial recompilations are carried out for this campaign. This recompilation is not required for the saboteur technique as well as for our developed technique. In addition, the speed-up of our method as compared to the saboteur technique is based on the static schedule of the control signals at the beginning of the fault-injection experiments, during the setup phase.

A comparison between the required CPU time of our proposed framework over some accelerated methods have been given in Table 3.4. The numbers shown in Table 3.4 have been extracted from [Ber02]. It is also important to mention that the numbers of Table 3.4 are extracted directly from the associated literature, so the target system is not identical for all these three methods. Even though the improvement of the check-pointing method is better than ours, the check-point method is only beneficiary if the size of the execution time of the workload is time-limited, meaning only a small number of checkpoints is required. Otherwise, storing and retrieving checkpoints impose a high overhead on CPU time, which will cause a significant increase in elapsed CPU time.

Table 3.4. Elapsed CPU time improvement using accelerated techniques.

Accelerated technique	CPU time improvement (%)
Check-pointing [Ber02]	43.9
Fault collapsing [Ber02]	15
Our presented method	23

3.5 Level of hierarchy versus results of simulation-based fault-injections

This section addresses two issues concerning simulation-based fault-injections. The first issue is addressing the impact of timing information of the net list in fault-injection experiments. Our goal here is to determine the difference in fault-injection results if the experiments are carried out on a pre-synthesized HDL net list versus a detailed post-synthesized timing net list of a processor.

The second issue is the impact of the number of fault-injection experiments on the convergence of fault-injection results. The motivation to address the latter is that a different number of fault-injections might lead to a different sensitivity level for a system [Tou07]. Our goal is to find the point in the number of fault-injection experiments where the behaviour of the processor to injected faults is not dependent on the number of fault-injections any longer.

In this section, the above-mentioned questions are answered by the assessment of the effect of SETs on a modern high performance DSP processor, the Xentium processor from Recore-Systems [Rec11]. The Xentium processor has been used as a case study here; however, the mechanism employed here can be used in any other processor to achieve a convergent point in fault-injection results. The detailed architecture of the Xentium processor has already been described in Chapter 2. Two sets of fault-injection campaigns have been carried out by using two models of the Xentium processor: the first model is a pre placed-and-routed VHDL net list (RTL net list, without timing information) and the second model is a post placed-and-routed Verilog net list (gate-level net list including estimated timing information). The number of fault-injection experiments has been increased from an initial value. The initial value should be fairly small; i.e. it would be possible to increase that number tens of times and still conducting fault-injection experiments for this increased number would be feasible. This initial value depends on the workload execution-time as well. For example if a workload takes 10s (simulation-time) to be executed, 500 fault-injections need 5,000 seconds (about 1 hour and a half). The number of fault-injections will be increased from that initial value until a point of convergence in fault-injection results is established.

These simulation experiments only investigate the effect of SETs on the data path of the Xentium processor. The framework described in Section 3.3 has been used to carry out SET injections on two models of the Xentium processor; a timed net list (hereafter referred to as gate-level net list) and a behavioural net list (hereafter referred to as RTL).

The types of impact of each injected fault on the processor can be classified as Silent Data Corruption (SDC) or Detected-Unrecoverable-Error (DUE) [Nic11]. Since the original design of the Xentium processor does not have an error indicator signal, which indicates detection of an error by the Xentium processor, the experimenter cannot observe DUEs. As a result, the functional response of the processor with respect to each injected fault has been classified into one of the following categories:

- Silent Data Corruption, SDC: this condition is met if the error propagates through the circuit without awareness of its occurrence by the system. So the processor will provide an output without any error flag while its output is not correct.
- Time out: is the possibility that the processor unexpectedly stops its application, before execution of the whole workload. The outputs of the processor provide no meaningful output data in this case.
- Correct behaviour: the processor completes the application.

The results of the processor have been represented by a percentage (%), e.g. if 10 out of 100 fault-injections produce a Silent Data Corruption (SDC) failure, the SDC failure-sensitivity of the processor will be represented as 10%.

The fault-injections have been carried out on all ten functional-units of the data path of the Xentium processor. In order to get an overall view of the data path behaviour, one should consider that the data path of the Xentium consists of six functional-units (A, E, M, S, C, P), so the total sensitivity rate of the data path (P_{total}) can be calculated as:

$$P_{total} = (A_E/A_{total})*P_E + (A_S/A_{total})*P_S + (A_A/A_{total})*P_A + (A_C/A_{total})*P_C + (A_P/A_{total})*P_P + (A_M/A_{total})*P_M \quad (3.2)$$

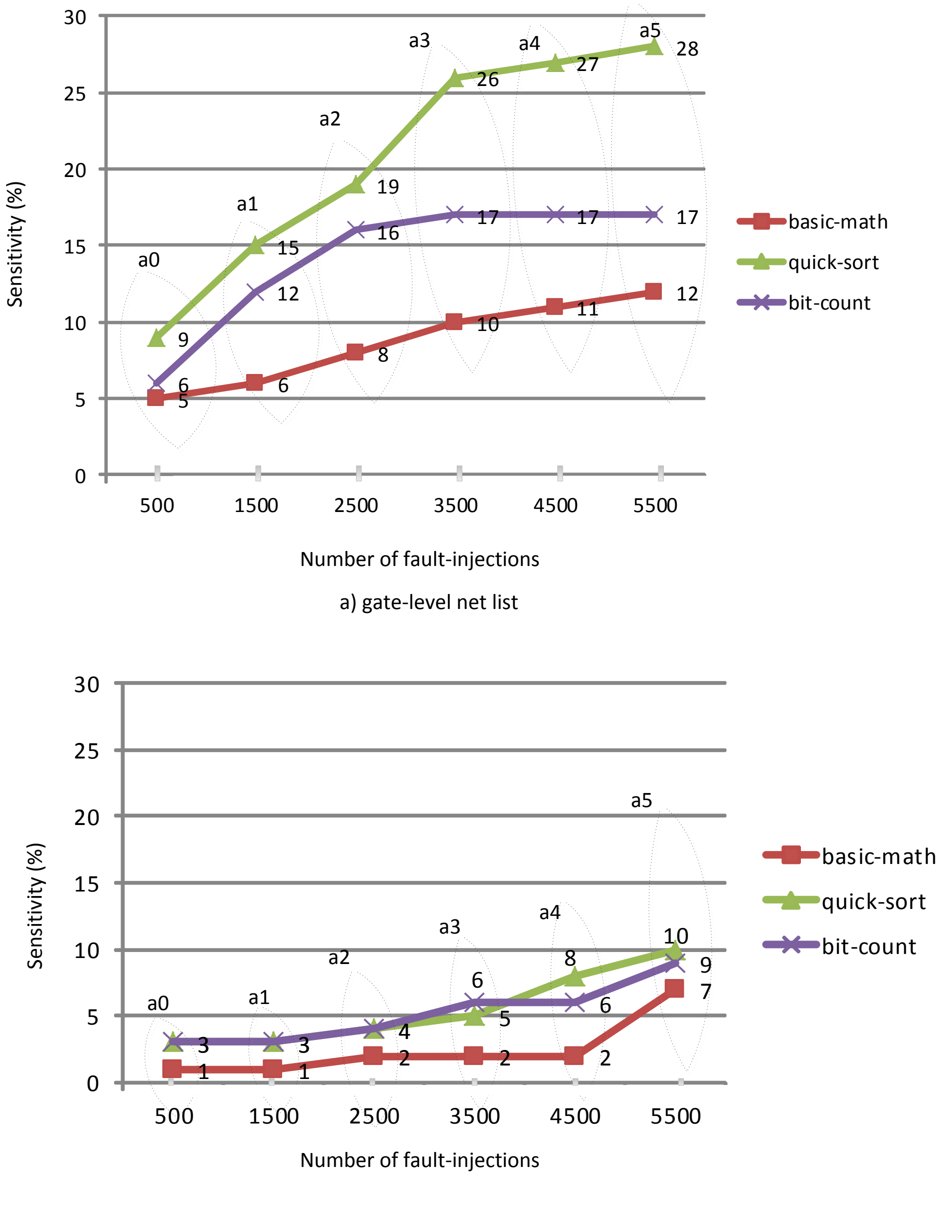
Here A_E, A_S, A_A, A_C, A_P, A_M denote the area of each functional-unit (E, S, A, C, P and M units, respectively) and the A_{total} represents the total area of the data path. This information can be extracted from the synthesis results of the Xentium Processor. The parameters P_E, P_S, P_A, P_C, P_P, P_M in Equation (3.2) are the SDC sensitivity for each functional-unit obtained directly from fault-injection results.

In each fault-injection experiment, one net within a specific functional-unit is automatically selected; then the desired fault model is injected into the design and finally the simulation results of the processor are compared with the correct values obtained from fault-free simulations. The number of fault-injection experiments, which means a higher number of selected nets, has been increased from 500 (0.5K) to 5500 (5.5K) experiments. It is important to note that all fault-injection experiments have been conducted for three processor workloads, quick-sort, bit-count and basic-math program from the Mibench benchmark repository [Gut01]. The basic-math workload implements more complicated functions, such as the square and exponential function.

The behaviour of each model of the Xentium net list for three different workloads has been depicted in the charts of Figures 3.7a and 3.7b. Figure 3.7a shows the results of the gate-level net list whereas Figure 3.7b depicts the results for the RTL net list. The Y-axis in these charts shows the sensitivity of the processor for Silent Data Corruption (SDC), because this category means the percentage of injected faults that produce a functional failure at the output. The X-axis represents the number of fault-injections. The initial value of the number of fault-injections has been randomly selected to 500. The selection of this number does not play any role in fault-injection results because this initial value will be increased until the difference between the results of fault-injection are negligible.

The first observation from the graphs of Figure 3.9 is with regard to the general behaviour of the RTL net list with regard to SETs. It can be seen that the RTL model shows lower sensitivity for SETs as compared to the gate-level net list. An explanation for this observation is that the gate-level net list has more possibilities for fault propagation than to the RTL gate-level net list. For example, an injected glitch will interfere with the delay of a gate in the gate-level net list and consequently changes the whole timing of an output, which in turn could result into a functional error; however this situation cannot occur in the RTL net list since there is no timing information for gates.

Our goal is to find the point where fault-injection results are almost identical and it is not worth increasing the number of fault-injection results after that. We call this behaviour the point of convergence. Also, the contribution of each net list to produce a point of convergence will be discussed. As a general rule, the sooner fault-injection results can reach a point of convergence, the better that net list qualifies to explore a fast estimation of the fault behaviour of the processor.



a) gate-level net list

b) RTL-level net list

Figure 3. 9. Sensitivity for SDC faults of different workloads versus the number of fault-injections (a0 to a5 correspond to the first to sixth campaign of fault-injections, respectively).

To explore the convergence of RTL versus the gate-level net list, the theory of convergence of real numbers has been employed; i.e. the distance between two real numbers is the absolute value of their difference. For example, if 'a_i' and 'a_j' are two terms of a sequence 'a_n', the distance between 'a_i' and 'a_j', denoted by '$d(a_i,a_j)$' is defined as:

$$d(a_i,a_j) = | a_i - a_j | \qquad (3.3)$$

We define '$a_{convergence}$' as a limit-of-sequence 'a_n' if for all terms after '$a_{convergence}$' the difference between two sequential terms is negligible. In another words, after some point, the terms of a sequence should get closer to each other. The mentioned behaviour can be formulated as
(Ɛ means a very small value and ∀ means for every instance)

$$d\left(a_n, a_{convergence}\right) < \mathcal{E}, \forall n > n_0 \qquad (3.4)$$

Which means for all the items after the location n_0 the difference between those items and the limit-of-sequence '$a_{convergence}$' *is* negligible. Furthermore, one can conclude that the sequence 'a_n' is a convergent sequence and it converges to '$a_{convergence}$'. In our experiments, 'a_n' is defined as the series of SDC sensitivities, which can be extracted directly from Figure 3.9. As an example, in the case of the gate-level net list (Figure 3.9a) for the basic-math benchmark, one obtains sensitivities as a_0=5%, a_1=6%, a_2=8%, a_3=10%, a_4=11% and a_5=12% (also indicated in the figure). It is interesting to know how the results of convergence of each fault-injection are. Let us assume that '$a_{convergence}$' in each fault-injection case is the SDC sensitivity of 5500 fault-injections for that specific workload, or 'a_5' (which can be directly extracted from the graphs in of Figure 3.9). For example, the '$a_{convergence}$' of the gate-level net list for the basic-math workload is 12%. If one can find an n_0 value in Equation (3.4) while the Ɛ value is relatively small (for example 0.2 as compared to 3), one could state that the fault-injection results are relatively increasing in terms of convergence.

To achieve the mentioned goal, the distance 'd' needs to be calculated for different numbers of fault-injections in each net list/workload. A reasonably small value of ‘d’ for the smallest number of fault-injections is an indication in terms of the speed of convergence of that sequence.

Table 3.5 shows the calculated ‘d’ for the gate-level net list for different numbers of fault-injections. Table 3.6 depicts ‘d’ for the RTL net list for different numbers of fault-injections. These numbers have been directly extracted by calculating the difference between the SDC sensitivity of a specific number of fault-injections (the chart in the Figure 3.9) and the SDC sensitivity of 5500 fault-injections for the same net list/workload. As the difference between 5500 fault-injections and itself is zero, the ‘d’ value for 5500 fault-injections has been removed from the tables. For example, in the case of the gate-level fault model and basic-math workload, a_1 is 6% while the ‘$a_{convergence}$’ is 12%, which gives:

$$d = |a_1 - a_{convergence}| = 6\%. \qquad (3.5)$$

This value has been indicated by underlining in Table 3.5. All values of ‘d’ in the Tables 3.5 and 3.6 have been calculated in a similar way. The last row in each table shows the average ‘d’ of all three workloads.

Table 3.5. Distance d between the SDC sensitivity and the SDC of 5500 fault-injections for the gate-level net list.

Benchmark	**Number of fault-injections**				
	500	**1500**	**2500**	**3500**	**4500**
Basic-Math	7	<u>6</u>	4	2	1
Quick-sort	17	13	9	2	1
Bit-count	9	5	1	0	0
Average value for three workloads	11	8	4.6	1.3	0.6

Table 3.6. Distance d between the SDC sensitivity and the SDC of 5500 fault-injections for RTL net list.

Benchmark	Number of fault-injections				
	500	1500	2500	3500	4500
Basic-Math	6	6	4	4	4
Quick-sort	5	5	6	5	2
Bit-count	6	6	3	4	3
Average value for three workloads	5.6	5.6	4.3	4.3	3

Let us assume that the epsilon value Ɛ is considered 2 in Equation (3.4). The average values (last rows) of Table 3.5 show that the gate-level net list indicates a better convergence with regard to the RTL net list if the number of experiments are equal or more than 3500; this is because the value 'd' for the average workloads is relatively small in the gate-level fault-injections at this point and after that. The RTL net list shows a smaller 'd', only if the number of fault-injections are smaller than 1500 (5.6 as compared to 11 and 5.6 as compared to 8); however, the d value for the gate-level net list is smaller if the number of experiments is more than 1500.

The above discussion shows that the results of increasing the number of fault-injections are more convergent in the gate-level net list if the number of experiments reaches 3500 fault-injections. In other words, the response of the Xentium processor will not be dependent on the number of fault-injections exceeding 3500.

As a summary, with the involvement of timing information of the net list and a sufficient number of fault-injections, the experimenter can rely on a relative small number of experiments to come to a solid conclusion with regard to the SDC sensitivity of a circuit under analysis. However, including timing information during the fault-injection phase will result in an increased value of the elapsed CPU time; this, because the Standard-Delay-Format (SDF) back annotation to assign the timing information to each net increases the simulation time. However, by using the developed framework in section 3.3, the overall experiments can be carried out faster.

3.6 Conclusions

In this chapter, an accelerated simulation-based fault-injection technique for soft-errors was presented. Considering that the main drawback of simulation-based fault-injection techniques is the long elapsed CPU time required to conduct experiments, the main goal of the proposed framework was to decrease the elapsed CPU time required to carry out fault-injection experiments. This goal was achieved by using the benefits of the saboteur technique in order to model different faults along with built-in commands techniques to speed-up fault-injections. A special distribution function, the exponential distribution, was used to statically schedule the characteristics of each FIS signal at the start of fault-injection experiments, during the start-up phase. Simulation experiments carried out on an AVR microprocessor revealed that the elapsed CPU time improves between 27% to 67% as compared to conventional fault-injection models.

Moreover, the importance of timing information of the net list in soft-error sensitivity analysis was addressed in this chapter. Conducting fault-injections on the gate-level net list which has timing information leads to a faster point of convergence in fault-injection results.

References

[Arl03] J. Arla, Y. Crouzet, J. Karlsson et al., "Comparison of physical and software implemented fault-injection techniques," in IEEE International On-Line Testing Symposium (IOLTS), pp. 1115-1133, 2003.

[Ale11] D. Alexandrescu, E. Costenaro, M. Nicolaidis, "A practical approach to single event transients analysis for highly complex designs," in IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), pp. 155-163, 2011.

[Ale12] D. Alexandrescu, E. Costenaro, "Towards optimized functional evaluation of SEE-induced failures in complex designs," in IEEE International On-Line Testing Symposium (IOLTS), pp. 182-187, 2012.

[Ava12] A. Avans, M. Nicolaidis, W. Shi-Jie et al., "RIIF - reliability information interchange format," in IEEE International On-Line Testing Symposium (IOLTS), pp. 103-108, 2012.

[Asa12] H. Asadi, M. B. Tahoori, "Soft-error modeling and remediation techniques in ASIC designs," in Microelectronics Journal, Vol. 41, No. 8, pp. 506-522, 2012.

[Bar00] J. C. Baraza, J. Gracia, D. Gil et al., "A prototype of a VHDL-Based fault-injection tool," in Proceedings of Design for Test Conference, pp. 396-404, 2000.

[Bar04] J. C. Baraza, J. Gracia-Moran, D. Gil-Tomas et al., "A prototype of a VHDL-based fault-injection tool: description and application," in Journal of System Architecture, Vol. 74, No. 2, pp. 847-867, 2004.

[Bar05] J. C. Baraza, J. Gracia, D. Gil et al., "Improvement of fault-injection techniques based on VHDL code-modification," in IEEE International Conference on High-Level Design Validation and Test (HLVT), pp. 16-26, 2005.

[Ben03] A. Benso, P. Prinetto, "Fault-injection techniques and tools for embedded systems reliability evaluation," Springer, ISBN 978-0-306-48711-8, 2003.

[Ben98] A. Benso, M. Rebaudengo, L. Impagliazzo et al., "Fault-list collapsing for fault-injection experiments," in Proceeding of the Reliability and Maintainability Symposium, pp. 383-388, 1998.

[Ber02] L. Berrojo, I. Gonzalez, F. Corno et al., "New techniques for speeding-up fault-injection campaigns," in Proceedings of Design Automation and Test in Europe (DATE), pp. 847-852, 2002.

[Gil03] D. Gil, J. Gracia, J. C. Baraza et al., "Study, comparison and application of different VHDL-based fault-injection techniques for the experimental validation of a fault-tolerant system," in Microelectronics Journal, Vol. 34, pp. 41-51, 2003.

[Gil08] D. Gil, J. C. Baraza, J. Gracia et al., "VHDL simulation-based fault-injection techniques," in Fault-Injection Techniques and Tools for Embedded Systems Reliability Evaluation, Springer, ISBN 978-0-306-48711-8, 2008.

[Gar12] M. Garcia-Valderas, R. Fernandez-Cardenal, A. Lindoso et al., "Soft-error sensitivity evaluation of microprocessors by multilevel emulation-based fault-injection," in IEEE Transactions on Computers, Vol. 61, No. 3, pp. 313-322, 2012.

[Gra10] J. Gracia-Moran, D. Gil-Tomas, J. C. Baraza et al., "Searching representative and low cost fault models for intermittent faults in microcontrollers: a case study," in Pacific-Rim International Symposium on Dependable Computing (PRDC), pp. 11-18, 2010.

[Gri11] J. Grinschgl, A. Krieg, C. Steger et al., "Automatic saboteur placement for emulation-based multi-bit fault-injection," in International Workshop on Reconfigurable Communication-Centric Systems-on-Chip, pp. 51-8, 2011.

[Gut01] M. R. Guthaus, J. S. Ringenberg, D. Ernst et al., "Mibench: a free, commercially representative embedded benchmark suite," in IEEE Workshop on Workload Characterization, pp. 3-14, 2001.

[Gaw10] P. Gawkowski, G. Smulko, "Speeding-up fault-injection experiments with dynamic code injection," in International Conference on Information Technology, pp. 171-174, 2010.

[Jen94] E. Jenn, J. Arlat, M. Rimen et al., "Fault-injection into VHDL models: the MEFISTO tool," in International Symposium on Fault-Tolerant Computing, pp. 66-75, 1994.

[Kar04] T. Karnik, P. Hazucha, "Characterization of soft-errors caused by single event upsets in CMOS processes," in IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 2, pp. 128-143, 2004.

[Lev05] R. Leveugle, "A new approach for early dependability evaluation based on formal property checking and controlled mutations," in IEEE International On-Line Testing Symposium (IOLTS), pp. 260-265, 2005.

[Lee09] D. Lee, J. Na, "A novel simulation fault-injection method for dependability analysis," in IEEE Design and Test of Computers, pp. 50-59, 2009.

[Men10] Mentor Graphics, www.mentor.com, 2010.

[Mis07] S. Misera, H. T. Vierhaus, A. Sieber, "Fault-injection techniques and their accelerated simulation in SystemC," in Euro-Micro Conference on Digital System Design Architectures, Methods and Tools (DSD), pp. 587-595, 2007.

[Nic11] M. Nicolaidis, "Soft-errors in modern electronic systems," Springer, ISBN 978-1-4419-6993-4, Cambridge, 2011.

[Pec13] A. Pecchia, A. Lanzaro, A. Salkham et al., "Leveraging fault-injection techniques in critical industrial applications," in Innovative Technologies for Dependable OTS-based Critical Systems, ISBN-13: 9788847027718, Lecture Notes in Computer Science (LNCS), 2013.

[Reb00] M. Rebaudengo, B. Prrota, M. Violante et al., "New techniques for accelerating fault-injection in VHDL description," in Proceedings of International On-Line Test Workshop (IOLTS), pp. 61-66, 2000.

[Rec11] Recore-systems, www.recoresystems.com, 2011.

[Rod02] F. Rodriguez, J. C. Campelo and J. J. Serrano, "Reducing the VHDL-based fault-injection simulation time in a distributed environment," in Proceedings of European Test Workshop (ETS), pp. 40-50, 2002.

[Sun11] A. Sunga, B. Choia, W. E. Wongb et al., "Mutant generation for embedded systems using kernel-based software and hardware fault simulation," in Journal of Information and Software Technology, Vol. 53, No. 10, pp. 1153-1164, 2011.

[Sha12] S. Z. Shazli, M. B. Tahoori, "Using Boolean satisfiability for computing soft-error rates in early design stages," in Microelectronics Journal, Vol. 50, No. 1, pp. 149-159, 2012.

[Tou07] E. Touloupis, J. A. Flint, V. A. Chouliaras et al., "Study of the effects of SEU induced faults on a pipeline protected microprocessor," in IEEE Transaction on Computers, Vol. 56, No. 12, pp. 1585-1596, 2007.

[Wan11] F. Wang, Y. Xie, "Soft-error rate analysis for combinational logic using an accurate electrical masking model," in IEEE Transactions on Dependable and Secure Computing, Vol. 8, No. 1, pp. 137-146, 2011.

[Zia04] H. Ziade, R. Ayoubi, R. Velazco, "A survey on fault-injection techniques," in International Arab Journal of Information Technology, Vol. 1, No. 2, pp. 171-185, 2004.

CHAPTER 4

Pulse-Length Determination Techniques for Rectangular SET Faults

Parts of this chapter have been published as paper titled "Pulse-length determination techniques in the rectangular single event transient fault model" in the IEEE International Conference on Embedded Computer Systems: Architectures, Modelling, and Simulation (SAMOS), 2013.

ABSTRACT - In the previous chapter, it was shown that the logic-gate level net list of a circuit is quite suitable to conduct simulation-based fault-injection with regard to Single Event Transients (SETs). However, the accuracy of the SET model itself is crucial in logic-gate level simulation-based fault-injections. For example, the fault duration of a rectangular pulse, which means how long a SET will manifest itself in a net list, will affect the influence of the injected fault on the system. To the best of our knowledge, there is still no consensus over techniques for determination of the length of SET fault models. This chapter addresses two techniques to determine the pulse length of a rectangular SET fault model. The first determination approach has been extracted from radiation testing (carried out by iRoC-Technologies [Iro12]) along with using a fined-grained transistor level SET analysis tool on simple library logic gates. The second determination approach has been extracted from analysing asymptotic behaviour of SETs in a 45nm CMOS technology node. These two determination techniques have been employed to develop two models for SETs at the logic-gate level. To examine the applicability of the developed models, they have both been applied to fault-injection experiments based on the logic-gate level net list of the Xentium processor. Fault analysis shows that applying these two fault models causes the fault-injection results to converge up to four times faster, as compared to conventional SET fault models.

4.1 Introduction

Single Event Effects (SEEs) have gained importance since the early nineties when several experiments repeatedly revealed that about one third of system failures are due to SEEs, rather than permanent faults [Rie94]. SEEs appear as a data corruption in the sequential or combinational logic of a digital circuit. As mentioned in Chapter 2, the impact of SEEs in the combinatorial logic might lead to a momentary voltage pulse at the output of the logic, a so-called Single Event Transient (SET) [Nic11]. This SET might be captured by a consecutive flip-flop and as a result, the status of a system will be erroneous.

Recently, there have been many industrial domains where the reliability of digital ICs is a crucial contributing factor in the reliability of the entire system. As examples, one can mention the automotive industry, in which full hybrid and auto-drive cars are about to be produced (at the time of writing this thesis, 2014), as well as medical instruments in which a single failure might harm a human life. As a consequence, there is a great interest to study the sensitivity of particular systems with regard to SEEs.

Fault-injection has long been recognized as a particularly attractive method to assess the vulnerability of a system with regard to SEEs [Arl03]. Fault-injection evaluates the vulnerability of a circuit under test by speeding up the occurrence rate of SEEs. Fault-injection can be carried out at different abstraction levels, including real-life radiation testing, simulation-based and emulation-based fault-injections [Zia04].

Starting from the first category, real-life radiation-testing is carried out by stressing the actual hardware with real environmental parameters, for example by means of a laser beam or bombarding it with high-energy particles. This injection method is similar to the real physical nature of SEEs, but conducting such experiments is very complex and expensive. Simulation-based fault-injections are conducted by modelling SEEs in a simulation model of a system; subsequently the faults will be evoked in the logic-gate level net list of a circuit. The simulation-based fault-injection is a very useful experimental approach to evaluate the vulnerability of a system; the system can still be under development, as only an HDL net list is used. Moreover, the simulation-based fault-injection provides a very high degree of controllability over where and at which time faults are injected. As a result, there is a high degree of observability of the propagation of an injected fault. Finally, emulation-based fault-injections which have recently been introduced in [Ent12], can combine the flexibility and controllability of simulation-based fault-injections with the speed of radiation-based fault-injections. However, the circuit under test must be fully synthesizable which limits the usage of benchmarks in fault-injection experiments.

Considering the above mentioned categories, simulation-based fault-injections have raised the attention in the academic community as well as in the industrial world [Arl03]. However, the development of a realistic simulation model is a crucial factor to conduct simulation-based fault-injections. In this chapter, the focus will be on the influence of SEEs in combinational logic, or SETs, and a realistic simulation model will be developed which can be used in complex logic-gate level net lists (in the remainder of this chapter, gate-level net list means logic-gate level net list).

One of the well-known models to imitate the effect of SETs in combinational logic is the rectangular pulse model (also known as the double exponential model) [Wir07]. However, determination of a pulse length which accurately imitates a realistic behaviour of a SET in this double exponential model is an open question. The key contribution of this chapter is the development of two approaches in order to determine the pulse length of a rectangular SET. The first approach of determination has been extracted based on the results of real-life radiation testing of a 45nm gate-level library along with a precise transistor-level SET analysis tool. The latter has been developed by iRoC-Technologies [Iro12]. This determination technique takes into account the contribution of different gates in the gate-level net list to determine its pulse length; i.e. it differentiates between an ‘OR’ gate and ‘AND’ gate to conduct a fault-injection. Moreover, a weighted probability of different pulse lengths is assigned to each gate, while in traditional pulse-length determination techniques a constant value is assigned to a pulse length.

The second pulse-length determination approach uses the asymptotic analytical behaviour of the SPICE representation of SETs [Lim12]. This model takes into account the runtime activities of the node of strike (node that is hit by a high-energy particle), i.e. whether the node of strike is idle or it is accessed by the circuit (read or write a value) at the time of striking.

In order to evaluate the accuracy of the developed SET models, four sets of fault-injection experiments with several SET models have been conducted on a DSP processor (post-synthesized, including timing information). The first two campaigns constitute of using the SET fault model with our developed pulse-length determination techniques. The other two campaigns use conventional determinations of the SET pulse length. Our case study employs a gate-level net list of the Xentium processor, developed by RecoreSystems which has been synthesized by using the 45nm Nangate library [Si212].

The remainder of this chapter has been organized as follows: section 4.2 briefly surveys some previous works dealing with SET fault models. Section 4.3 introduces the first pulse-length representation derived from physical laser-based stimulation along with a transistor-level SET-analysis tool. The development of the analytical-based pulse-length determination technique has been described in section 4.4. Section 4.5 discusses details of fault-injection experiments on the Xentium processor for different fault models, including our developed models as well as two conventional models. The results of fault-injection are discussed in section 4.6 and finally the conclusion is provided in section 4.7.

4.2 Conventional determination of pulse length in rectangular SETs

A considerable amount of literature has been published to develop a realistic model for a gate-level representation of a SET. As a definition, a SET is a momentary corruption of the voltage of a signal that would appear at a random time (known as time instance) and lasts for a brief period of time (known as fault duration) [Ent12].

There is a well-accepted model for the circuit-level representation of a SET pulse, i.e. the double exponential pulse model [Wir07, Nas07], as depicted in Figure 4.1.

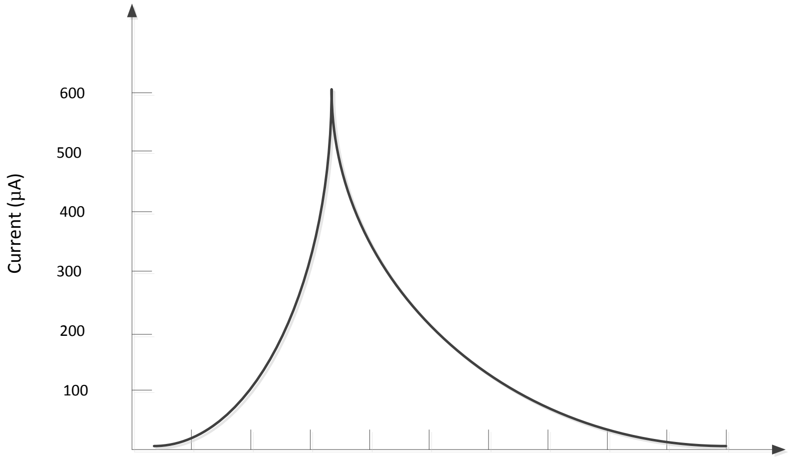


Figure 4.1. The double-exponential model of a SET [Wir07].

The pulse depicted in Figure 4.1. can have different durations. Up to this moment, there is no consensus on the pulse-length determination for this model in literature. For example, the authors in [Bar90] define the pulse length as the time between a time instance of the strike and the end of a simulation run, which is the time that a workload takes to be executed. The authors in [Rie94] and [She08] interpret a SET as a momentary pulse with a random time instance and a fixed fault length. The technique in [Rie94] deals with older CMOS technology nodes in which the length of a pulse is in the order of hundreds of nanoseconds, while [She08] determines the fault length in sub 100nm CMOS technology nodes in the order of hundreds of picoseconds. None of the above mentioned models take into account the contribution of different gates in the net list. As a result, a constant value is assigned to all SETs regardless of the specific gate. Recent work published in [Lim12] determines a fixed pulse length for every different gate in a gate-level net list, so the value of a pulse length depends on the gate on which the SET occurred. It will be shown that a more realistic determination technique is to consider a weighted probability of different pulse lengths for each gate. A publication that considers the system clock period in pulse-length determination is [Kan95], in which the pulse length is determined between the time instance until the start of the next succeeding clock edge. Another recent work, is [Ent12] where the authors take the length of each SET identical to the system clock period.

Recently, dedicated tools such as TFIT [Iro12] and HSECT-SPI [She08] have been specifically developed to predict and improve the logic gate Soft Error Rate (SER) performance at the circuit-level. These tools use a SPICE model (transistor-level) of a cell which is considered as the most accurate level of representation for a SET. While it perfectly fits for SET analysis purposes, a transistor-level description is cumbersome, difficult to generate for large circuits and cannot be fully simulated in acceptable time. A more practical approach consists in using a Gate-Level Net list (GLN, in a Verilog/VHDL format) complemented with timing information (e.g. SDF - Standard Delay Format files). However, SETs have to be represented in a logic model in order to be used within the mentioned descriptions. Our first determination technique of SETs in the circuit-logic model uses the well-known rectangular-pulse model in which its pulse length has been extracted based on the results of the SER characterization for a complete standard cell library (via TFIT). Hereafter, we call this first model the circuit-based model. The second model (hereafter to be referred as analytical-based model) uses the most common SPICE-level representation of a SET in order to develop a simplified logical pulse length.

4.3 The circuit-based determination approach

In order to extract the pulse length from real physical experiments, the first step consists in characterizing the electrical effects induced by energetic particles in a standard-cell library. This characterization is performed using a transistor implementation of each cell in a gate-level library. The used library is the 45nm open-access NANGATE repository [Si212]. The circuit implementation of the library and real measurements have been gathered in a SER database to build a logic fault model dedicated to each cell. As our ultimate aim is to determine a SET logic model for each cell, the following parameters with regard to the SET have been investigated:

a) The Pulse Length (PL) of the SET

b) The Soft-Error-Rate (SER) of a SET with a specific PL

The above mentioned information will be unique for each cell in the standard-cell library. A complete characterization will produce an occurrence rate for a specific pulse length for each gate (cell), represented as $\text{cell}_{(PL,SER)}$.

In order to find the (PL, SER) associated with each cell, an Electronic Design Automation (EDA) tool can be used. Dedicated tools such as TFIT [Iro12] from iRoC-Technologies represents a new generation of tools that allows a reasonably accurate calculation of the impact of electrical effects of particles with respect to a transistor or a library cell. A vast number of experiments have been carried out to specify a complete database for all possible $\text{cell}_{(PL,SER)}$ in the 45nm NANGATE library. Since the scope of this chapter is to use the outcome of those experiments to develop a pulse length in order to build a realistic logical SET-pulse model, the following paragraphs give a general overview of performing those experiments. The detailed explanation of experiments has been published by iRoC technologies in [Ale11].

The TFIT tool works by pre-characterizing a cell with regard to SETs. Different SETs are injected and simulated in all the relevant nodes of a library cell. The expected outcome is a SET with the minimal duration. Then, the tool analyses the operating environment, for example being exposed to neutron or alpha particles. TFIT uses a nuclear database to evaluate any possible secondary particle produced by an atomic reaction between a neutron/alpha particle and the silicon atoms. Direction and energy of those secondary particles are studied to account for their interaction with the sensitive volumes of the cell. Depending on the type of interaction, a current is injected while the output value of the cell is monitored to observe any possible electrical event. The tool also uses a technology SER process-response model (in this case a 45nm generic), which is a database where a collection of relevant (with respect to a given process technology) current-pulses are stored. These current-pulses are the ones used to perform the analysis. By cross-checking the possible environment-induced events versus the data recorded during the experiments, the TFIT tool is able to compute the Soft-Error-Rate (SER) value, expressed in FIT (Failure In Time) for each pulse length value.

The specific pulse length values have been defined as 50ps, 75ps, 100ps, 125ps, 150ps and 175ps. The occurrence probability of a pulse length smaller than 50ps is almost zero [Ale11]. Table 4.1 presents the *SER* (expressed in 'FIT') for different cells in the 45nm NANGate library. As an example, for the 'AND2' cell, the SER of a SET with a pulse length of 50ps is equal to 51.1 FIT, which means 51.1 times of occurrence in 114 years (1 billion hours).

Table 4.1. SER expressed in FIT for different cells in the NANGATE Library under different pulse lengths.

Cells	Pulse Lengths (PL)					
	50ps	75ps	100ps	125ps	150ps	175ps
AND2	51.10	29.88	9.39	0.30	0	0
NAND2	41.60	20.60	2.80	0.02	0	0
XNOR2	69.60	60.20	30.41	2.00	0	0
AND3	40.70	28.50	12.62	2.15	0	0
AND4	38.00	17.30	12.02	7.93	0.76	0
AOI211	41.30	17.30	12.02	7.93	0.76	0
AOI21	41.30	27.30	17.38	2.47	0.04	0
AOI221	40.40	24.00	19.54	11.30	2.22	0.07
INV	28.30	10.10	0.50	0	0	0
NAND3	44.30	25.70	10.29	0.40	0	0
NAND4	46.80	27.00	15.03	3.58	0.10	0
NOR2	30.90	20.49	7.35	0.15	0	0
NOR3	19.50	10.71	7.95	1.68	0.02	0
NOR4	16.80	5.56	4.48	3.88	0.19	0
OAI211	76.50	44.30	3.69	1.00	0.18	0
OAI21	66.70	44.26	20.66	0.83	0	0
OAI221	70.90	44.00	36.60	12.33	6.43	0.34
OAI222	61.30	39.10	28.10	23.60	15.99	5.51
OAI22	52.20	30.90	23.71	11.17	0.50	0
OAI33	43.00	21.00	14.20	11.10	10.10	2.60
OR2	56.90	33.70	10.93	4.75	0.10	0
OR3	52.50	26.90	9.96	6.15	0.61	0
OR4	54.20	23.90	6.41	3.99	2.90	0.15
TINV	12.10	11.33	6.61	0.21	0	0
MUX2	63.00	37.50	36.70	30.70	13.08	10.10
TBUF	16.20	13.23	7.94	0.88	0.50	0.20
XOR2	62.50	50.30	34.08	5.86	0.10	0
BUF	47.30	21.84	8.14	2.87	2.14	1.32
CLKBUF	48.10	22.22	3.39	0	0	0
FA on S	52.00	50.00	45.80	43.60	16.89	0.10
FA on CO	47.30	29.70	19.90	15.64	3.39	0.01
HA on S	64.80	42.30	34.35	5.47	0.09	0
HA on CO	55.20	29.22	4.50	0.08	0	0

A comprehensive list of (PL,SER) values for all the cells in the library (thirty-two cells), similar to Table 4.1, is stored in a database. The next step will be representing a logic-gate level model based on the SER calculations and pulse lengths.

Given the fact that the key concept of simulation-based fault-injections is to accelerate the occurrence rate of faults, assume the initial number of fault-injections for each cell is β_{total}. The logic fault model for one cell can then be represented by Equation (4.1):

$$cell_{PL} = \frac{SER_{cell_{PL}} * \beta_{total}}{\sum_{PL=50}^{175} SER_{cell_{PL}}} \text{ for all } cell \in lib \qquad (4.1)$$

(PL=50, 75, 100, 125, 150, 175)

Where cell_{PL} is the relative occurrence-number of a pulse with a length of 'PL' and $\text{SER}_{\text{cell}_{\text{PL}}}$ is the SER of a pulse with length 'PL' for a particular cell, labelled as 'cell' in the library 'lib' (this information can be directly extracted from the cell SER repository, Table 4.1.) Therefore cell_{PL} values can be readily calculated for all the possible PLs (six values in our experiments), and for all the possible cells in the library (thirty-two cells for our NanGate library) using any conventional mathematical tool.

As a result of Equation (4.1), the combination of (Cell, PL, cell_{PL}) can be defined for all the available cells and pulse lengths. This information will be stored in a database for any desirable number of fault-injections. Table 4.2 uses Equation (4.1) and the information of Table 4.1 to calculate the occurrence-number for each pulse length for every cell if the total number of fault-injections for each cell (β_{total}) equals to 1000 (this in order to simplify the calculations). This table shows that, e.g. for the 'AND2' cell, 564 out of 1000 SET signals have the length of 50ps, 329 have the length of 75ps and so on.

Table 4.2. Occurrence rate of all six different PLs if the total number of fault-injections for each cell is 1000.

Cell	Pulse lengths					
	50ps	75ps	100ps	125ps	150ps	175ps
AND2	563	329	103	3	0	0
NAND2	639	301	43	0	0	0
XNOR2	427	369	186	12	0	0
AND3	484	339	150	25	1	0
AND4	500	227	158	104	10	0
AOI211	500	227	158	104	10	0
AOI21	460	308	169	27	1	0
AOI22	424	321	195	56	1	0
INV	727	256	12	0	0	0
NAND3	549	318	127	5	0	0
NAND4	505	291	162	386	1	0
NOR2	524	347	124	3	0	0
NOR3	489	268	200	43	1	0
NOR4	543	179	144	125	6	0
OAI211	609	353	29	8	2	0
OAI21	503	334	155	6	0	0
OAI221	415	257	214	72	37	2
OAI222	353	225	161	135	90	0
OAI22	440	260	200	94	4	0
OAI33	265	129	87	68	62	17
OR2	520	308	127	43	1	0
OR3	540	279	103	63	6	0
OR4	592	261	70	43	31	2
TINV	400	374	218	6	0	0
MUX2	329	169	192	160	68	50
TBUF	415	339	203	23	13	5
XOR2	408	329	222	38	1	0
BUF	565	261	97	34	25	15
CLKBUF	675	303	45	0	0	0
FA on S	250	240	219	209	80	1
FA on CO	407	256	171	134	292	0
HA on S	440	287	233	37	1	0
HA on CO	620	328	50	1	0	0

Starting from an initial number of fault-injections, the simulator program selects a cell in the synthesized gate-level net list and based on the occurrence numbers shown in Table 4.2, the exact number of that specific pulse length will be injected into the selected cell. The initial number of fault-injections will be increased until a point of convergence for fault-injection results will be recognized. Figure 4.2 shows the relationship between FIT values of Table 4.1 and the occurrence rate derived by the simulator program. As can be seen in this figure, the FIT value of each cell per pulse length is taken from Table 4.1, considering that the total value for fault-injection is given by the experimenter; the occurrence rate of each pulse length for each cell can be determined by using Equation (4.1).

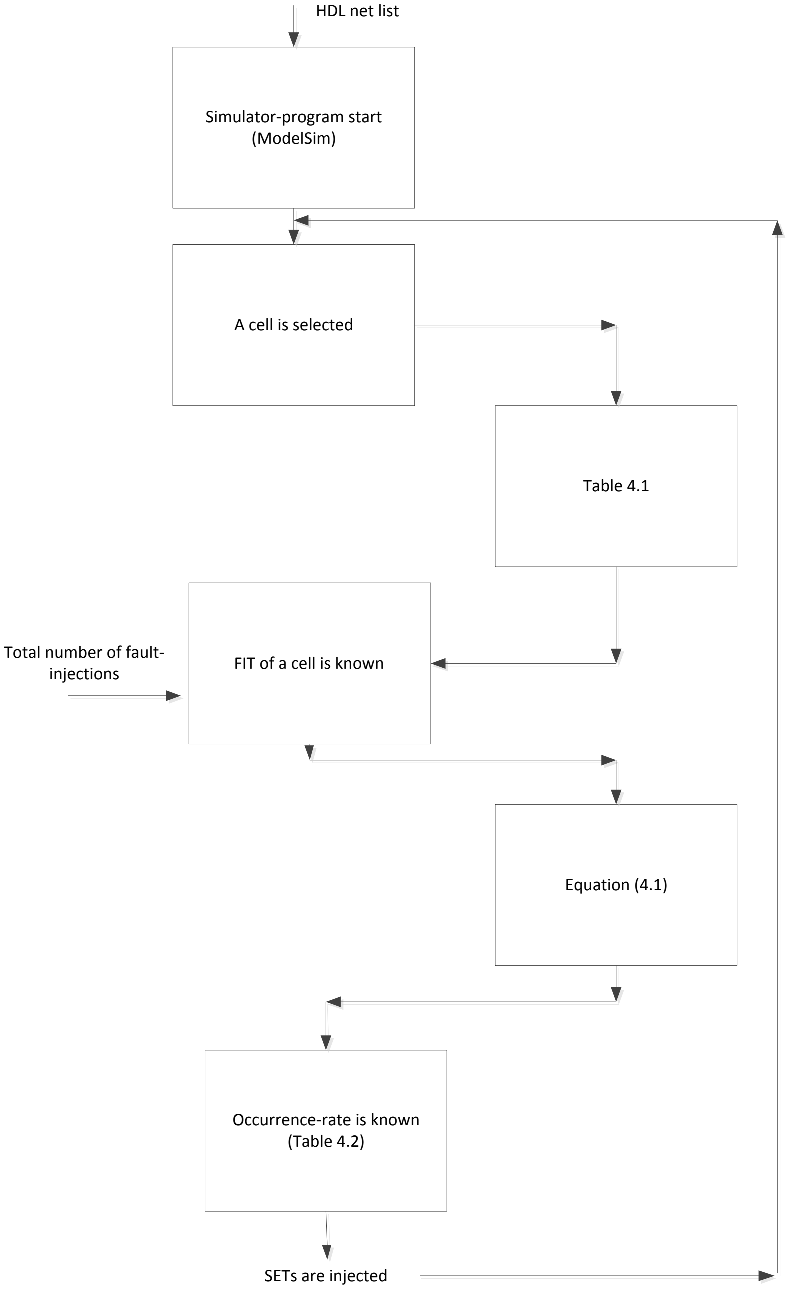

HDL net list
Simulator-program start (ModelSim)
A cell is selected
Table 4.1
Total number of fault-injections
FIT of a cell is known
Equation (4.1)
Occurrence-rate is known (Table 4.2)
SETs are injected

Figure 4.2. The relationship between FIT and occurrence rates.

Equation (4.2) shows a formula that takes into account the initial number of fault-injections as well as the final value to calculate the total number of fault-injections for each pulse length. The j value denotes the number of fault-injections which will be increased from an initial value until reaching the point of convergence. The details of reaching the point of convergence has already been explained in chapter 3.

$$cell_{PL} = \sum_{j=initial}^{end-point} \frac{SER_{cell_{PL}} * \beta_j}{\sum_{i=50}^{175} SER_{cell_i}} \quad \text{for all } cell \in lib \qquad (4.2)$$

Where β_j denotes the total number of fault-injections in each run and $cell_i$ is the occurrence rate of a pulse length with 'i' Pico-seconds on a gate labelled as 'cell'. It is important to mention that the proportion of different gates in a gate-level net list should be taken into account when the initial number of fault-injections is calculated. For example, if the number of 'NAND2' gates in the net list is two times the number of 'XOR2' gates (this information can be easily extracted from synthesis results), then if the initial number of fault-injections in 'NAND2' was defined as 100, the number of fault-injections in 'XOR2' should be defined as 50.

4.4 The analytical-based determination approach

This section describes a logical pulse model in which the pulse length will be determined based on the analytical model of a SET at the SPICE-level. The starting point is to define a SPICE representation of a SET. There are several models for such a phenomenon. The most commonly used representation is a transient pulse current $I_P(t)$, inserted between ground and the strike node, as shown in Figure 4.3.

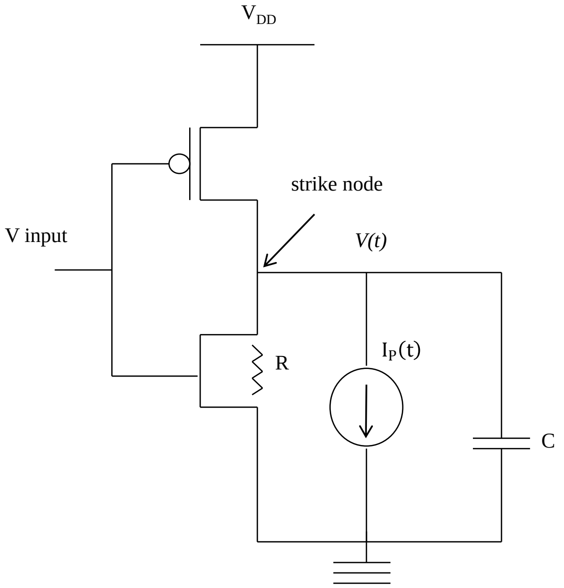


Figure 4.3. SPICE representation of the induced voltage V(t) in the strike node as result of a strike (R is the resistance of the NMOS).

Figure 4.3 indicates that part of the induced transient current comes from the node capacitance C, while the other part comes from VDD via the other transistor (in this case the PMOS transistor). While several models have been employed to formulate $\mathrm{I_P(t)}$, the most commonly accepted model is the double exponential pulse method which has been presented in [Wir07, Nas07] and given by Equation (4.3). This model has two timing parameters, α and β, which are respectively the rising and falling time constants of the exponential equation.

$$I_P(t) = I_0\left(e^{\frac{-t}{\alpha}} - e^{\frac{-t}{\beta}}\right) \qquad (4.3)$$

Where I_0 represents the maximum charge collection current. The values of I_0, α and β are dependent on the used technological process and the particle of interest.

In the following paragraphs, an analytical calculation will be made in order to simplify Equation (4.3) to determine the pulse length of this equation based on the timing parameters of the node, i.e. α and β. Parts of this calculation being Equations (4.4) to (4.6) originate from reference [Wir07]. The remainder has been developed by us.

From Figure 4.1, the following first-order differential equation can be derived (this is if the PMOS is off and the NMOS is on, so the output is 0):

$$C\frac{dV(t)}{dt}+\frac{V(t)}{R}=I_P(t) \tag{4.4}$$

Where $\mathrm{V(t)}$ is the voltage of the strike node and *R* is the resistance of the strike transistor (in this case NMOS transistor because it is on). Solving this differential equation provides the voltage $\mathrm{V(t)}$ at the strike node. This equation is given by:

$$V(t)=\frac{I_0\mathrm{\alpha R}}{\mathrm{\alpha-RC}}\left(e^{\frac{-t}{\mathrm{\beta}}}-e^{\frac{-t}{\mathrm{RC}}}\right) \tag{4.5}$$

We are interested to extract two parameters from Equation (4.5). First, the time $\mathrm{t_{peak}}$ at which the strike node voltage $\mathrm{V(t)}$ reaches its peak value $\mathrm{V_{peak}}$. The other parameter is $\mathrm{t_{de-active}}$, the time when the SET voltage will be de-activated due to conduction of the NMOS transistor. The difference between $\mathrm{t_{peak}}$ and $\mathrm{t_{de-active}}$ defines the pulse length. The threshold of de-activation is set to $\frac{V_{DD}}{2}$, in order to simplify the calculations. Using mathematical optimization theory, the value of $\mathrm{t_{peak}}$ can then be represented as:

$$t_{peak}=\frac{\ln(\frac{\mathrm{\alpha}}{RC})\mathrm{\beta RC}}{\mathrm{\alpha-RC}} \tag{4.6}$$

Solving Equation (4.5) for $\mathrm{V(t)}$ = $\frac{\mathrm{V_{DD}}}{2}$ will provide the value of $\mathrm{t_{de-active}}$, as shown in Equation (4.7):

$$t_{de-active} = \frac{\ln(\frac{\alpha}{RC})\beta RC}{\alpha - RC} - RCln\left(\frac{\frac{V_{DD}}{2}}{V_{peak}}\right) - \alpha \ln\left(\frac{\frac{V_{DD}}{2}}{V_{peak}}\right) \quad (4.7)$$

As the Pulse Length (*PL*) is defined as the difference between $\mathrm{t_{de-active}}$ and $\mathrm{t_{peak}}$:

$$PL = t_{de-active} - t_{peak} = N.RC + \alpha.\mathrm{N} \quad (4.8)$$

Where N denotes $ln\left(\frac{\frac{V_{DD}}{2}}{V_{peak}}\right)$. In Equation (4.8) α is a constant number which depends on the technology of implementation. However, it will be shown that using asymptotic analysis [Mil06] applied to Equation (4.8) will decrease the importance of α in calculating the pulse length.

Equation (4.8) shows the pulse length as a function of the strike-node parameters, such as R, C and α. It is important to mention this model of pulse length is valid if the maximum amplitude of a SET (V(t)) is lower than the V_{peak} (which is typically true). In the following paragraphs, the asymptotic behaviour of Equation (4.8) will be exploited in order to define a simplified model for PL.

Let us suppose that RC is much larger than α, in Equation (4.8). In that case the Pulse Length (PL) will be dominated by the RC of the strike transistor and the particle would not have sufficient energy to alter the output of the gate (V(t)). This condition would be fulfilled if the strike transistor (NMOS) changes its state (on to off or vice-versa) by $\mathrm{V_{input}}$ at the moment of strike. However, if α is much larger than RC, then the pulse length (PL) will be dominated by the energy of the particle. Hence, as long as there is no transaction happening at the strike node, the pulse length will be dominated by the energy of the particle; otherwise the pulse length will be dominated by the parameters of the strike node (R) and the load (C) of the strike node.

The described behaviour can be implemented in modern logic simulators via advanced programming languages. We have developed a VPI interface (Verilog Peripheral Interface) that emulates the described behaviour of Equation (4.8). The gate-level net list of the Xentium processor has been used to assess the applicability of the SET models described in this chapter.

The following mechanism has been used to apply the asymptotic model (Equation (4.8)) via a logic simulator. During execution of a workload, a random signal will be selected for fault-injection. Also a random time will be assigned to conduct a single fault-injection. The logic simulator tracks the net that has been selected for fault-injection at the time of injection; if there is a transaction at the strike node, such as writing a new value, the impact of the particle will be dominated by the transaction. However, a particle can impose a perturbation on a node if at the time of strike there is no transaction on that particular node. In this situation, the particle can change the status of a circuit and the SET model will behave as the one developed in section 4.3.

In the next section, the behaviour of the two presented models will be compared to some conventional SET models by carrying out a fault-injection campaign on the Xentium processor. It is important to mention that based on the experiments carried out by iRoC-Technologies [Iro12] on simple gates, the model of section 4.3 only deviates 15% from the real-life laser-based experiments. However, conducting such a laser-based experiment for a DSP processor is very complex. In the section it is assumed the model of section 4.3 is the most accurate model (based on the laser experiments on simple gates). Our goal is to assess the accuracy of the model presented in section 4.4 with regard to the model presented in section 4.3; moreover, the accuracy of some conventional SET models will be evaluated.

4.5 Details of fault-injection

To show the behaviour of different pulse-length determination techniques in the rectangular SET fault model, four different sets of gate-level fault-injection campaigns have been conducted on the Xentium processor. The detailed architecture of the Xentium processor was discussed in Chapter 2 of this thesis.

The first fault-injection campaign uses the circuit-based extracted pulse length which is based on Equation (4.2) in section 4.3. This model takes into account a weighted contribution of each cell in the library as well as the system clock. Moreover, following the laser-based experiments carried out by iRoC-Technologies on basic library gates, it is known that this pulse model only deviates 15% from the real impact of laser-based experiments. Since conducting laser-based experiments for a complex processor is not feasible, we consider this circuit-based model as the baseline for comparison to assess the accuracy of other determination techniques.

The second campaign is based on the analytical pulse-length determination technique which has been discussed in section 4.4, which is represented by Equation (4.8). This model takes a possible transaction of each strike node into account along with the circuit-based model to construct a pulse-length model. In order to compare the efficiency of these two SET pulse-length determination models over conventional models, two other sets of fault-injection experiments have been carried out.

The third campaign uses a conventional determination technique of SETs where a constant pulse length value is assigned to every gate that is injected with a fault [She08]. The constant value is different from 100ps to 300ps for different CMOS technology nodes. However, we have selected the value of 100ps which has been recommended for a 45 / 90nm CMOS technology node [She08].

Finally the fourth campaign employs the discrete logic of a SET pulse model that calculates pulse-lengths based on an exponential distribution-function of the system clock period. The details of this model have already been discussed in Chapter 3. This model takes the system clock period into account, but not the contribution of different cells in the library. Table 4.3 shows the parameters which have been considered in each model.

Table 4.3. Parameters which have been taken into account in different pulse-length models.

Pulse-model	System clock	Cell contribution	Technology
Circuit-based	yes	yes	yes
Analytical-based	yes	yes	yes
Constant	yes	no	yes
Distribution-function-based	yes	no	yes

A digital signal processing program, being the Finite Impulse Response (FIR), has been used as a workload for the Xentium processor in all fault-injection experiments. The Xentium processor receives all the required inputs (filter coefficients, input vector) from an input text file and produces the output vector in an output text file. The required time to execute one run of the FIR program is about eight seconds (in real time on the simulation host computer).

The types of failures in the processor can be classified as Silent Data Corruption (SDC) or Detected Unrecoverable Error (DUE) [Muk08]. Since the original design of the Xentium processor does not have an error indicator signal, which indicates detection of an error by the Xentium processor, the experimenter cannot observe DUEs. As a result, the functional response of the processor with respect to each injected fault has been classified into one of the following categories, as described in Chapter 3:

- Silent Data Corruption, SDC: this condition is met if the error propagates through the circuit without awareness of its occurrence by the system. So the processor will provide an output without any error flag while that output is not correct.
- Time out: is the possibility that the processor unexpectedly stops its application, before execution of the whole workload. The outputs of the processor provide no meaningful output data in this case.

- Correct behaviour: the processor completes the application with the correct output.

The results of the processor have been represented as a percentage (%), e.g. if 10 out of 100 fault-injections produce a Silent Data Corruption (SDC) failure, then the SDC failure-sensitivity of the processor will be represented as 10%.

The fault-injections have been carried out on all ten functional units of the Xentium processor, which was shown in Figure 2.7 in chapter 2. In each fault-injection experiment, one net within one functional unit is automatically selected; then the desired fault model is injected into the net list and finally the results of the processor are compared with the correct values obtained from a Java-based fault-free simulator, called XentiumSim [Rec10]. The number of experiments has been increased from 500 to 16,000 experiments. It is worth mentioning that with 500 experiments, only 1% of all nets in the Xentium data path are affected. This rate is about 32% if the number of experiments reaches 16000. Table 4.4 shows the percentage of nets which are affected, as well as the elapsed CPU time (on a dual six-core Intel processor) if the number of fault-injection grows from 500 to 16000.

Table 4.4. Percentage of affected nets and elapsed CPU-time for different numbers of fault-injections.

Parameters of fault-injection	**Number of fault-injections**					
	500	**1000**	**2000**	**4000**	**8000**	**16000**
% of affected nets	1	2	4	8	16	32
CPU time (hours)	1.1	2.3	5	11	24	49

4.6 Experimental results

The behaviour of the Xentium processor for the rectangular SET model with four different pulse-length determination techniques has been depicted in Figure 4.4. The Y-axis in this chart shows the sensitivity of the processor for Silent Data Corruption. This sensitivity is derived based on the fault- injection experiments. The X-axis is the number of fault-injections (campaign) which ranges from 500 to 16000. The initial number of fault-injection experiments has been set randomly. Increasing the number of fault-injections is stopped at 16000 fault-injections because the elapsed CPU-time to accomplish this number of fault-injections then reaches forty-eight hours (two days).

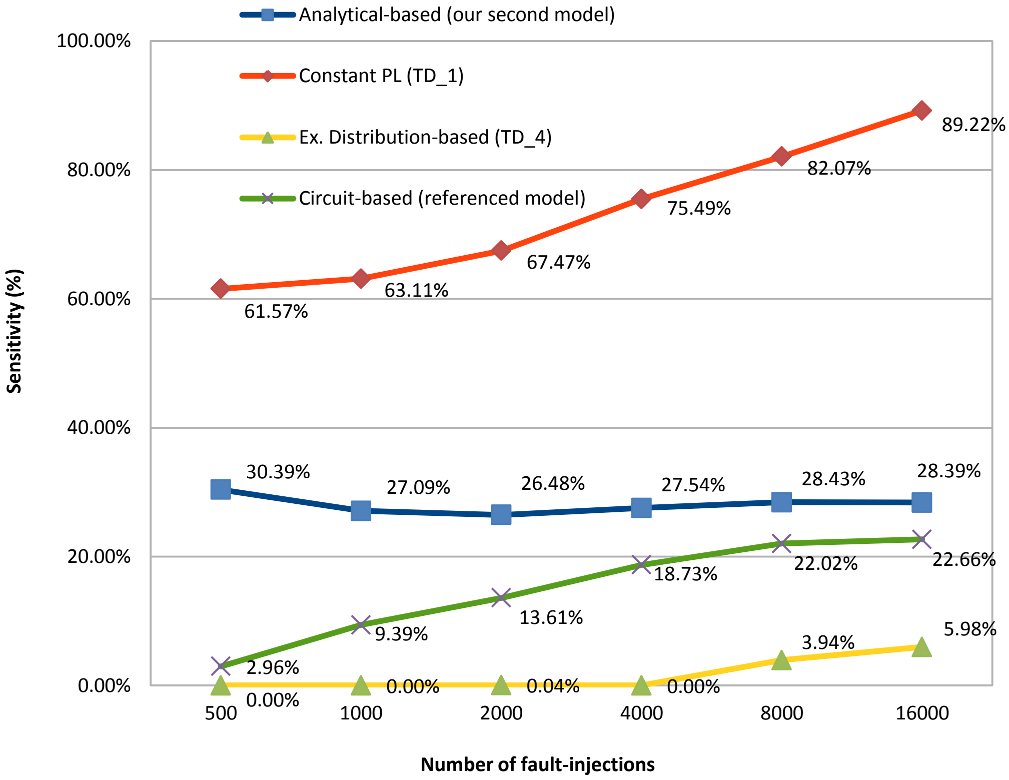


Figure 4.4. The SDC sensitivity of the Xentium processor for different SET fault models [1].

[1] TD_1 stands for Traditional model 1 and TD_4 stands for Traditional model 4

The first observation from Figure 4.4 can be made about the general behaviour of conventional SET models, labelled with constant Pulse Length (PL) and depicted with the red line (TD_1). The constant length of each pulse has been set to 100ps to be consistent with the associated literature [She08]. It can be seen that the constant pulse-length model overestimates up to three times the influence of SETs in the Xentium processor as compared to the other fault models. This can be explained due to the worst-case scenario which is applied to the circuit under test by using this model. For example Table 4.2 clearly shows that most of the pulses in the realistic pulse-length model (circuit-based) have a duration less than 100ps and contribution of longer pulse durations is very small. Our further detailed analysis of fault propagation in this model shows that many signals will be forced to change their value for a relatively longer time and as a result an injected fault usually changes the status of the succeeding storage elements. This in turn results to a pessimistic model of SET contribution in failures.

The exponential distribution-based model has been explained in chapter 3 and is depicted by a yellow line (TD_4). It is important to mention that the workload here is different from the ones being used in chapter 3; as a result the behaviour of the Xentium processor is different as compared to the results of chapter 3. As can be seen in Figure 4.4, this model underestimates the contribution of SETs as compared to the realistic fault model (circuit-based model), especially for a low number of fault-injections. However, if the number of fault-injection grows, the results will be closer to the realistic model. This can be explained due to the fact that by increasing the number of fault-injections more perturbations are generated overlapping with clock edges (the so-called effective faults). However, we were not able to see the actual behaviour of this model with regard to fault-injections for higher numbers (more than 16000) of experiments, since conducting fault-injection with a higher number of fault-injections was not manageable in time. The conclusion for this fault model is that a low number of fault-injections is useless for this fault model while extracting a convergence point requires a very high number of fault-injections. Subsequently an enormous amount of elapsed CPU time is required by using this model.

The behaviour of the Xentium processor for the two newly developed fault models is also shown in Figure 4.4. This figure shows that if the number of fault-injections grows, the behaviour of the circuit-based model and analytical-based model will be similar. Since the experiments which have been carried out by iRoC-Technologies on simple gates showed that the circuit-based pulse determination technique only deviates only 15% from the real behaviour of soft-errors, we consider this model as the most accurate model of SETs. Figure 4.4 shows that the behaviour of fault-injection for the analytical-based pulse model is getting close (6% difference for 16000 fault-injections) to the circuit-based model. This is because the analytical-based model uses circuit-based model determination if there is no activity at the striking node. However an important interesting aspect is the speed of convergence of the response of the processor for these two developed pulse models. In the other words, the sooner a fault model can reach a point of convergence, the better that fault model is to explore a fast anticipation of the behaviour of a system with regard to SETs.

A detailed analysis about finding a point of convergence in fault-injection experiments was already provided in chapter 3 section 3.5. The extracted numbers of Figure 4.4 have been applied to Equations (3.3) to (3.5) to define a point of convergence for these two fault models.

Table 4.5 shows the calculated *d* for each fault model for different numbers of fault-injections. As mentioned before, these numbers have been extracted by applying Equations (3.3) to (3.5) of chapter 3 to the numbers of Figure 4.4.

Table 4.5. The distance 'd' between the SDC sensitivity and the final SDC (for 16000 fault-injections) for each fault model.

Fault model	Number of fault-injections				
	500	1000	2000	4000	8000
Analytical-based	2	1.3	1.91	0.85	0.04
Constant-based	27.6	26.1	21.75	13.73	7.15
Exponential distribution-based	5.9	5.9	5.9	5.9	2.04
Circuit-based	19.07	13.27	9.05	3.93	0.64

Referring to Table 4.5, it can be concluded that if Ɛ is assumed to be 4 (in Equation (3.4) of chapter 3), the analytical-based and circuit-based fault models show the best convergence (smaller d, which are indicated by underlining in Table 4.5) if the number of experiments are equal and more than 4000. This means that the results of the fault-injections is more stable for these two SET fault models, as compared to the constant-based and exponential distribution-based models if the number of fault-injections is 4000 or more. In other words, the response of the Xentium processor will be less dependent on the number of fault-injections for 4000 fault-injections and more in the case of analytical and circuit-based fault models. Hence that results after 4000 fault-injections already give a good indication for the SDC sensitivity of a system. For the other two fault models, the constant PL and exponential distribution-based models, the results of fault-injection are still deviating if the number of fault- injections is even in the order of 8000. This indicates that no conclusion can be made based on a small number of fault-injections for constant and exponential distribution-based fault models. One needs to increase the number of fault-injections for these two fault models (especially the constant -PL model, since it exhibits a large 'd' in Table 4.5) to reach convergence.

Figure 4.5 shows the 'd' number (the behaviour with regard to convergence) of the analytical-based model and the circuit-based model (derived from Table 4.5) in one diagram. The X-axis shows the number of fault-injections while the Y-axis shows the 'd' value for each number of fault-injections. As mentioned earlier, the circuit-based model has been considered as the most accurate simulation model in our experiments; however, an interesting observation in Figure 4.5 can be made by looking at the convergence of the analytical-based fault model for a very small number of fault-injections (starting from 1000) as compared to the circuit-based model. It shows that one is able to have a very quick estimation of the processor sensitivity with regard to a SET even with a very low number of experiments (1000 fault-injections instead of 16000), while experience shows that the required CPU time to carry out 1000 experiments is sixteen times smaller as compared to the required time to carry out all 16000 fault-injections (linear dependency). Therefore it has a linear relationship.

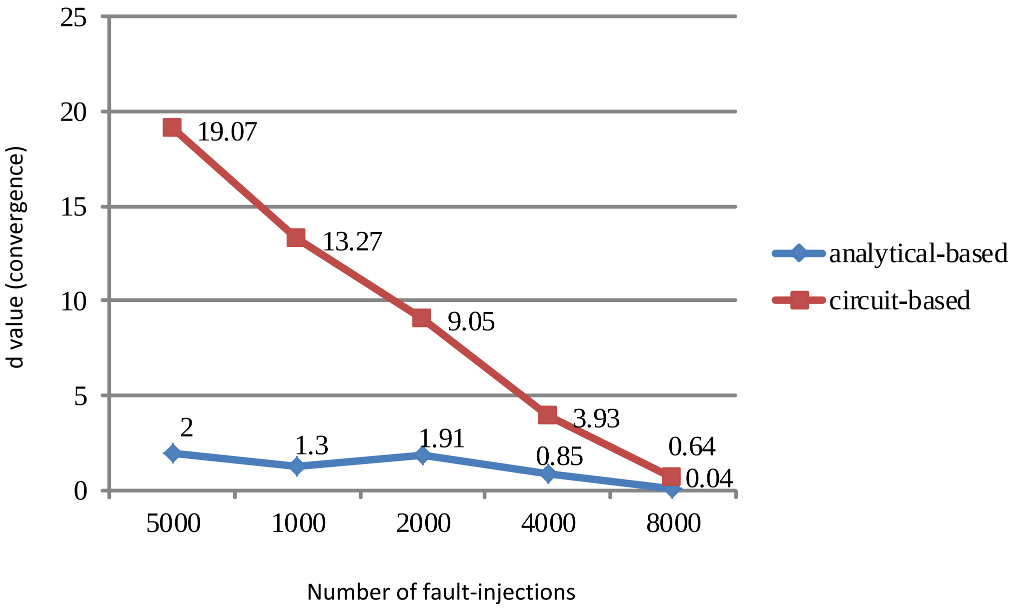


Figure 4.5. The convergence of the analytical-based and circuit-based models.

4.7 Conclusions

In this chapter, two approaches have been introduced for the determination of pulse lengths to be used in the rectangular SET logic model. The first method has been extracted from laser-based experiments along with a detailed transistor-level SET analysis tool. This model represents the most realistic model to anticipate the system response to SETs.

The second determination technique is based on the asymptotic behaviour of SETs in the SPICE model along with the circuit-based model. We showed that the analytical-based model can provide a very fast anticipation of the behaviour of the system while the accuracy of fault-injection results is very close to the final response of fault-injection. Hence, the circuit-based model is useful if the accuracy of fault-injection results is important (only 15% deviation from real-life laser-based experiments) while the analytical-based model is beneficiary if the elapsed time of fault-injection experiments is important (21% deviation from real-life laser experiments). These two fault models will contribute to solve the current challenge of developing/adopting EDA tools for fast and improved SER evaluation. In the next chapter, the behaviour of the Xentium processor with regard to SETs has been used to propose a dependable architecture for the data path and control-logic of each functional unit in the Xentium processor.

References

[Ale11] D. Alexandrescu, E. Costenaro, M. Nicolaidis, "A practical approach to single event transients analysis for highly complex designs," in IEEE Symposium on Defect and Fault Tolerance in VLSI Systems (DFTS), pp. 155-163, 2011.

[Arl03] J. Arlat, Y. Crouzet, J. Karlsson et al., "Comparison of physical and software implemented fault-injection techniques," in IEEE Transactions on Computers, Vol. 25, No. 2, pp. 247-252, 2003.

[Bar90] J. H. Barton, E. W. Czeck, Z. Z. Segall et al., "Fault-injection experiments using FIAT," in IEEE Transactions on Computers, Vol. 39, No. 4, pp. 575-582, 1990.

[Ent12] L. Entrena, M. Garcia-Valderas, R. Fernandez-Cardenal et al., "Soft-error sensitivity evaluation of microprocessors by multilevel emulation-based fault-injection," in IEEE Transactions on Computers, Vol. 61, No. 3, pp. 313-323, 2012.

[Iro12] iRoC-Technologies, www.iroctech.com/soft-error-tools/tfit-cell-level, 2012.

[Kan95] G. A. Kanawati, N. A. Kanawati, J. A. Abraham, "FERRARI: a flexible software-based fault and error injection system," in IEEE Transactions on Computers, Vol. 44, No. 2, pp. 248-260, 1995.

[Lim12] D. B. Limbrick, W. H. Robinson, "Characterizing single event transient pulse widths in an open-source cell library using SPICE," in IEEE Workshop on Silicon Errors in Logic System, pp. 6-10, 2012.

[Mil06] P. D. Miller, "Applied asymptotic analysis," American Mathematical Society, ISBN 978-0821840788, 2006.

[Muk08] S. Mukherjee, "Architecture design for soft-errors," Elsevier, ISBN 978-0-12-369529-1, 2008.

[Nas07] R. Naseer, Y. Boulghassoul, J. Draper et al., "Critical charge characterization for soft-error rate modelling in 90nm SRAM," in IEEE Symposium on Circuits and Systems, pp. 1897-1882, 2007.

[Nic11] M. Nicolaidis, "Soft-errors in modern electronic systems," Springer, ISBN 978-1-4419-6993-4, Cambridge, 2011.

[Rec10] Recore Systems, www.recoresystems.com, 2010.

[Rie94] G. L. Ries, G. S. Choi, R. K. Iyer, "Device-level transient fault modelling," in IEEE International Symposium on Fault-Tolerant Computing, pp. 86-94, 1994.

[She08] W. Sheng, L. Xiao, Z. Mao, "Versatile and efficient techniques for speeding-up circuit level simulated fault-injection campaigns," in IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 17-23, 2008.

[Si212] Si2, www.si2.org/openeda.si2.org/projects/nangatelib, 2012.

[Wir07] G. I. Wirth, M. G. Vieira, F. G. Lima, “Accurate and computer efficient modelling of single event transients in CMOS circuits,” in IET Journal of Circuits Devices Systems, Vol. 1, No. 2, pp. 137142, 2007.

[Zia04] H. Ziade, R. Ayoubi, R. Velazco, “A survey on fault-injection techniques,” in International Arab Journal of Information Technology, Vol. 1, No. 2, pp. 171-185, 2004.

This page intentionally left blank

CHAPTER 5

Soft-Error Mitigation Techniques for DSP Functional units

Parts of this chapter have been published as papers titled "An on-line soft-error mitigation technique for control logic of VLIW processors" in the international symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS 2013) in Austin, USA; and "Two soft-error mitigation techniques for functional units of DSP processors" in the European Test Symposium (ETS 2014) in Paderborn, Germany; and a European patent titled "Functional unit for a DSP processor" filed in November 2013, under the number EP13191370.

ABSTRACT- This chapter presents two soft-error mitigation techniques for Digital Signal Processing (DSP) processors. As explained in the previous chapters, each DSP processor consists of several functional units, which are subsequently composed of sequential parts and pure combinational logic. The sequential parts include a local control unit and input registers, while the combinational logic is just a collection of combinatorial logic gates. Since the control unit of each functional unit is an unstructured part, it is impossible to use Error-Detection-And-Correction (EDAC) codes to mitigate the impact of soft-errors in these units. Moreover, because the combinatorial nature of combinational logic, the effect of SETs will be destructive in the combinational parts of the functional unit. To develop an effective method to mitigate the effect of soft-errors in the two before-mentioned parts, unique characteristics of DSP workloads have been deployed to develop a masking mechanism for the local control unit of each functional unit. At the same time the combinational logic of each functional unit has been enhanced with a fast recovery mechanism to isolate the faulty unit from the other fault-free functional units and re-execute the erroneous instruction. An assumption in all the proposed methods is that the input registers inside of each functional unit are robust to soft-errors. This is a fair assumption since there are straightforward solutions such as EDAC codes to mitigate soft-errors in structured sequential parts of a design. The developed techniques have been implemented in the Xentium DSP processor, in order to assess the achieved enhanced SET resilience versus the imposed area and performance penalty. The experimental results show that the soft-error sensitivity will decrease by a factor of eight in the local control units and by a factor of two in the combinational logic. The penalty on area and clock-speed is less than 10%.

5.1 Introduction

Increasingly miniaturized CMOS processes along with the reduction of operating voltage have made soft-errors a major source of threat for today's digital Integrated Circuits (ICs). As discussed in the previous chapters, soft-errors can arise from different sources, including high-energy particles from cosmic radiations or terrestrial phenomenon such as power-supply sparks and high-energy particles emitting from inside the packaging due to impurities [Mie12]. The impact of soft-errors on a digital IC can be classified into two categories: Single Event Transient (SET) and Single Event Upset (SEU). In the case of SET, a high-energy particle hits the combinational logic of a circuit and consequently a momentary voltage pulse will be generated at the output of the strike gate; this in turn might reach either a storage element (flip-flop, register) or the output of the succeeding logic gates. Chapter 4 of this dissertation showed a simulation model for this kind of soft-errors. We have used that developed model to evoke the SETs in the Xentium DSP processor.

Next to SETs, a SEU will be generated if a high-energy particle directly hits the sequential parts of a processor, in which their stored value might be toggled [Sch04]. Even-though SEU was the main concern in the soft-error community in the past, it has been forecasted that increasing the system frequency will cause the system errors to be dominated by the SETs originating from the combinational logic rather than SEUs from the sequential logic [Tou07]. In other words, the regular and structured elements of a processor, such as SRAM memories and register-files were the major point of concern with regard to soft-errors in the past; thus effective EDAC codes have been developed in order to decrease the vulnerability of these structures against soft-errors. In contrast, developing low-overhead mitigation methods for unstructured and irregular parts of a processor such as the control unit is still an open question [Gha08]. This problem is escalating with advancements in processor structures as the amount of chip area devoted to complex structures will grow with chip complexity. On the other hand, traditional hardware redundancy-based approaches exploiting m-way replication of complex structures of a processor are no longer viable as they impose an unacceptable overhead on the entire system. The scope of this chapter is about the impact of SETs in the unstructured parts of a functional unit.

The next subsection quickly surveys some state-of-the-art mitigation methods which are being used in high-performance processors. Having mentioned the existing methods and their limitations, the contribution of our developed mechanisms will be discussed in subsection 5.1.2. Section 5.2 describes the details of our masking mechanism in local control units while section 5.3 deals with the recovery mechanism in combinatorial logic.

5.1.1 State-of-the-art

One of the most well-known approaches to eliminate the impact of soft-errors in modern processors is the Checkpoint and Recovery (CR) method [Akk03] in which the current state of the processor is saved in a memory device at various points in the execution of the program code (referred to as check-points). If a soft-error is detected, the processor status will be re-loaded with the last check-point (this reloading process is referred to as roll-back) . The program execution is resumed once the status of the processor is restored from the latest check-point. Generally, CR-based methods impose a heavy load on the system, as the whole status of the processor needs to be stored and reloaded at specific intervals. Many parameters are involved in when and how a roll-back needs to be triggered. Thus, different versions of CR-based methods have been proposed in the literature.

Wang et al. [Wan06] proposed the well-known ReStore architecture, in which the activation of a rollback is triggered by some symptoms which alert the presence of a soft-error, such as control flow miss-speculations or a high number of cache misses within the normal flow of a program. Ghasemzadeh-Mohammadi et al. [Gha08] presented a signature-based error detection and rollback recovery technique for the control logic of MIPS processors. As specific works on combinatorial logic, Chen et al. [Che06] reconfigured the redundancy of functional units of a DSP processor as an M-way replication architecture to mitigate soft-errors. Even though this method could considerably diminish the impact of soft-errors, the execution time of a program will increase significantly (up to three times). This is due to assigning some functional units to the fault-mitigation mechanisms. Recently, a hardware/software CR-based scheme, called Reli, has been proposed in [Tli12] which is based on enhancing micro-instructions with additional micro-operations to facilitate check-pointing. However, it suffers from a common issue in all CR-based methods, which is a long recovery time (16 clock cycles in the case of Reli).

Another category of recovery methods is based on employing redundancy techniques to achieve fault-tolerance. In general, the detection latency of these methods is negligible (less than one clock cycle) but the imposed overhead on the area, or power might be significant.

Gaisler et al. [Gai02] employed EDAC codes along with Triple Modular Redundancy (TMR) to provide a spatial redundancy in the combinational logic of a Scalable Processor Architecture (SPARC) processor. The main drawback of their methods is a high degradation in the performance. Cota et al. [Cot01] explored the usage of a special finite-state machine-based controller that uses the Hamming code to correct SEUs in the control unit of MIPS processors. Kim et al. [Kim01] and Ganesh et al. [Gan06] explored a signature-based caching scheme to mitigate soft-errors during run-time. In their methods, all the control signals used in each pipeline stage are integrated into a signature which is subsequently verified before the commitment stage. A potential drawback of this method is that data dependency can stall the pipeline stages for a very long time.

As a software mechanism, Bolchini [Bol03] has proposed a software methodology for detecting hardware faults, while Chen et al. [Che10] proposed a reliable data path using Duplication And Comparison (DAC) along with the TMR method.

VOLTaiRE, is a low cost fault detection solution tailored for DSP processors that has been proposed by Shyam et al. in [Shy06]. Their method deals with detection of faults in the data path of DSP processors. Two soft-error mitigation schemes, being Soft-Error Mitigation (SEM) and Soft and Timing Error Mitigation (STEM), use the approach of multiple clocking of data for protecting combinational logic from soft-errors; they have been recently proposed in [Avi12]. While both of those methods can detect nearly 100 percent of soft-errors, they unfortunately impose a 100 percent deterioration in the speed of the processor.

In this chapter, a mitigation method for the control logic and a recovery method for the combinatorial logic of DSP processors will be developed. It will be shown that by exploiting the characteristics of DSP workloads and DSP architectures, our method can benefit from the advantages of redundancy-based methods (very short detection latency) and CR-based methods (low overhead in area/power/performance).

5.1.2 Our DSP mitigation techniques

This chapter proposes a new architecture for DSP processors by developing two architectural mechanisms to mitigate SEUs and SETs in functional units of a DSP processor. These mitigation mechanisms have been developed based on exploiting the unique characteristics of DSP workloads as well as DSP architectures. This section gives an overall view of these two methods and following sections provide the details of each mechanism.

As depicted in Figure 5.1, a DSP processor is consisting of several functional units that execute a Very-Long-Instruction-Word (VLIW) instruction in parallel. Each functional unit is composed of several input registers, a Local Control Unit (LCU) and combinational logic. Considering the fact that input registers can be protected by readily available EDAC codes, a soft-error masking method has been developed for the LCUs, and a SET recovery mechanism has been designed for the combinatorial part of each functional unit.

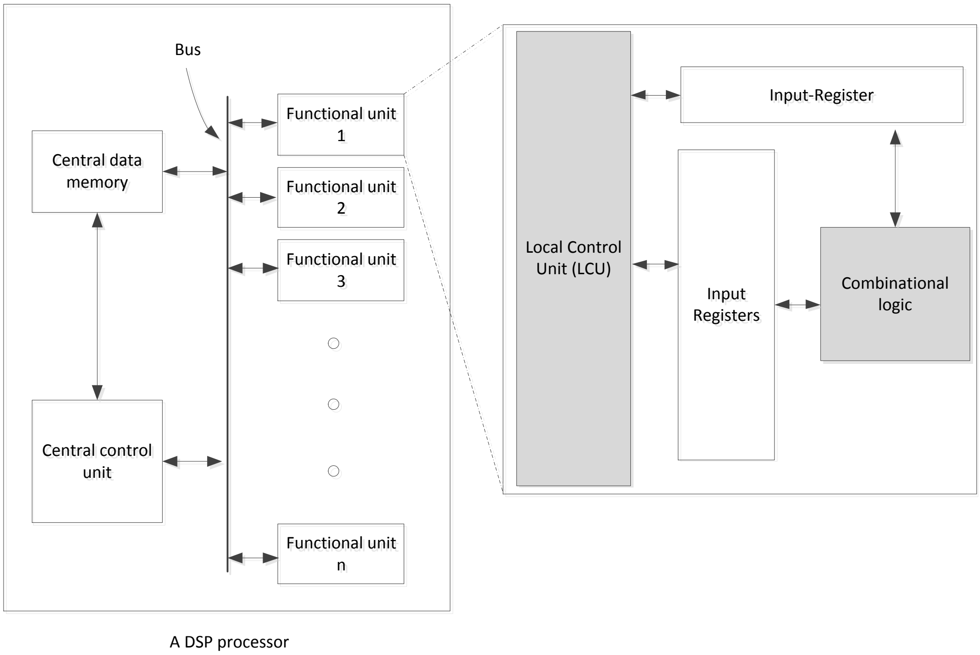


Figure 5.1. A typical architecture of a DSP processor. The grey parts will be enhanced in this chapter.

LCUs are responsible for generating control signals based on the fetched opcode. In order to protect the control-signals produced by each LCU, the control signals have been classified into either opcode-dependent or instruction-dependent control signals, based on their changeability over time during the execution of an instruction. To avoid a momentary change in the value of these opcode-dependent control signals during an execution of an opcode, they have been replaced by a Read-Only Memory (ROM) memory. This ROM memory acts as a Look-Up Table (LUT).

To protect instruction-dependent control signals, an inherent characteristic of DSP workloads, the locality of references [Hen11], has been employed. The details of these proposed architectures will be explained later. Experimental results show that the percentage of failures drops from 40% to 5.4%, while they impose a 4% increase in silicon area and a 10% deterioration in speed.

In the second approach, the combinational logic inside each functional unit has been enriched with shadow registers which enables re-execution of very fine grained part of an instruction while the rest of the processor is waiting, the so-called freezing. Considering that the duration of soft-errors in a modern digital ICs is less than one clock cycle [Ale11], this freezing period can mitigate the impact of soft-errors. Experimental results show that this recovery mechanism in the combinatorial logic part imposes a 10% increase in silicon-area and no degradation in speed, while the percentage of induced failures drops from 30% to 15%.

The above mentioned recovery mechanism in the combinational logic has several advantages, such as:

a. There is no need to store (checkpoint) or reload (rollback) the whole status of the processor during a recovery, as the recovery mechanism performs a very fine-grained local re-execution.

b. Freezing the healthy functional units of a DSP processor for only one clock cycle while the erroneous part of the instruction is being re-executed. This mechanism can employ the existence of the 'wait' signal in a processor to freeze the healthy parts of the processor;

therefore the central-control logic of the DSP processor does not need to be modified.

c. Storing the minimum amount of information as back-up data in each functional unit considerably decreases the reloading overhead during the recovery time. As mentioned before, our recovery method needs only one clock cycle to recover from soft-errors, while one of the fast versions of the check-point and recovery technique [Wan06] requires 16 clock cycles to recover from a soft-error.

d. This recovery mechanism imposes a negligible area overhead in the processor and there is no penalty in the performance.

5.2 Our SET masking mechanism in LCUs

The mechanism of masking SETs in the LCUs is based on classifying the control signals of each functional unit, generated by LCUs, to either opcode-dependent or instruction-dependent control signals. Figure 5.2 shows the concept of opcode-dependent and instruction-dependent control signals.

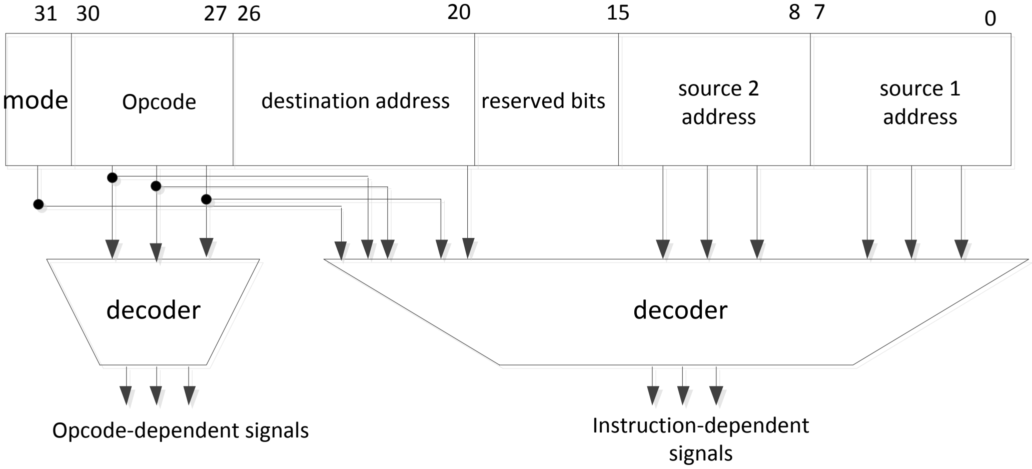


Figure 5.2. Opcode- and instruction-dependent control signals.

As can be seen in this figure, the value of opcode-dependent control signals depends only on the opcode part of an instruction. For example as long as the opcode part of an instruction is 'add', the value of the 'add-signal' is 1, irrespective of the address part. In contrast, the value of instruction-dependent control signals depends on the whole instruction and not only on the opcode part. For example, the 'write-address' of a register depends on the address part of an instruction as well as the opcode part. The following subsections present two different masking mechanisms for each category.

5.2.1 Opcode-dependent control signals

Since the value of an opcode-dependent control signal depends only on the opcode part of an instruction, and the number of possible opcodes per functional unit is limited, a distributed ROM memory has been used to store the value of the opcode-dependent control signals for each opcode. The term distributed implies that each execution unit can access this ROM unit. A limited number of different opcodes per functional unit (32 different opcodes per functional unit in our case study) along with a limited number of opcode-dependent control signals (16 different signals in our case study) make it feasible to store the value of these control signals for each opcode in a ROM memory during the design phase and then retrieve them during run time. The organization of this ROM memory is depicted in Figure 5.3 and consists of several entries (equal to the number of different opcodes per functional unit) and the expected value of their control signals as the contents. In order to retrieve the value of a particular control signal during the run-time, the opcode of a fetched instruction is converted into an input address for the ROM memory in which the expected value of a particular control signal has already been stored (e.g. 'ALU-selection').

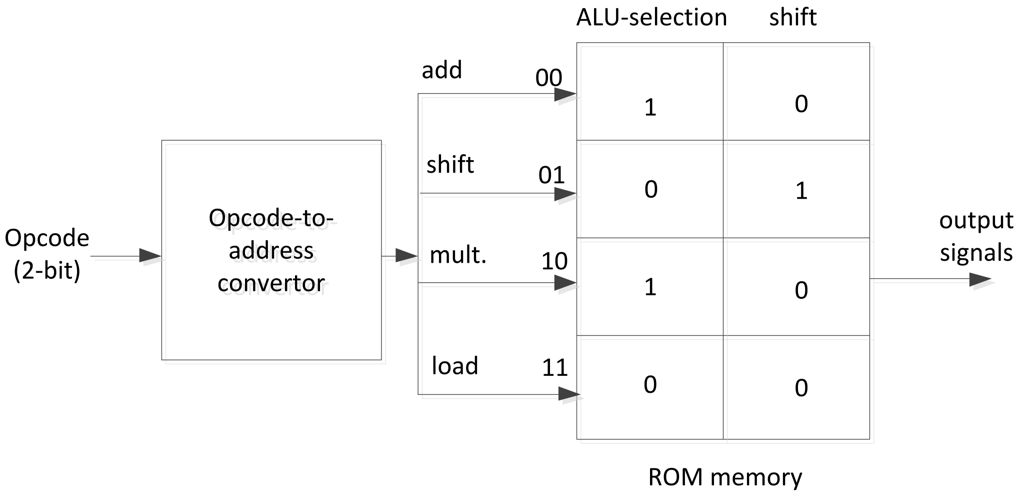


Figure 5.3. ROM structure to mask soft-errors in the opcode-dependent control signals.

Suppose that a typical functional unit has four different opcodes (add, shift, multiply and load). The opcodes are recognized by two-bit binary numbers, add=00, shift=01, multiply=10, load=11. This functional unit has also two control signals of which their value depends on the opcodes, named 'ALU-selection' and 'shift' . The designer knows the value of these two control signals for each opcode during the design phase of a processor, so the value of these control signals per opcode can be stored during the design phase. For example, 'ALU-selection' is 1 for the add and multiply opcodes and 0 for the other two opcodes. The 'shift' signal is only 1 during the execution of the shift operation. Consequently, the ROM structure has four entries (associated with four opcodes) while each entry has a two-bit width representing the contents. The content of this ROM memory is constant and independent of the executed workload.

The probability that a SEU or SET can change the contents of a ROM memory is very low (near 0%) as compared to the traditional unstructured organization of local control units [Esa11]. Moreover, EDAC codes can be readily used to protect this ROM memory [Wen96] since it is a regular structure, i.e. there is at least one clock latency between when a value is written and when it is being read. However, the input/output lines or the opcode-to-address convertor are still vulnerable to SETs. In the experimental results, the efficiency of this method will be assessed.

It is important to mention that the value of each opcode-dependent control signal will be generated by this ROM-unit in a look-up table manner.

5.2.2 Instruction-dependent control signals

Another category of control signals is the instruction-dependent control signal. Since the number of different instructions per functional unit is infinite, the previously introduced look-up table is not feasible in this case. In order to propose a novel soft-error masking mitigation method, a common principle in computer architectures, the so-called locality of reference [Hen11] has been employed. This concept implies that most of the program execution time is spent on a small piece of code. Especially for DSP workloads, about 90% of the computational time is spent in a very small kernel [Smi07]. As a result, the variety of instructions per workload is limited; however, the exact instructions are not known to the designer at design-time.

Our idea that has been used here is to store a history of an instruction-dependent control signal during the first and second execution of an instruction and then subsequently compare the succeeding generated run-time values with the ones stored as a history of the signal to detect any momentary change. Considering that the values of an instruction-dependent control signal are identical for all the executions of the same instruction, unless an error occurred, this mechanism can detect any singular errors in these signals.

In order to implement the previously mentioned idea, a cache structure has been used. This cache architecture has several entries which are associated with the number of different instructions within the kernel of the DSP program. The higher number of entries in the cache, the more different instructions can be tracked. To track each instruction, the unique Program-Counter (PC) can be used.

The structure of this cache has been depicted in Figure 5.4. Suppose that N different signals have been classified as instruction-dependent control signals. The 'PC-to-cache-address-decoder' assigns a unique address in the cache entries to each instruction. The values of the control signals, which have been produced by the normal control unit during the run-time (Figure 5.1), are saved in the cache memory. This occurs during the first and second execution of the kernel of the DSP program. From the third execution onwards, the run-time value of a signal (which has been generated by the conventional LCUs) will be compared with two previously stored instances. The final output is the result of a majority vote of these three values. Considering that the likelihood of an identical perturbation of two or three instances of one signal is very small, this scheme can mask the effects of soft-errors for the associated signals.

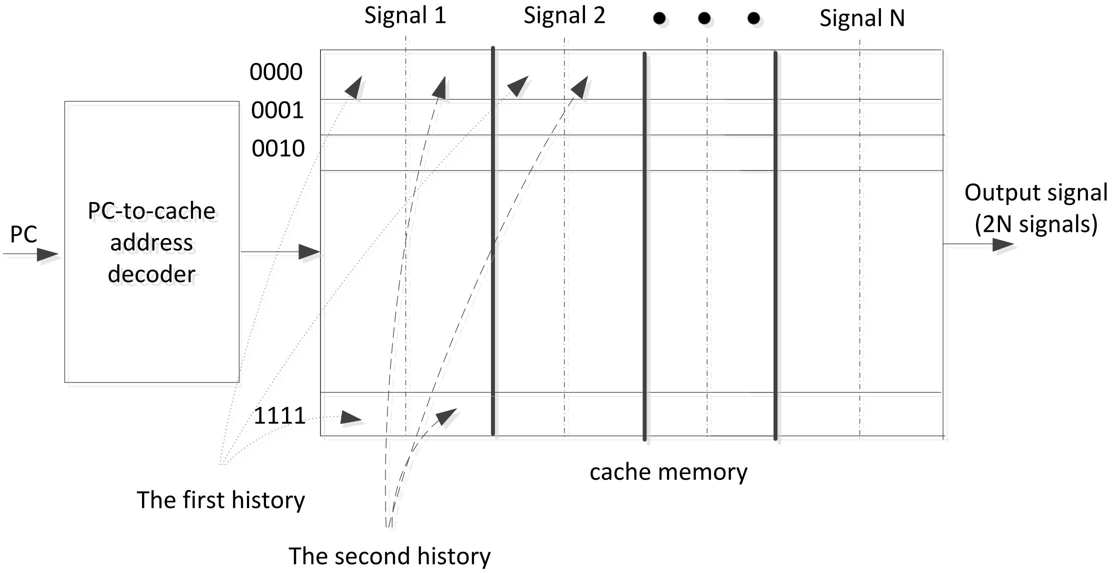


Figure 5.4. Cache structure to store a history of control signals.

The replacement mechanism of this cache structure plays an important role in the efficiency of our mechanism. The random replacement policy, which randomly selects a candidate for being discarded from the cache, was used here to simplify the implementation. Moreover, the number of entries of each cache memory has been limited to 16, i.e. 16 different instructions can be tracked at any given point of time. A larger cache can protect more signals, however as a trade-off, the complexity of the 'PC-to-cache-address-decoder' and the area overhead of the cache structure need to be considered as well. Another issue that needs to be addressed here is the controller (or FSM) which is responsible for determining the status of an instruction-dependent control signal with regard to its history. Figure 5.5 depicts the state diagram of this FSM.

The state diagram shown in Figure 5.5 takes the signal 'active' as input. In this figure, the output signal of each label is indicated by underlining (for example output) while the normal letters shows the label of a status (for example no-history). This signal is unique for each instruction and means that the program flow has reached this instruction. 'Active$_j$ = 1' means that the program flow for instruction 'j' has been executed for the first time. The 'output' signal indicates whether three executions of instruction 'j' have occurred in the program flow or not. So 'output = 0' means less than thee executions of a particular instruction occurred while 'output = 1' indicates three or more executions of an instruction have taken place. As can be observed from the FSM machine, the third execution of instruction 'j' causes the FSM machine to reach the status 'three-history'. In this state ('three-history') the 'output' signal is always high because at least three executions of instruction 'j' have occurred before.

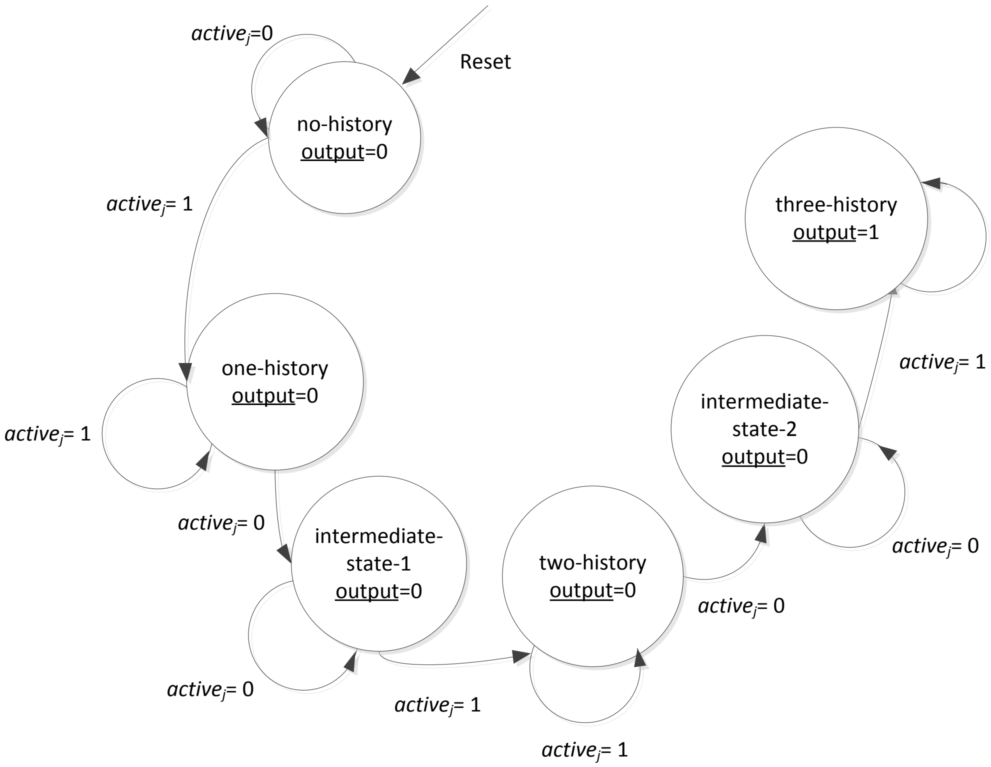


Figure 5.5. The state diagram of the FSM in the cache structure.

Figure 5.6 shows the complete scheme. This figure will be explained by following a simple pseudo-code which has been depicted in Figure 5.7.

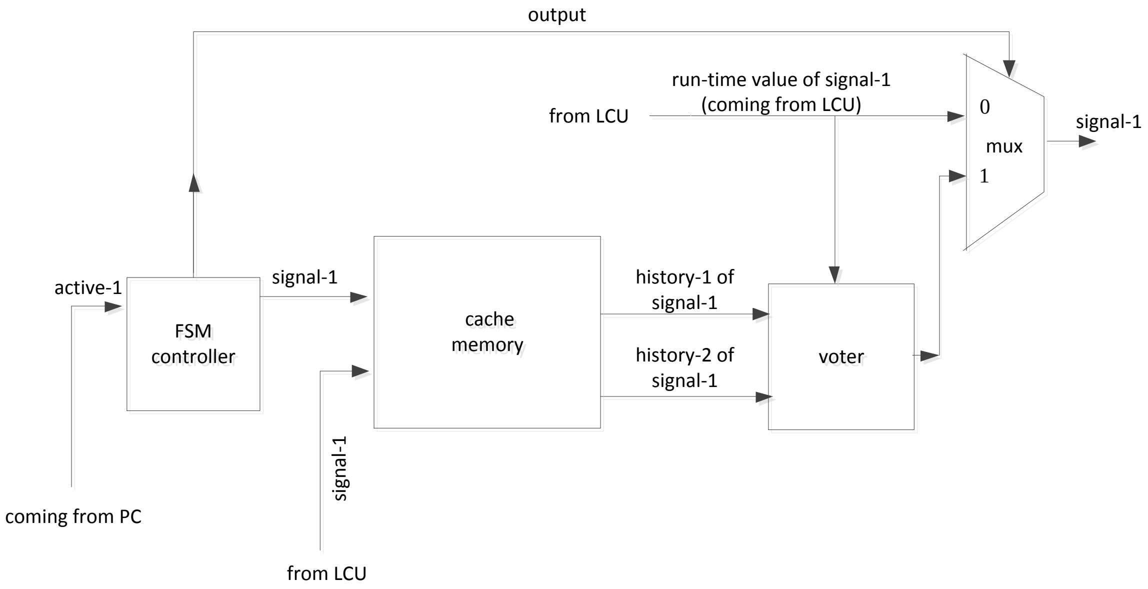


Figure 5.6. The complete scheme composed of a cache memory, majority voter and the FSM controller.

Suppose that a control signal 'i' has come from the LCU and its value depends on the whole instruction. This signal is also enhanced during the execution of a particular instruction, named 'instruction-1' in the pseudo-code of Figure 5.7.

PC	Instruction
0XXXX	beginning of the program
0X0011	Loop L1
0X0100	instruction-1 (the 'active-1' signal is 1)
0X0101	The remaining of the loop L1
0X0110	end loop
0XXXX	rest of the program

Figure 5.7. A pseudo-code consisting of one loop.

Prior to the first iteration of the loop, there is no history of the 'i' signal. Therefore the FSM of Figure 5.5 will be in the status labelled by 'no-history' in which the 'output' signal is zero. This means that the current value of 'i' is written in the cache memory of Figure 5.6 and more importantly it will be passed by the multiplexer to the rest of the system.

During the first iteration of the loop (first execution of 'instruction-1') the activation signal named 'active-1' is made one via the PC which indicates that 'instruction-1' has been reached the program flow. As a result the FSM will move on to the status labelled 'one-history' which means one history of signal 'i' resides in the cache. At this stage, the value of 'output' signal of FSM machine is still zero and hence the value which is passed to the system by the multiplexer of Figure 5.6 will not yet come from the cache.

During the second execution of 'instruction-1', the value of 'signal-i' signal will be stored as the second history of this signal in the cache. The FSM machine will be in the status labelled 'two-history' and the value of the 'i' signal will be passed from the LCU to the system (as the 'output' signal is still 0).

During the third iteration of the loop, the 'active-1' signal will become one for the third time. Now the FSM machine will move to the status labeled 'three-history' and the value of 'output' signal will be '1'. At this point the multiplexer in Figure 5.6 will assign the value that results from the majority voter to the control signal.

Even if one instance is corrupted in the above mentioned scenario, being either one of the stored values of the cache or the value of the LCUs, the faulty value will be masked by the majority voter and the fault-free signal will traverse through the system.

It is important to note that the structure of Figure 5.6 is a redundant module along with the conventional LCUs, i.e. this structure will not replace the LCUs, but work as a redundant unit along with the LCUs to enhance redundancy. It is also worth mentioning that many readily available mechanisms to harden the cache memory with regard to soft-errors can be used in the mentioned structure in order to make the cache structure of Figure 5.6 more resilient [Zar03].

5.3 A recovery mechanism in combinational logic

Another concern in a functional unit of a DSP processor, apart from the unstructured LCUs, is the combinatorial logic. This is because each functional unit receives its associated opcode from the program memory and the required data from the data memory. The opcode of the received instruction is decoded by the LCU (or equivalent logic, such as a look-up table, as indicated in section 5.2) and subsequently the decoded signals will be stored in associated input registers. Similarly, the required operands from the memory or register-file will be fetched and stored in their associated input registers. As long as the data presented in the input registers are identical at time instances T_1 and T_2, the output signal of the combinational logic at time instances T_1 and T_2 will be identical.

The idea of our recovery method is based on accompanying every input register with one shadow register in order to hold a copy of the associated data during one consecutive clock cycle. Since every instruction in a VLIW architecture is distributed over different functional units, it is feasible to halt the fault-free functional units and re-execute the faulty operating in that specific functional unit.

To achieve this goal, both the decoded signal received from the LCU and the data received from the data memory need to be available for one extra clock cycle. Upon error detection, the normal flow of the processor will be halted, and the stored data will be sent to the combinational logic one more time, resulting in a one clock cycle latency on the overall execution time. The limitation of this method is that if two SETs occur at two consecutive clock cycles, the proposed mechanism will fail to recover the processor. Even though the probability of such an occurrence is very rare, adding more shadow registers per input register can solve this problem. However, only one shadow register per input register will be used here.

There are two possibilities to implement this mechanism:

The first is to store the value of an input register at the 'ith' clock cycle, denoted at 'data$_i$' in a shadow register; then upon error detection, pass the 'data$_i$' to the combinational logic at the '(i+1)th' clock cycle and simultaneously store the new arrived value 'data$_{i+1}$' (which was supposed to be applied to the combinational logic) in the input registers.

The second implementation is to re-fetch the 'data$_i$' at the '(i+1)th' clock cycle and store the 'data$_{i+1}$' in the shadow register. The second implementation has been selected in this work because the implementation is simpler. This mechanism explained above is shown Figure 5.8.

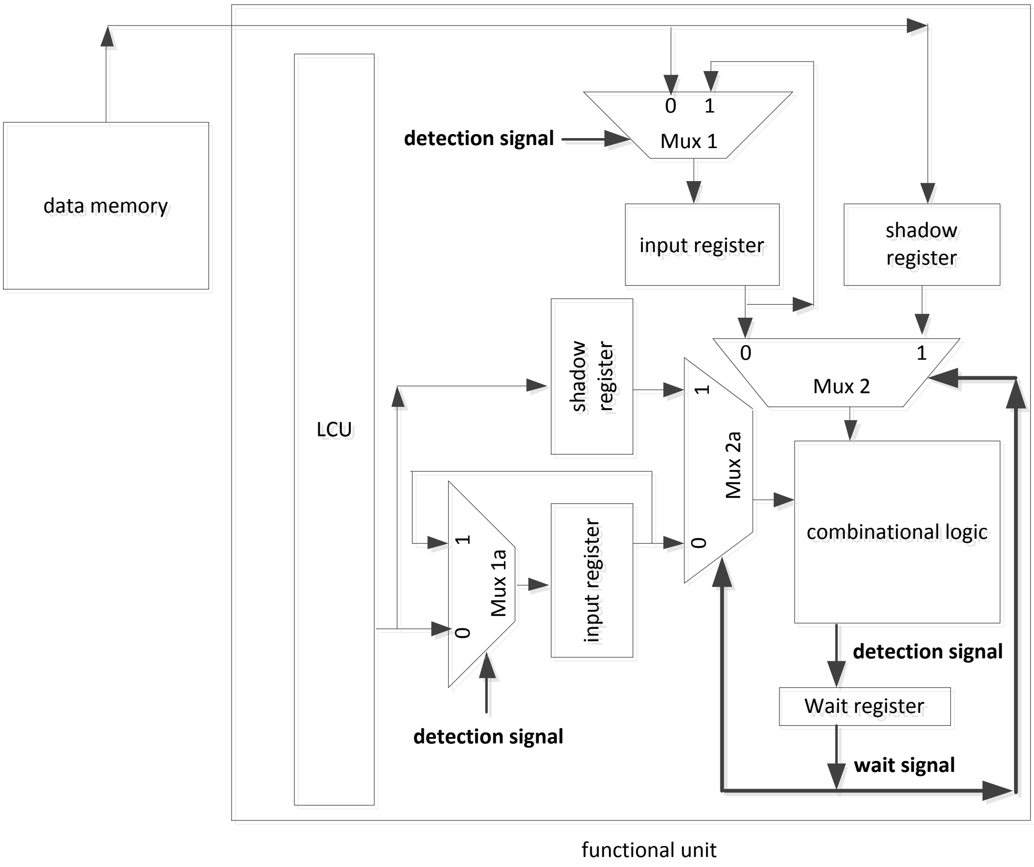


Figure 5.8. The recovery method in the combinational logic part.

Referring to this figure, a detection signal will be generated by the combinational logic (shown with bold letters in Figure 5.8). This signal can be generated by any mechanism, providing that it detects an error in less than one clock cycle (a so-called zero latency), such as the Duplication With Comparison (DWC) approach [Hen11]. This detection signal will set a wait register that will raise the wait signal during the next consecutive clock cycle (also indicated by bold letters) to halt the fault-free functional units.

The timing diagram of the error detection and correction scheme is depicted in Figure 5.9.

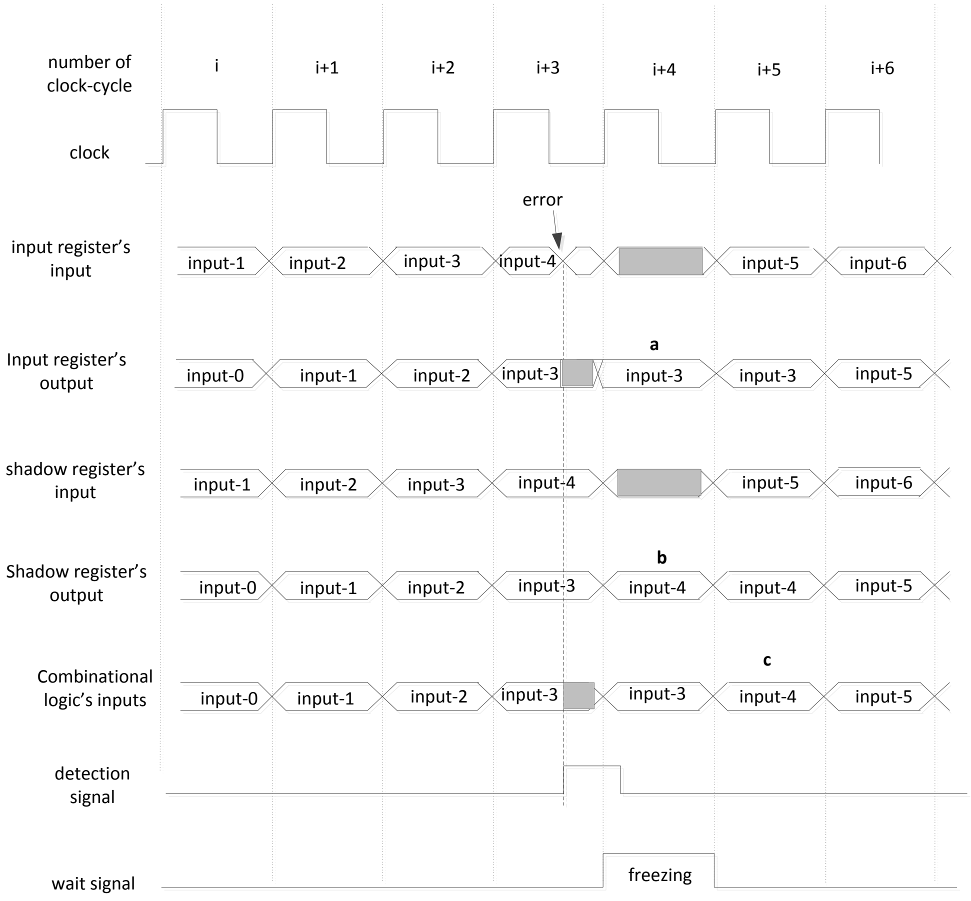


Figure 5.9. Timing diagram of the recovery mechanism in a functional unit (the grey parts are invalid data).

As can be seen in this figure, after error detection during the '(i+3)th' clock cycle (denoted by 'error'), the input register will be loaded during the next clock cycle ('(i+4)th' clock cycle) with previous data (labelled as input-3) instead of a new data (this status is indicated by letter **a** in the timing diagram). New data labelled as 'input-4' is temporary saved in the shadow register (this status is indicated by letter **b** in the timing diagram). This will be passed to the combinatorial logic during the '(i+5)th' clock cycle (this status is indicated by letter **c** in the timing diagram).

The multiplexers 1 and 1a in Figure 5.8 provide the possibility of loading either a value from the data memory or the value of the last clock cycle, depending on the value of the detection signal. The multiplexers 2 and 2a provide the possibility of loading the output of an input register or a shadow register into the combinatorial logic, depending on the value of the wait-signal.

Referring to Figure 5.9, during the '(i+3)th' clock cycle, the detection-signal becomes high, while at the beginning of the '(i+4)th' clock cycle, the detection-signal is still high and each input register will be loaded by its previous value. During the '(i+4)th' clock cycle, 'input-3' will be processed again in the combinatorial logic. At the beginning of '(i+5)th' clock cycle, the wait-signal is still high and the combinational logic will be loaded by the contents of the shadow registers.

It is worth mentioning that considering the experiments carried out by iRoC-Technologies [Iro12] and [Ale11], the duration of a SET is considerably less than one clock cycle. So, simultaneously re-executing the faulty instruction after a clock cycle will stop the faulty results to propagate through the rest of the processor.

The main novel feature of the presented recovery method is the isolation of the faulty functional unit from the fault-free ones for one clock cycle, referred to as freezing, and simultaneously re-executing the faulty part of the instruction. Another novel feature is that a minimum amount of information needs to be stored in each functional unit; this mechanism decreases the recovery timing overhead to only one clock cycle, while a typical recovery mechanism takes 16 clock cycles for the CR-based mechanism [Tli12]. Moreover, the speed of the enriched processor is identical to the performance of the original processor, as long as no SET is present in the system. Furthermore, several clock cycles are required to store a check-point in the conventional CR-based methods irrespective of the occurrence rate of SETs. Our presented method stores the value of every input register simultaneously in a shadow register; therefore as long as no error has been detected by the detection mechanism, the total execution time of a workload is identical in both cases of the original as well as the enriched processor.

5.4 Experimental results

In this section, our results are presented based on the implementation of the described methods in a DSP Xentium processor from Recore System [Rec11]. The details of the Xentium processor have been already discussed in chapter 2.

The RTL code of the Xentium processor has been modified in such a way that the LCUs of functional units are enhanced by the method presented in Section 5.2 and the combinatorial parts of the functional units have been modified based on the mechanism presented in the section 5.3.

5.4.1 Area overhead and performance degradation

To assess the area overhead and performance degradation induced by the presented methods, a fault-tolerant version of the DSP Xentium processor was developed using the RTL VHDL code.

First, twenty different DSP workloads were executed on the enhanced version of this processor to assure the correct functionality of the modified processor. Subsequently, the synthesis tool Synopsys Design Compiler was used to synthesize the RTL design using the UMC 90nm technology node. The implementation data has been divided into two parts: the area/timing overhead induced by the LCU-related methods (first approach) and the overhead imposed by combinatorial logic–based methods (second approach). By doing so we were able to compare the efficiency of our method with its counter parts available in literature.

The achieved results are shown in Table 5.1. The ‘original Xentium’ label stands for the original implementation of the Xentium processor, without any modification. The ‘FT LCU’ is a Xentium processor in which the LCUs of all its functional units have been enhanced by the method of section 5.3, while the remainder of the hardware in the functional units is identical to the original design. The next two rows (FT. combinational logic including detection and excluding detection) show a Xentium processor in which the combinational logic has been modified on the basis of the method presented in section 5.3. Moreover, the reported area for the combinational logic has been divided into two parts: including the detection mechanism and disregarding the detection mechanism. As detection mechanism for combinational logic, a DWC approach [Gos08] has been employed, as shown in Figure 5.10. For example, for the 32*32-bit multipliers inside the M functional unit [Figure 2.7 of Chapter 2], two 8-Least Significant Bits (LSB) of each input are concurrently multiplied by a smaller redundant multiplier and then the calculated result will be compared to the 8-LSB of the 32*32 bit multiplier, in which any mismatch indicates an error. However, this partial comparison suffers from an inability of discovering an error in the high significant bits of an input. It is important to mention that this simple partial comparison has been employed here just as an example of the detection of soft-errors in a functional unit.

Table 5.1. The area overhead and performance deterioration for the proposed approaches.

	total cell area (μ^2)	area overhead (%)	critical-path (ns)	speed deterioration (%)
original Xentium	293462	0	7.87	0
FT. LCU	304785	4	8.70	10
FT. combinational logic (including detection)	341316	16	7.87	0
FT. combinational logic (excluding detection)	325247	10	7.87	0

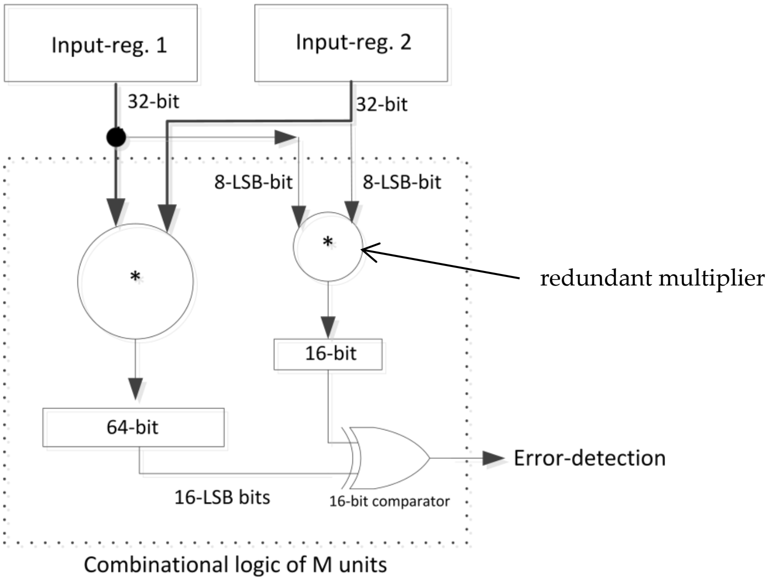


Figure 5.10. A possible detection mechanism (partial DWC).

5.4.2 SET sensitivity

A simulation-based fault study at gate-level implementation has been conducted to assess the achieved fault tolerance of the enriched processor. For the simulation model of SETs, the most recent model of SETs presented in chapter four has been employed. During the fault-injection process, a digital signal-processing program, the well-known Finite Impulse Response (FIR) has been used as a workload. Using additional DSP workloads would have been desirable, however, the computational time to conduct FIR experiments was already more than several days and involving more workloads was not feasible at this time.

The induced effect of a fault can be classified as wrong-results, which means the injected fault has been propagated into the system while correct-behaviour indicates the injected fault has been masked before propagating into the system [Muk08]. The behaviour of the processor has been indicated in terms of a percentage, for example if 10 out of 100 fault-injections produce wrong-results, the sensitivity of the processor is 10%.

The number of fault-injections in each set of experiments has been increased from an initial value (200 for LCU and 500 for the combinatorial logic) until a clear convergence could be recognized in the obtained sensitivity level of the processor. The mathematical details of calculating the convergence point is exactly the same as discussed in chapter four.

Table 5.2 shows the results of the sensitivity analysis. It can be seen that for the original processor, the percentage of propagated faults in the LCUs is 40% while this number is 30% for the combinational logic (these two numbers are indicated by underlining in Table 5.2). The sensitivity of the enriched LCUs (the row labelled with FT LCU) has decreased to 5.4% with a detection-latency of 0 clock cycles (indicated by **bold** letters in Table 5.2) which means faults will be masked. Further investigation showed that undetected faults have escaped from the detection mechanism as they occurred in the 'opcode-to-address-convertor' in the look-up table scheme. For fault-tolerant combinational logic (labelled as FT. combinational logic), 15% of injected faults could escape from the detection mechanism (is indicated by **bold** letters in Table 5.2). However, as long as a fault is detected, the recovery mechanism can stop the fault from propagating through the rest of the processor and the processor is recovered within one clock cycle.

Table 5.2. SET sensitivity in the enriched Xentium processor.

	# fault-injections		**# wrong answers**		**sensitivity (%)**		**detection latency (# clk. cycles)**
	LCU	**comb.**	**LCU**	**comb.**	**LCU**	**comb.**	
original Xentium	12800	32000	5120	9600	40	30	N.A
FT. LCU	12800	32000	700	9600	**5.4**	30	0
FT. comb.*	12800	32000	5120	5100	40	**15**	1

*comb. stands for combinational logic

5.4.3 Comparison of our methods with other available methods

Table 5.3 shows the comparison of our proposed methods with some available solutions of soft-error mitigation in either LCU or combinational logic. A thorough comparison is not feasible as the imposed overhead depends on many parameters such as the exact architecture of the case study, the workloads, etc.

Starting with the mitigation methods in the control unit, our work has been compared with [Gha08]. It can be seen that even though the area-overhead of [Gha08] is similar to ours (3.4% as compared to 4%), our method will cause less performance deterioration (10% as compared to 17%).

Comparing our combinational logic mitigation method with the one presented in [Che06], our method is quite competitive in terms of area overhead (10% as compared to 17%) while both methods cause no performance deterioration.

Table 5.3. Comparison of our method versus other known methods

penalties / method	area overhead (%)		speed deterioration (%)	
	LCU	combinational logic	LCU	combinational logic
our method	4	10	10	0
[Gha08]	3.4	N.A	17	N.A
[Che06]	N.A	15	N.A	0

5.5 Conclusions

DSP processors are emerging more and more in in domains, such as automotive applications, where low cost and dependability are primary concerns. As traditional hardware redundancy methods are usually not affordable for current-day applications, new solutions for modern processors with regard to the ever-increasing threat of soft-errors have to be developed. In this chapter, two novel solutions to mitigate soft-errors in DSP processors were introduced.

The first methods use the limited number of opcode-dependent control signals to construct a look-up table to retrieve the value of each signal. Moreover, they have used the high degree of locality of reference in DSP kernels to organize a cache memory structure to mitigate soft-errors in the control unit of a DSP processor. These methods could decrease the sensitivity of control unit of a Xentium processor from 40% to 5.4% with regard to SETs while imposing 4% on silicon area and 10% deterioration on the speed of the processor.

The second approach targets the combinatorial logic and benefits from the inherent architecture of DSP processors to be able to isolate faulty functional units from the fault-free ones in order to carry out a fast recovery. Our simulation results showed that the proposed methods are able to reduce the vulnerability of a DSP functional unit from 30% to 15% with regard to soft-errors, while the area overhead and performance deterioration imposed on the system are 16% and zero, respectively.

References

[Akk03] H. Akkary, R. Rajwar, S. T. Srinivasan, “Checkpoint processing and recovery: towards scalable large instruction window processors,” in International Symposium on Microarchitecture (MICRO), pp. 20-32, 2006.

[Ale11] D. Alexandrescu, E. Costenaro, M. Nicolaidis, “A practical approach to single event transients analysis for highly complex designs,” in IEEE Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), pp. 155 - 163, 2011.

[Avi12] N. D. P. Avirneni, A. K. Somani, “Low overhead soft-error mitigation techniques for high-performance and aggressive designs,” in IEEE Transactions on Computers, Vol. 61, No. 4, pp. 488 – 501, 2012.

[Bol03] C. Bolchini, “A software methodology for detecting hardware faults in VLIW data paths,” in IEEE Transactions on Reliability, Vol. 52, No. 4, pp. 458 – 468, 2003.

[Che06] Y. Y. Chen, K. L. Leu, C. S. Yeh, “Fault-tolerant VLIW processor design and error coverage analysis,” in International Conference of Embedded and Ubiquitous Computing, pp. 754 - 765, 2006.

[Che10] Y. Y. Chen, K. L. Leu, “Reliable data path design of VLIW processor cores with comprehensive error-coverage assessment,” in Journal of Microprocessors and Microsystems, Vol. 34, No. 1, pp. 49 – 61, 2010.

[Cot01] E. F. Cota, F. Lima, S. Rezgui et al., “Synthesis of an 8051-like microcontroller tolerant to transient faults,” in Journal of Electronic Testing: Theory and Applications (JETTA), Vol. 17, No. 2, pp. 149 – 161, 2001.

[Esa11] Radiation Effects Mitigation handbook, European Space Agency (ESA) handbook, 2011.

[Gai02] J. Gaisler, “A portable and fault-tolerant microprocessor based on the sparc V8 architecture,” in International Conference on Dependable Systems and Networks, pp. 409 – 415, 2002.

[Gan06] T. S. Ganesh, V. Subramanian, A. Somani, “SEU mitigation techniques for microprocessor control logic,” in European Dependable Computing Conference (EDCC), pp. 77 – 86, 2006.

[Gha08] H. Ghasemzadeh-Mohammadi, H. Tabkhi, S. G. Miremadi et al., “A cost-effective error detection and roll-back recovery technique for embedded microprocessor control logic,” in International Conference on Microelectronics, pp. 470 – 473, 2008.

[Gos08] M. Gossel, V. Ocheretny, E. Sogomonyan et al., “New methods of concurrent checking,” Springer, ISBN 978-1-4020-8420-1, 2008.

[Hen11] L. L. Hennessy, D. A. Patterson, “Computer architecture, fifth edition: a quantitative approach,” the Morgan Kaufmann Series in Computer Architecture and Design, ISBN 9780123838735, 2011.

[Iro12] iRoC Technologies, http://www.iroctech.com, 2012.

[Kim01] S. Kim, A. K. Somani, “On-line integrity monitoring of microprocessor control logic,” in International Conference on Computer Design, pp. 314 – 319, 2001.

[Mie212] A. Miele, C. Sandionigi, M. Ottavi et al., “High-reliability fault-tolerant digital systems in nanometric technologies: characterization and design methodologies,” in IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), pp. 121 – 125, 2012.

[Muk08] S. Mukherjee, “Architecture design for soft-errors,” ISBN 978-0-12-369529-1, Morgan Kaufmann Publishers, 2008.

[Rec11] Recore Systems, www.recoresystems.com, 2011.

[Sch04] L. Schiano, M. Ottavi, F. Lombardi, “Markov models of fault-tolerant memory systems under SEU,” in International Workshop on Memory Technology, Design and Testing, pp. 38 – 43, 2004.

[Shy06] S. Shyam, S. Phadke, B. Lui et al., “Voltaire: Low-cost fault detection solutions for VLIW microprocessors,” in Workshop on Introspective Architecture, pp. 20 – 27, 2006.

[Smi07] G. J. Smit, A. Kokkeler, P. T. Wolkotte et al., “The chameleon architecture for streaming DSP applications,” in European Association for Signal Processing Journal (EURASIP) on Embedded Systems, pp. 11, 2007.

[Tli12] T. Li, R. Ragel, A. Parameswaran, “Reli: Hardware/software checkpoint and recovery scheme for embedded processors,” in Design, Automation and Test in Europe (DATE), pp. 875 – 880, 2012.

[Tou07] E. Touloupis, J. A. Flint, V. A. Chouliaras et al., “Study of the effects of SEU induced faults on a pipeline protected microprocessor,” in IEEE Transactions on Computers, Vol. 56, No. 12, pp. 1585 – 1596, 2007.

[Wan06] N. J. Wang and S. J. Patel, “Restore: symptom based soft-error detection in microprocessors,” in IEEE Transactions on Dependable and Secure Computing, Vol. 3, pp. 188 – 201, 2006.

[Wen96] X. Wendling, R. Rochet, R. Leveugle, “ROM-Based synthesis of fault-tolerant controllers,” in Workshop on Defect and Fault-Tolerance in VLSI Systems, pp. 304 – 308, 1996.

[Zar03] H. Zarandi, S. G. Miremadi, A. Ejlali, “Dependability analysis using a fault-injection tool based on synthesizability of HDL Models,” in Symposium on Defect and Fault Tolerance in VLSI Systems (DFTS), pp. 485 – 492, 2003.

CHAPTER 6

Using Multi-core Architectures to Mitigate Soft-Errors

Parts of this chapter have been submitted as a paper titled "A soft-error mitigation technique in a multicore architecture composed of DSP cores" in 16th IEEE Latin-American Test Symposium (LATS), 2015 in Mexico.

ABSTRACT- As the organization of modern computer systems is moving more and more towards multi-core architectures, the investigation of the effect of soft-errors in multi-core architectures is more important than ever before. Multi-core architectures have some unique features that can be exploited for the purpose of soft-error mitigation. For example, existence of several identical cores in a typical multi-core architecture can be very beneficiary to discover any mismatch between the internal architecture of two identical cores. This chapter extends the soft-error mitigation mechanism presented in the previous chapter (chapter 5, for single-core) to a multi-core processor environment. We will exploit the existence of similar cores in a semi-homogenous multi-core architecture to enhance the fault coverage of our mitigation method. The internal modification of each core is the same as the freezing method which has been already explained in chapter 5. However, external units to control the internal actions of each core after a soft-error detection are required. A zero-latency soft-error detection mechanism has been implemented by comparing the internal status of each functional unit of two identical cores, while both cores execute the same workloads. The status of internal functional units are compared continuously and in case of a mismatch, which is an indication of the occurrence of soft-errors, both cores re-execute the last instruction one more time to recover. Due to our zero-latency detection mechanism, re-execution of the very last instruction can recover the correct status of functional units. This internal re-execution eliminates the need of copying the entire status of one processor core to another one, which is currently used in some state-of-the-art soft-error mitigation solutions. As a result, the performance loss during the recovery time of our mechanism is much shorter as compared to similar methods. A detailed RTL model of our architecture has been designed and an excessive fault-injection campaign has been performed to evaluate the achieved soft-error coverage. The fault-injection results show that 90% of the failures caused by soft-errors can be mitigated.

6.1 Introduction

Shrinking the technology to sub-100nm technology nodes has several consequences with regard to the performance and soft-error sensitivity of modern digital integrated systems. On one hand, complex processors can be built with much faster performance; however, because there are very irregular and complex structures in the architectures of these modern designs, the soft-error sensitivity of these high-performance systems needs to be investigated accurately. Because of Moor's law, one is able to implement Integrated Circuits (ICs) with an enormous number of transistors; but since many of these transistors are packed tightly together, a huge power density and low noise margin will make the advanced digital systems very vulnerable to environmental failures. As a result, counter-measures should be considered to enhance the reliability of such a system which is manufactured in an advanced process technologies.

Another phenomenon that is pursued in implementing digital systems is that V_{th} and V_{DD} shrink in every new generation of transistors.

Since the transistors work with a very small V_{th} and V_{DD}, even very low energy particles from cosmic radiation are able to modify the behaviour of a transistor. As a result, a large variety of radiation particles are able to toggle the stored value of a flip-flop or latch or cause a glitch in the combinatorial logic. Hence, soft-error sensitivity should be considered carefully for digital systems which are implemented in sub-nanometer technologies.

Another trend is that the technology of computer systems experiences the movement from single-core architectures towards multi-core architectures. Multi-core architectures remain the main architecture of computer systems for the upcoming years because they provide a solution to continue enhancing the performance, while the increase in power consumption is still manageable. As a result, multi-core architectures have become very popular recently in the computer architecture community [Jey11], e.g. the Intel core 2 duo, Intel core i7, AMD Opteron and IBM Cell processor. However, these architectures host an enormous number of transistors, while the resilience of these systems to soft-errors is still an open question. As a result, the resilience of the whole architecture with regard to soft-errors needs to be investigated carefully. In the next paragraph, a well-documented effect of soft-errors in a multi-core architecture that has recently happened will be discussed.

Soft-errors are a threat for causing temporary damage in complex computer systems. For example in 2003, a multi-processor Sun server, called SUN flagship, experienced a temporary crash. The crash lasted for a few seconds but the server needed to be reset in order to recover again. Investigation of the logged data showed that the failure was induced by toggling the value of one of the flip-flops; it turned out it was most probably caused by the strike of neutron particles originating from cosmic radiation [Lyo00]. That crash happened only once in the life time of that Sun server; however studies show that if the technology of implementation is less than 90nm, the soft-error rate at the end-point server can reach the rate of once per 170 hours [Jey11]. With the trends of shrinking technology and reduction of power supply, advanced digital systems can experience a failure induced by soft-errors at a rate of once per 24 hours [Kay00]. This failure rate will be critical if the end-point server will be used in mission-critical applications. For example, modern electric cars use multi-core processors [Tes14] to process data which can be related to navigation, collision-avoidance and lane detection. The other obvious example is airplane computers in which most computations are highly mission-critical.

Multi-core architectures are categorized into homogeneous and heterogeneous architectures. In a homogeneous architecture, all cores (or at least the majority of cores) are identical, with regard to the internal architecture. Moreover the type of workload which is being executed in each core falls in the same category (for example DSP workloads).

The architecture and type of the cores in a heterogeneous architecture are diverse. Some architectures use a mixture of homogeneous and heterogeneous architectures; so while there are different cores in the design, some parts of the design (most frequently the majority of cores) are identical. If the majority of cores are identical, the architecture is called semi-homogeneous architecture. As will be shown later, the architecture of our target multi-core system in this chapter falls into this category.

One similarity between homogeneous and heterogeneous architectures is that they are both composed of redundant processor cores. At some instances of time, some processor cores are idle and can be used for other purposes, such as soft-error mitigation mechanisms. In this chapter, the inherent redundancy of identical cores in a semi-homogeneous architecture will be used for soft-error mitigation purposes. Our method does not strongly depend on whether the architecture is fully homogeneous or not. When two similar cores can be identified in a multi-core architecture, the proposed method of this chapter can be implemented.

6.2. State-of-the-art methods

Different methods to enhance the soft-error resilience of multi-core systems with regard to soft-errors have been already proposed at all levels of design hierarchy, including packaging level [Bau95], fabrication level [Can04], circuit design [Roc92] as well as software level [Shr10].

In chapter 5, various architectural-level methods to decrease the vulnerability for soft-errors in single-core architectures were discussed.

Generally speaking, all those single-core based methods are applicable to a multi-core architecture. For instance, every core can be extended with error-detection-and-correction codes so that it is able to handle a fault that occurred by its own.

However, two issues have to be considered carefully if a single-core based method is applied to a multi-core architecture. First, inherent redundancy of multi-core architectures might be ignored if all the mechanisms to mitigate soft-errors are implemented internally in each single-core. Second, the overhead of each processor core will contribute to the overall overhead of the entire multi-core system. As a result, recent research has suggested that the problem of soft-error resilience in multi-core systems needs to be approached from an orthogonal perspective [Jey11]; this means the approach should consider redundancy of different resources during the soft-error mitigation procedure.

An important issue with regard to the reliability of a multi-core architecture which uses redundancy of similar cores is to maintain identical instruction streams between redundant processor cores. In other words, the internal status of two cores should be the same at any given point of time. One of the most well-known methods to tightly synchronize two processors is called lock-stepping which has been proposed in [Smo06]. As a result, the first challenge of synchronizing two or more processor cores can be resolved by using lock-stepping. It will be shown that most of the methods that work based on comparing results of two identical processor cores rely on lock-stepping as a way of synchronization. However, lock-stepping incurs some drawbacks on the system. For example, if one processor core has to wait for an external handler, the other core needs to halt its procedure as well, until both processors will be timely aligned again.

A number of methods that take the advantage of inherent replication of processor cores in a multi-core architecture for soft-error detection or corrections can be found in [Agg07, Gom03]. These methods work by pairing two or more cores to check their execution results. The two above mentioned papers use lock-stepping in order to maintain the synchronization between identical processor cores. However the amount of information which is checked to detect a mismatch is different between the different approaches. For example, while [Agg07] checks all the outputs of two processors at every clock cycle, the Redundant-Multi-Threading (RMT) [Rei00] is developed in such a way that only stored addresses and associated values will be checked to detect soft-errors. Limiting the amount of data that should be checked for soft-error detection is useful to develop a light and fast soft-error detection mechanism; however, the error coverage of its error-detection mechanism might be compromised.

The authors of [Gup08] proposed a method which connects the internal pipeline stages of two processor cores via high-speed routers. Since the intermediate results of a pipeline stage are earlier available in the pipeline stage as compared to the main output of the processor, a faster soft-error detection can be obtained. The drawback of this method lies in a more complicated structure of the communication grid. Moreover, if the multi-core architecture is highly homogeneous, the regular structure of the multi-core architecture might be jeopardized. As a result, this architecture is more suitable for designs which are composed of a combination of dedicated cores, general-purpose cores, memories etc. (so-called semi-homogeneous systems).

As mentioned before, the inherent redundancy of multi-core architectures was an interesting framework to implement many readily available redundancy-based soft-error mitigation methods. One can mention the Dual Modular Redundancy (DMR) [Vad10], Triple Modular Redundancy (TMR) [Tha08], and check pointing [Wan06] techniques. The following paragraphs will discuss two redundancy-based soft-error mitigation techniques, including their shortcomings.

An example of one of the promising redundancy-based methods is Reunion [Smo06] which is based on check-point and recovery. It offers a low overhead recovery mechanism. In this method, a set of instructions called ‘fingerprints’ are generated by identical cores at some specific time intervals. Before committing each instruction, the ‘fingerprints’ should be compared against the same redundant cores executing the same instructions. The instructions are allowed to change the status of the processors only if the compared ‘fingerprints’ match each other. In the case of a mismatch, the status of both processors will be resumed from the last known status (check-point). At that moment the stored status will be loaded again in both processors and the execution will be continued from a previously known status.

The Reunion architecture has the following drawbacks:
a) It is required to add a new pipeline stage to each processor core because every instruction should be verified before the commit stage. This requirement adds an extra complexity to this design and it incurs extra power overhead.

b) Serializing instructions might cause problems. Since every instruction should be halted for one extra clock cycle, this can decrease the performance of the processor if the workload is serially oriented.

Another state-of-the-art method has been proposed in [Jey11]. It is a promising method which is called Unsync. In this method each core will be enriched with some hardware soft-error detection-and-correction mechanisms (such as the parity or Hamming codes). Upon detection of soft-errors in one of the cores, the status of the correct core will be transferred to the status registers of the erroneous core. Figure 6.1 shows this mechanism. The bold link between the two processors (indicated as link) is being used to copy the required status, including memory and register contents, from the correct core to the faulty core. Each processor signals the interrupt unit if an error occurs in one of the processor cores (via the previous hardware-detection mechanism). The interrupt handler halts both processors and then transfers the status of the correct processor to the faulty processor.

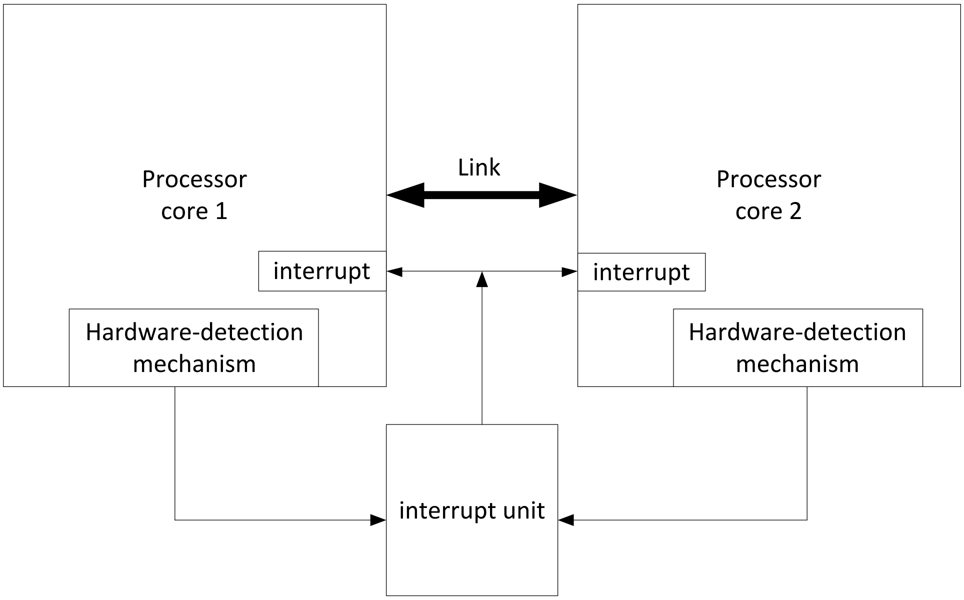


Figure 6.1. The basic set-up of the Unsync architecture.

The most important benefit of the Unsync architecture is that there is no need to synchronize the internal status between the two processors; since soft-error detection is carried out by internal error-detection-and-correction codes rather than comparing the internal status of identical cores. However, this architecture suffers from the following shortcomings:

a) Transferring the correct status of one processor to the other one might be very costly. This is because the content of all registers and memory needs to be transferred from one processor core to the other one.

b) Performing soft-error detection by internal hardware mechanisms might be a solution to eliminate the synchronization between two cores; however, the coverages of those methods are very low. For example, comparing the status of two processors clock-by-clock can hamper the propagation of all single errors. However, the parity code is only able to detect an error in the sequential parts. Moreover, there should be one clock-cycle delay between reading and writing data in that particular sequential unit so that the parity code can detect an error.

6.3. The motivation to propose our technique

Our motivation to develop a new soft-error recovery mechanism was to eliminate the requirement of transferring the status of one processor to another during the recovery process. Also, there was an interest to integrate the soft-error mitigation mechanism that has been developed in chapter 5 into a multi-core architecture.

With regard to the soft-error detection in our proposed architecture, the output of each functional unit of each processor core will be connected together for the purpose of soft-error detection. Therefore the complexity of our communication grid is less than methods which connect pipeline units; on the other hand it is more than those ones that connect the output of each processor core for comparison.

It is also important to mention that the architecture of our multi-core system is composed of DSP processor cores, a general purpose processor, memories and peripherals. A scheme of this multi-core architecture is shown in Figure 6.2. As can be seen, the architecture of our system is semi-homogeneous; therefore introducing a new communication grid to connect functional units does not jeopardize the regularity of this architecture.

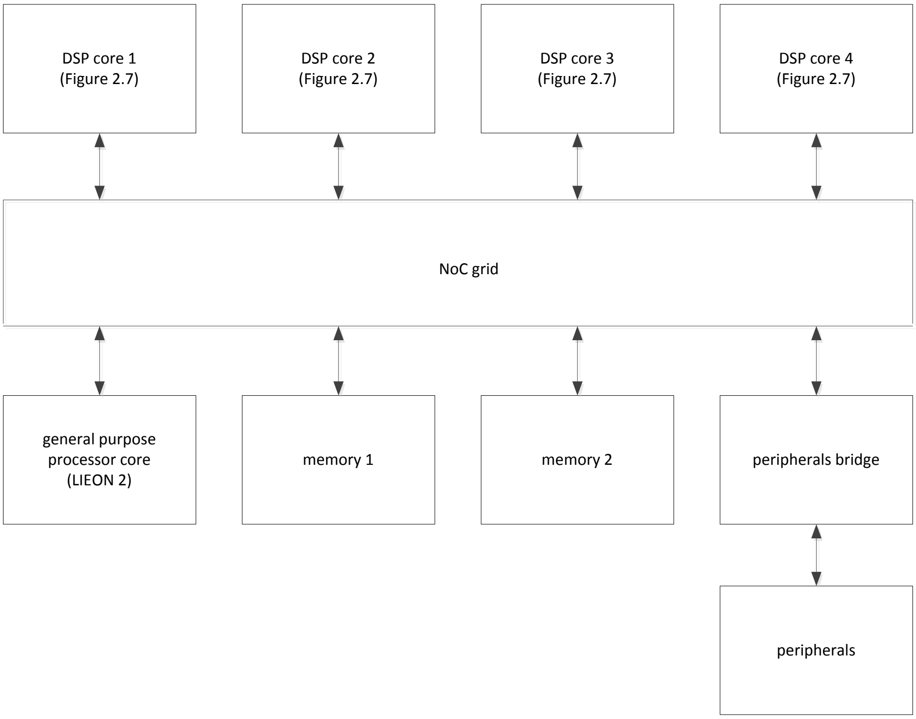


Figure 6.2 The organization of our multi-core system.

With regard to the soft-error correction method in our architecture, each core resolves the impact of soft-errors internally and therefore there is no need to copy the whole status of one processor core to the other one. This mechanism works by re-execution of the last instruction locally in both processor cores as it will be shown later, this mechanism provides 90% soft-error detection coverage, while the recovery overhead to mitigate the impact of soft-errors is very low.

6.4 Our approach for soft-error mitigation in multi-core systems

The first subsection discusses the details of the soft-error detection mechanism in our architecture and the second subsection provides our recovery mechanism.

6.4.1. Soft-error detection approach

The soft-error detection mechanism in our architecture is based on comparing the output of identical functional units of two processor cores against each other. This is because our multi-core architecture is composed of several DSP cores which in turn are VLIW architectures composed of several functional units (as described in chapter 2 section 2.5). It is therefore feasible to use a more fine-grained detection mechanism for each core as compared to the Reunion architecture [Smo06]. Moreover, we require a fast detection mechanism which can detect a failure in a processor as quickly as one clock cycle after its occurrence. This latter criteria is required for the correct operation of the 'freezing' mechanism, which was introduced in chapter 5, in this multi-core architecture.

Having mentioned all required considerations, the immediate output of each functional unit is compared to the counterparts functional units in identical cores to detect an impact of soft-errors. Figure 6.3 shows this mechanism; identical functional units will be compared with each other. For example, the A0 unit of processor core 1 is compared to the A0 unit of processor core 2, while processor core 1 and core 2 are executing the same workload. By comparing the output of two functional units, it is feasible to detect an error almost immediately before propagating to the higher succeeding levels, such as Memory units. This quick soft-error detection mechanism is essential in our architecture because we are interested to employ the error-recovery mechanism which was proposed in Chapter 5 (was called 'freezing') in this multi-core environment. This is because the 'freezing' approach works by re-execution of the very last instruction and hence a zero-latency detection mechanism is absolutely necessary for its correct operation.

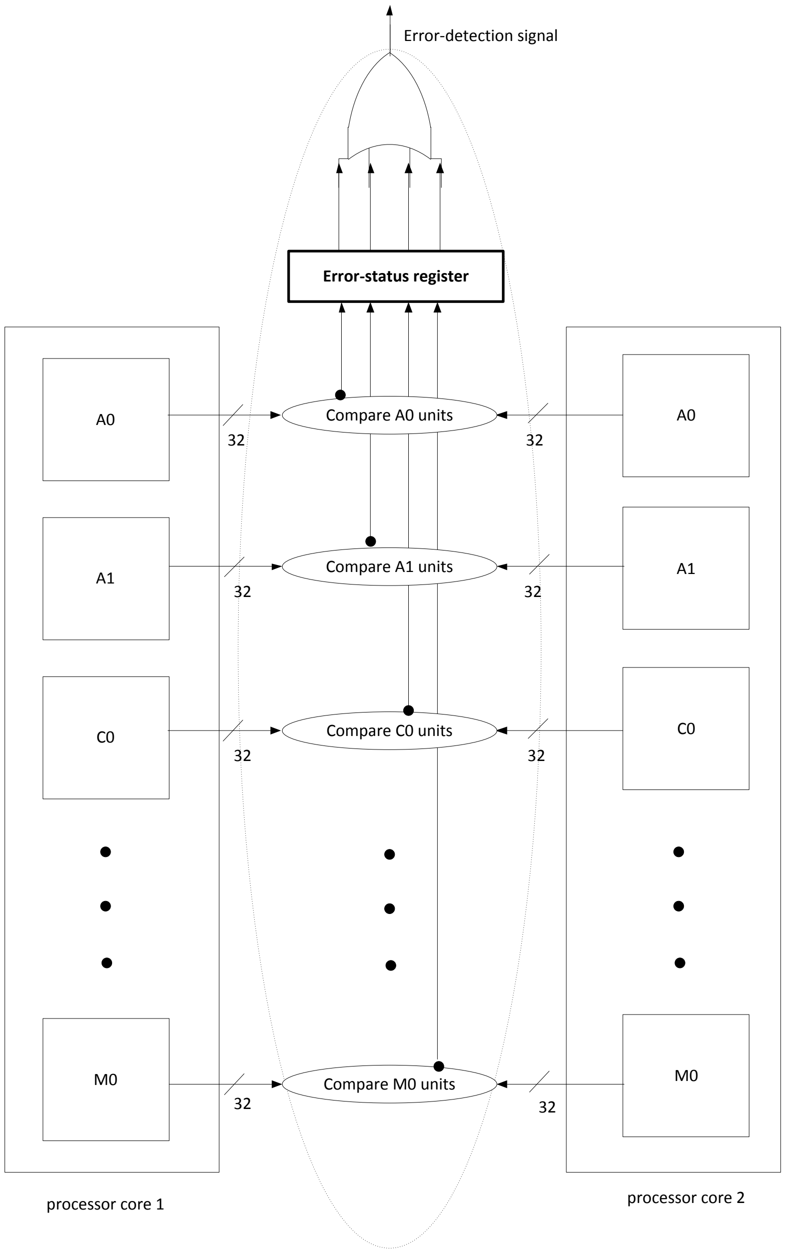
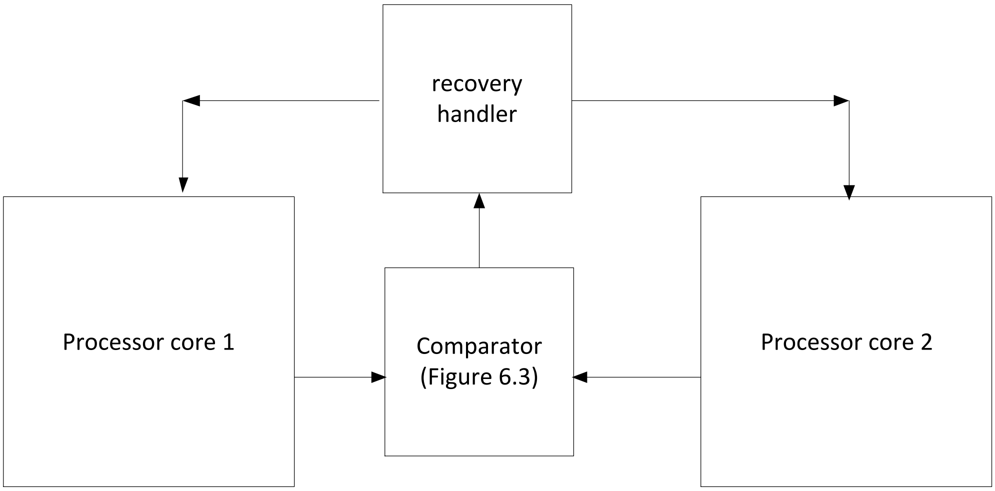


Figure 6.3. Comparison between identical functional units in identical processor cores.

The error-status register in figure 6.3 (indicated with bold lines) signals an error in at one of the functional units. The bits in this register indicate which functional units are erroneous. For example, bit 0 for A0, bit 1 for A1 etc. All the bits of this error-status register go to an 'OR' gate which its output is providing a soft-error detection signal. This signal indicates that at least one of the functional units is erroneous. All the compare units, error-status register and soft-error detection signal (indicated by a dotted line) are hereafter referred to as the 'comparator'.

Figure 6.4 depicts the overview of the entire architecture. It shows two processor cores that are compared together to form a core pair. The details of the comparator have already been shown in Figure 6.3. The workloads that are executed in both processor cores are identical and synchronized. Hence, the internal status of these two processor cores should be the same; otherwise there is a failure in one of the cores. However, this mechanism does not indicate which core is erroneous.

Each core in this architecture has been modified by employing the mechanism of chapter 5; this means there are shadow registers in each functional unit which make it feasible to re-execute the last instruction in each core upon detection of a soft-error. The soft-error recovery unit is responsible to initiate a re-execution in both processor cores. To be able to re-execute the last instruction, the normal operation of a core must be halted and then the very last inputs will be propagated again in the functional unit one more time. The details of the re-execution mechanism will not be discussed here since detailed explanations have already been discussed in Chapter 5. In the remainder of this chapter the focus will be on how to integrate that architecture into a multi-core environment.



igure 6.4. The general architecture of our mechansim.

6.4.2 Soft-error recovery approach

As shown in Figure 6.4, a recovery handler has been added to the architecture to be able to carry out the recovery mechanism. The responsibility of the recovery handler is to start the recovery process in each core. The recovery mechanism is based on halting the healthy functional units in each core, and activating the re-execution of the last instruction in the erroneous functional unit (or functional units, if more than one). As a result, both cores will halt their normal flow after a soft-error detection, and subsequently the last instruction will be re-executed in both cores. Providing that the detection mechanism can detect a soft-error within at most one clock cycle after its occurrence, this partial re-execution will eliminate the effect of a soft-error in the erroneous core.

Figure 6.5 shows the timing of the described mechanism. At time T1, a soft-error has occurred in core 1 (indicated by X), it has been detected in T2 by the comparison mechanism of the two processor cores 1 and 2. As a result, a re-execution has started in both cores at time T3. For one clock cycle (T3 - T4), both processor cores will re-execute the last instruction. After re-execution, both processor cores can continue their normal execution providing that the soft-error is eliminated (the detection-signal is turned low in this case).

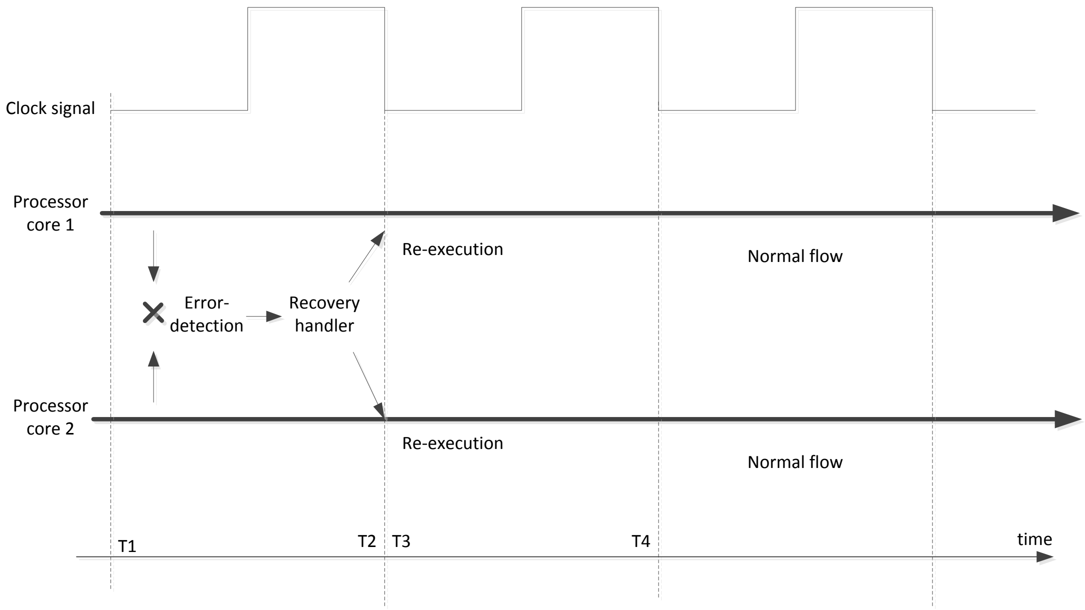


Figure 6.5. Timing diagram of error-recovery of two processor cores 1 and 2.

However, there exist some situations where this re-execution is not able to recover the correct status of the erroneous processor core. For example, in some cases the fault has already occurred several clock cycles prior to detection. In this case, re-execution of the last instruction is not helpful to recover the correct status of the processor cores. In other words, if a fault has occurred more than one clock cycle before the detection, rolling back one clock cycle before will re-produce an erroneous status again, and hence the recovery mechanism has failed.

Figure 6.6 shows an example of the mentioned scenario. Suppose that a fault has occurred in register-1 of processor core 2 of Figure 6.6 during clock cycle 'I'. This change takes two clock cycles for being read and propagated to the output of the functional unit. So detection can only take place in clock cycle 'I+2'. At this point, rolling back one clock cycle before, or 'I+1' and reading the status of the processor core is useless. This is because the data which is present in clock cycle 'I+1' in register-1 is already faulty.

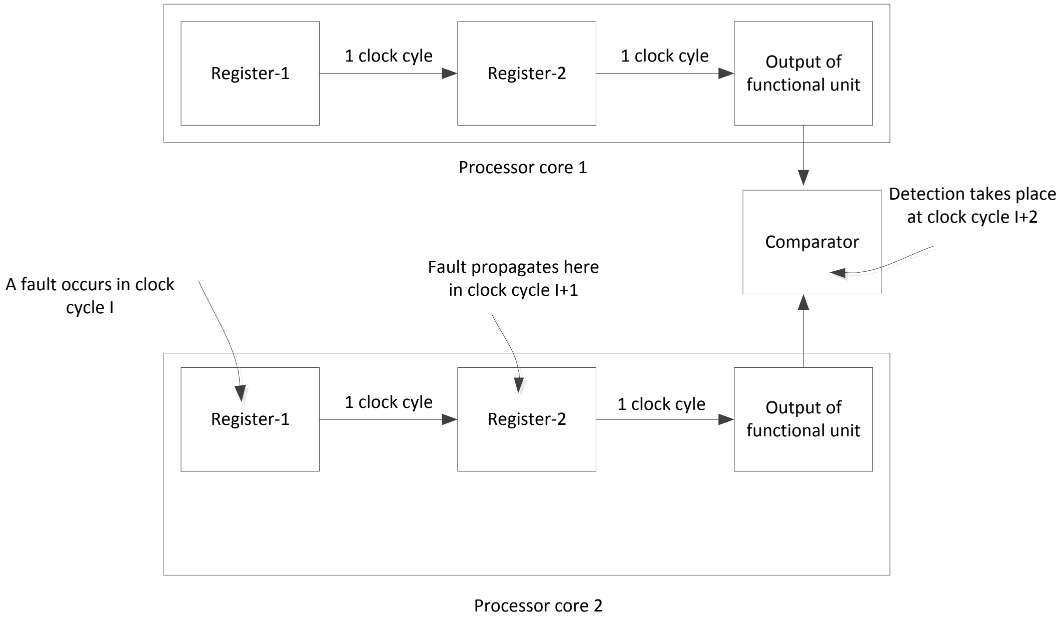

Register-1
1 clock cyle
Register-2
1 clock cyle
Output of functional unit
Processor core 1
Comparator
Detection takes place at clock cycle I+2
A fault occurs in clock cycle I
Fault propagates here in clock cycle I+1
Register-1
1 clock cyle
Register-2
1 clock cyle
Output of functional unit
Processor core 2

Figure 6.6. Detection of an error more than one clock cycle after the occurrence.

The probability of the above mentioned scenario is rare, because the execution time for most of the instructions in a DSP processor is one clock cycle (DSP instructions are based on Reduced-Instruction-Set-Computer, RISC architecture). However, in some cases a fault can be latent, i.e. a fault can manifest itself in a register that is going to be used in several (more than one) following clock cycles.

In the case of a detection of more than one clock cycle after occurrence, two scenarios can be followed. The first approach suggests loading a safe check-point from one clock cycle prior to the fault occurrence.

This scenario requires a heavy overhead for saving the intermediate status of each processor core at several time intervals.

The second method is to reset both processors to the initial state, a so called re-boot procedure to the beginning of the program code. This method implies a large performance loss, because both processor cores have to execute their entire workload one more time from the beginning. It is important to note that the detection mechanism of our method does not have the possibility of recognizing the erroneous processor core; so every action including a re-boot should be applied to both processor cores.

6.4.3 Operational phases of our architecture

The operational mechanisms of the proposed architecture can be best understood by observing its different phases of operation:

a) Error-free mode: the two identical cores execute the same instructions in a synchronous mode. The memory provides the same data for both cores and also the interrupts are received in both cores at the same time. As long as the detection-signal is low it implies that the internal statuses of the two cores are identical and the normal behaviour of the system will be continued.

b) Detection mode: detection is carried out by the comparator which was depicted in Figure 6.3. This comparator compares the outputs of two identical functional units in two identical processor cores (such as A0 in processor core 1 and A0 in processor core 2, while processor core 1 and 2 are identical) in order to detect a mismatch. Upon a mismatch detection, the comparator signals the recovery handler of the core pair, as shown in Figure 6.4. The recovery handler starts the recovery mode subsequently.

c) Recovery mode: in the case the recovery handler receives a signal from the compare unit, a recovery process in both processor cores of the core pair will be started. In this mode, the following procedure will be carried out in both processor cores:

First, the normal execution flow in both processor cores of the core pair will be halted temporarily. Then the correct functional units in both cores will enter the 'freezing' state (which means the output data of unaffected functional units will be held for one clock cycle), while the erroneous functional units in both processor cores will re-execute the last instruction to mitigate the error. One clock cycle after this re-execution, two scenarios are likely to happen: the first scenario is that the detection-signal is low which implies the erroneous functional unit has recovered. In this case the normal procedure of the core pair can be continued. The second scenario is that there is still a mismatch between the two processor cores. As discussed earlier, this can be due to the occurrence of a fault in an earlier clock cycle than the clock cycle prior to its detection. As a result, rolling back to one clock cycle before the detection cannot recover the error. In this case, both processors cores will be re-booted to the starting point of workloads (after IO initialization). This scenario has been depicted in Figure 6.7. This shows two processor cores, core 1 and 2. The second mismatch after the first mismatch indicates that the re-execution could not recover the processor core pair and hence a re-boot has been performed in both processors.

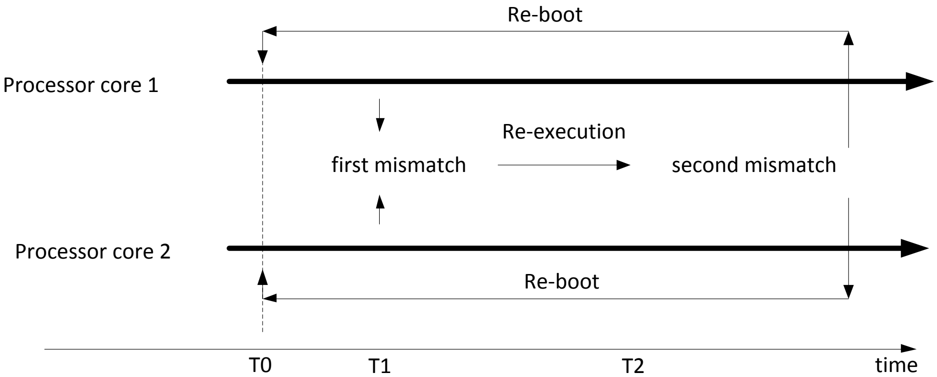


Figure 6.7. Timing diagram of our proposed architecture.

It is important to mention that even though the re-boot results in a large performance loss, fault-injection experiments show that about 90% of detected soft-errors can be mitigated within only one clock cycle via the re-execution method and only 10% require a re-boot. For example, if one considers the extreme rate of soft-errors of occurring once per day, our mechanism induces a re-boot with a rate of only once per ten days.

6.5 Additional features of our architecture

The architecture we have proposed in this chapter has some benefits as compared to other similar state-of-the-art methods. However, some challenges should be resolved in order to implement this mechanism in a multi-core environment. In the following paragraphs, the advantages and disadvantages of the proposed architecture will be discussed.

The first advantage is that inter-core communication is not required. Our proposed method recovers processor cores by re-execution of the last instruction in each core separately, in the case the fault has occurred just one clock cycle before its detection; or it re-boots both cores separately in the scenario that a fault has manifested itself in the system and is appearing several clock cycles after its occurrence. In any case, there is no need to exchange the status between two processor cores. This feature is beneficiary as compared to Unsync [Jey11] in which the correct status of a core should be copied from the unaffected core to the erroneous one.

A second advantage is that there is no need to add any extra unit to the internal pipeline units of processor cores. For example, in the Reunion [Smo06] architecture, one extra check unit should be incorporated in the pipeline units of each core to be able to check the correctness of fingerprints. That modification incurs a performance degradation to each core even in the situation that there is no error in the system. However, the only modification which is required in our method is the insertion of shadow registers and multiplexers which hardly interfere with the normal flow of the pipeline units.

Last but not least, the performance of each individual processor core will not be compromised. Every core in the modified multi-core architecture has the same performance as the base-line core architecture. This factor is an advantage over the Reunion architecture in which the fingerprints should be generated (extra clock cycles) and then be compared together no matter of the occurrence of a fault or net.

However, there are still some challenges that need to be resolved in order to implement our method in a multi-core architecture. First, it is required to have a reliable method of synchronization between two processor cores. Since the fault detection is carried out by comparing the partial results, these two processor cores should execute exactly the same instruction at the exact same moment. This challenge also exists in other methods [Smo06] which rely on comparison to detect an error. Unsync [Jey11] claims that there is no need to synchronize cores (as the name implies), because the detection mechanism is carried out by internal hardware error-detection mechanisms, such as usage of the Hamming code. However, the soft-error coverage of this method is about 33% which is much lower as compared to comparison based methods. Moreover, input data might be lost during a re-boot, however, all the other approaches have similar problems. We use two cores for one workload, so the efficiency is reduced, but other approaches have this problem as well.

The re-boot process is depicted in Figure 6.7; even though it might seem simple, it requires a more detailed explanation.

The normal re-boot process, a so-called hard re-boot or reset, is performed via the hardware. In this case, the Program-Counter (PC) is rolled back to the start address of the program memory and all the data will be lost. This address is normally 0 as it requires a hardware reset in the register containing the PC.

However, another form of re-boot has been used in our mechanism where the PC will be rolled back to an intermediate address which is not necessarily the start of a program code. This concept is called a soft re-boot and is depicted in Figure 6.8. As can be seen in this figure, the information resides in the program memory which is divided into two parts; the initialization part where information like the address of a NoC handler and the memory address are stored (from address 0000 to 00FF) and the program code in which the actual code of a workload is stored (address 0100 to FFFF).

This initial information is essential in a multicore environment for the correct execution of the program code and this information needs to be loaded again if any re-execution is performed.

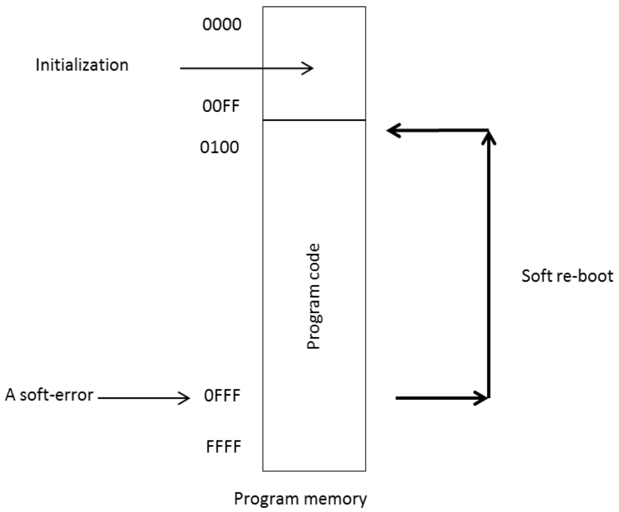


Figure 6.8. Concept of soft re-boot.

Suppose that a soft-error has been occurred when the PC is at the address 0FFF and a re-execution is performed to recover from this error. A soft re-boot implies that the PC is set to the beginning of the program code and after the initialization information (0100 in this example); hence the initialization information will be intact. However, one is able to perform a soft re-boot only in a simulation environment and not in the real hardware. This issue needs to be investigated if the proposed mechanism is going to be implemented in a chip. One solution could be storing the initial information in a separate memory and then re-loading that information upon a hard re-boot. However, the cost on performance needs to be studied first. This issue has not been addressed in this thesis any further, since all the experiments were performed in a simulation environment.

Table 6.1 summarizes the differences between our proposed method as compared to the Reunion [Smo06] and UnSync approach [Jey11].

Table 6.1. Comparison of some features of our approach with others.

feature / method	modification of pipeline units	requires synchronization	soft-error coverage	Inter-core communication
Reunion [Smo06]	Yes	Yes	High (more than 75%)	Yes
UnSync [Jey11]	No	No	Low (33%)	Yes
Our proposed approach	No	Yes	High (90%)	No

6.6 Experimental setup and evaluation of our approach

The first sub-section discusses the details of our evaluation method and the second subsection provides the results of our analysis.

6.6.1 Experimental set-up

In order to evaluate the effectiveness of our proposed architecture, a VHDL Register Transfer Level (RTL) set-up has been developed. The RTL architecture is composed of four DSP Xentium cores (as was already shown in Figure 6.2). These four Xentium processor cores have formed two core pairs combinations. Moreover, each core pair has only one memory handler, one interrupt receiver and one clock source. In this case, both Xentium processors in a core pair will receive and observe the same actions and remain synchronous with each other. In addition, one comparator and one recovery handler are required for each core pair. These units have been developed in VHDL as well. The internal architecture of each Xentium processor is exactly identical as described in Chapter 5.

The workload which is executed in each processor core is a FIR filter program. It is only required to have an identical workload in each core pair as the results of two processor cores are compared with each other to detect a mismatch. So, it is possible that one core pair executes a FIR program while the other core pair executes a FFT program.

During execution of the workloads, a fault-injection process has been carried out on all functional units of each Xentium processor to evaluate the soft-error coverage of the modified multi-core architecture. The fault injection process has been conducted by injection of SET fault models at random times in random nets of each functional unit. The final outputs of the processor cores have been monitored to trace the propagation path of each injected fault. The model used to emulate SETs is the same model which was described in Chapter 4. The simulation framework which has been used to conduct soft-error injection is the one described in Chapter 3.

6.6.2. The soft-error coverage

As mentioned before, a simulation-based soft-error study at VHDL level has been carried out to assess the achieved sensitivity of our multi-core architecture with regard to soft-errors.

Each core pair executes a FIR workload and during the execution of the workload, predefined soft-error fault models were injected into the design. Soft-error faults have been injected in all functional units of each Xentium core during our fault-injection process.

The classification of the impact of faults in the output of each core pair are categorized as failure and not-effective ones. If an injected fault can alter the output of the core pair, then that fault is classified as a cause of a failure, otherwise it is counted as a not-effective fault that has no impact on the system. The behaviour of each core pair with regard to injected faults has been stated in terms of a percentage. For example, if 10 out of 100 injected faults are able to modify the output of the core pair, then the sensitivity of this core pair with regard to injected faults is 10%. Our goal is to decrease the sensitivity of each core pair.

The number of fault-injections in each core has started from the initial value of 1000 (a random value) and has been increased until a point of convergence can be observed in terms of the sensitivity of a core pair. The mathematical details of obtaining the convergence point are out of the scope of this chapter and have been discussed in detail in Chapter 4.

Table 6.2 shows the number of fault-injections and the sensitivity of each core pair for 20,000 fault-injection experiments. As can be seen in this table, about 8,130 faults can change the correct output of a core pair in an unprotected design, resulting in a SET sensitivity value of 41%. This value is for an architecture without any error mitigation measure. As a result, 59% of the injected faults could not modify the output of a core pair while 41% of injected faults propagated from the internal functional units to the output core level.

As Table 6.2 shows, our modified architecture can mitigate 94% of failures in a core pair. In another words, only 1230 faults could escape the detection mechanism and change the correct functionality of a processor core; which implies the sensitivity of our architecture is 6%. The reason that still 6% of SET faults can escape the detection mechanism is that our detection mechanism only checks the output of the functional units. So, some injected faults were unable to reveal themselves at the output of a functional unit while they could propagate to the output of a core pair.

The last two columns of Table 6.2 (indicated by grey) show the recovery latency of our proposed architecture. It can be seen that 90% of the detected faults could be recovered with the re-execution approach, which takes only one clock cycle. However 10% of the detected faults could not be mitigated via the re-execution approach and hence a re-boot to the initial status of both cores was required. If one considers the occurrence rate of soft-errors in digital systems as once per 24 hours, our architecture incurs a re-boot only once per 10 days.

Table 6.2. The parameters of our mitigation method.

Parameter / Design	# Fault-injections	# Failures	Soft-error Sensitivity (%)	Recovery method (%)	
				With Re-execution	With Re-boot
Original design	20,000	8130	40.65	N.A	N.A
Our enhanced architecture	20,000	1230	6.15	90	10

Table 6.3 compares the achievements of our design as compared to UnSync [Jey11] and Reunion [Smo06] architectures. As a comparison between our method and UnSync, this latter approach has a low detection coverage (indicated by underlining), since the detection-mechanism in UnSync is based on error-detection-and-correction codes, for example the parity code. It is important to mention that the parity code is able to discover an error in sequential units only, while there should be at least one clock cycle delay between writing information and reading in that particular sequential unit. Moreover, the recovery latency in the Unsync architecture is higher (indicated by underlining) as compared to ours since the mitigation method in this architecture is based on transferring the status between two processors.

As a comparison between our architecture and Reunion, Table 6.3 shows that the Reunion architecture has a long recovery latency. This is because the recovery mechanism in the Reunion is based on checkpoints and recovery. So, after a detection, both processor cores should load a correct checkpoint (which includes an image of all the registers and memory files). The recovery latency of our method results from the fact that 90% of the detected faults can be mitigated within one clock cycle while the remaining 10% need a re-boot of the processor. Based on our fault-injection experiments, the average latency of re-boot was 80 clock cycles which results in a latency of our method to 8.9 (0.9 * 1 clk + 0.1*80 clk) clock cycles (indicated by grey).

The outcome of Table 6.3 is that our architecture has the best recovery latency among these three methods. So, if the performance is a concern, our method is the best candidate to implement a soft-error reliable multi-core architecture.

Table 6.3. The comparison between our architecture and two other state-of-the-art methods.

Parameter **design**	**Detection coverage (%)**	**Recovery Latency (clock cycles)**
UnSync	33	30
Reunion	100	18
Our architecture	94	8.9

6.7 Conclusions

Computer architectures are moving towards multi-processor core approaches. Using increasing dense architectures in the new generation of multi-cores along with shrinking the technology dimension to 14nm make the introduction of lightweight soft-error resilient mechanisms a mandatory feature for future architectures. We proposed in this chapter a soft-error mitigation technique for multi-core architectures which is based on re-execution of the last instruction of each core. This architecture does not require transferring the whole status of one processor core to another. Hence the recovery time is short, as compared to other state-of-the-art methods. Fault-injection simulation results on the Xentium multi-core target system show that 90% of the detected SET faults can be recovered within one clock cycle while the remaining 10% will incur a re-boot of the system. As a result, if the execution time of a workload is limited (a re-boot to the start of the workload does not incur a huge performance loss) our proposed method will be beneficiary.

References

[Agg07] N. Aggarwal, P. Ranganathan, N. P. Jouppi et al., “Configurable isolation: building high availability systems with commodity multi-core processors,” in Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA), pp. 470-481, 2007.

[Bau95] R. Baumann, T. Hossain, S. Murata et al., “Boron compounds as a dominant source of alpha particles in semiconductor devices,” in IEEE International Symposium of Reliability on Physics (ISRP), pp. 297-302, 1995.

[Can04] E. Cannon, D. Reinhardt, M. Gordon et al., “SRAM SER in 90, 130 and 180 nm bulk and SOI technologies,” in IEEE International Symposium on Reliability on Physics (ISRP), pp. 300-304, 2004.

[Gom03] M. Gomaa, C. Scarbrough, T. Vijaykumar et al., “Transient fault recovery for chip multiprocessors,” in Proceedings of 30th Annual International Symposium on Computer Architecture (ISCA), pp. 98-109, 2003.

[Gup08] S. Gupta, S. Feng, A. Ansari et al., “StageNet-Slice: a reconfigurable microarchitecture building block for resilient CMP systems,” in Proceedings of International Conference on Compilers, Architectures and Synthesis for Embedded Systems, pp. 1-10, 2008.

[Jey11] R. Jeyapaul, F. Hong, A. Rhisheekesan et al, “UnSync: A soft-error resilient redundant multi-core architecture,” in IEEE International Symposium on Parallel and Distributed Processing, pp. 632-642, 2011.

[Kay00] S. Kayali, “Reliability considerations for advanced microelectronics,” in Proceedings of Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 99-105, 2000.

[Lyo00] D. Lyons, “Sun screen: soft-error issue in Sun enterprise servers,” www.members.forbes.com/global/2000/1113/0323026a.html, 2000.

[Rei00] S. K. Reinhardt, S. S. Mukherjee, “Transient fault detection via simultaneous multithreading,” in Proceedings of International Symposium on Computer Architecture (ISCA), pp. 25-36, 2000.

[Roc92] L. R. Rockett, “Simulated SEU hardened scaled CMOS SRAM cell design using gated resistors,” in IEEE Transactions on Nuclear Science, Vol. 39, No. 5, pp. 1532-1541, 1992.

[Shr10] A. Shrivastava, J. Lee, R. Jeyapaul, “Cache vulnerability equations for protecting data in embedded processor caches from soft-errors,” in Proceedings of Conference on Languages, Compilers, and Tools for Embedded Systems, Vol. 45, pp. 143-152, 2010.

[Smo04] J. Smolens, B. Gold, J. Kim et al., “Fingerprinting: bounding soft-error-detection latency and bandwidth,” in Proceedings of International

Conference on Architectural Support for Programming Languages and Operating Systems, Vol. 24, No. 6, pp. 22-29, 2004.

[Smo06] J. C. Smolens, B. T. Gold, B. Falsafi et al., "Reunion: complexity-effective multi-core redundancy," in Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture, (MICRO), pp. 223-234, 2006.

[Tes14] www.teslamotors.com, 2014.

[Tha08] D. D. Thaker, F. Impens, I. L. Chuang et al, "On using recursive TMR as a soft-error mitigation technique," in Proceedings of International Conference on Parallel Processing (ICPP), pp. 10-15, 2008.

[Vad10] R. Vadlamani, J. Zhao, W. Burleson et al., "Multi-core soft-error rate stabilization using adaptive dual modular redundancy," in Proceedings of the Conference on Design, Automation and Test in Europe (DATE), pp. 27-32, 2010.

[Wan06] N. J. Wang, S. J. Patel, "ReStore: symptom-based soft-error detection in microprocessors," in IEEE Transactions on Dependable and Secure Computing, Vol. 3, No. 3, pp. 188-201, 2006.

CHAPTER 7

Conclusions, Contributions and Recommendations for Future Work

7.1 Introduction

The topic of this thesis is soft-error mitigation techniques in digital circuits. In order to be able to propose an efficient mitigation technique, other prerequisites need to be investigated first. This includes investigation of sources of soft-errors, a framework to study the impact of soft-errors in digital circuits and also a realistic and practical simulation model for soft-errors. Being able to study the impact of soft-errors in the early development phases enables the designer to propose efficient soft-error mitigation mechanisms.

7.2 Contributions

The contributions of this thesis can be listed as follows:

- In chapter 2 the sources and effects of soft-errors have been investigated and related work in the field of soft-error injection have been addressed.

- In chapter 3, a framework has been developed to assess soft-error vulnerability of complex designs which are modelled in an HDL language. Minimizing the required CPU time to carry out soft-error analysis is one of the main challenges of an HDL–based soft-error analysis framework. The developed framework can invoke a conventional model of soft-errors in a design, assess its vulnerability to soft-errors and also extract the most vulnerable parts of an HDL design with regard to soft-errors. Since this soft-error analysis is carried out during the design development phase (before prototype manufacturing), the vulnerability of components can be investigated in detail and redesigned before the actual component is ready for manufacturing. Fault-injection experiments show that the elapsed CPU time to carry out a soft-error assessment can be decreased by 10% as compared to fast fault-injection methods and over 67% as compared to normal simulation-based fault injection methods [Roh11a, Roh11b, Roh13a].

- One of the current challenges regarding soft-error vulnerability analysis lies in how realistic simulation SET models are. In other words, there is still no consensus on the accuracy of the degree of similarity between the conventional discrete-pulse model of logic gate-level simulation as compared to real physical soft-errors. In chapter 4 of this thesis, a new logic gate-level fault model has been developed that accurately imitates (less than 15% error) the real behaviour of SETs for a 45nm CMOS technology node. This SET model has been extracted using laser-based fault-injections along with the asymptotic behaviour of SETs in a SPICE model. These simulation fault models solve one of the challenging aspects of the HDL-based soft-error simulation, which is the lack of a realistic model of soft-errors to mimic the impact of soft-errors in the development phase of a product [Roh13b].

- In chapter 5 of this thesis, an efficient detection and correction method for control logic of Very Large Instruction Word (VLIW) processors is introduced. The characteristics of VLIW processors along with features of DSP workloads have been used to develop an efficient soft-error mitigation technique for VLIW processors. This mitigation method imposes 4% overhead on silicon area and causes a 10% performance degradation, while the sensitivity of the DSP processor with regard to soft-errors decreases from 40% to 5% [Roh12].

- Another mitigation method which has been proposed in chapter 5 targets the data path of VLIW processors. The developed method, called freezing, has the ability to limit the penalty imposed on a design based on the soft-error occurrence rate that a system is experiencing. This method benefits from the architecture of DSP processors to halt the correct functional units for one clock cycle while re-execute the unit that was hit by a soft-error. Experiments show that the proposed method can mitigate half of the occurred soft-errors while the imposed overhead in terms of silicon area is only 10% and the speed of the processor is similar to the baseline design [Roh13c, Roh14a].

- Multi-core architectures have some unique features that can be exploited for the purpose of mitigation of soft-errors, such as the existence of several identical cores. One of our contributions in chapter 6 of this thesis is to develop a soft-error mitigation mechanism which exploits the existence of similar cores in the homogenous parts of multi-core architectures. In case of a soft-error detection, the two processor cores re-execute an instruction. This internal re-execution eliminates the requirement to transfer the entire status between two processor cores. The fault injection results show that 94% of detected soft-errors can be mitigated with this mechanism. The average recovery time is 8.9 clock cycles [Roh14b].

7.3 Conclusions

This thesis investigated different aspects of soft-errors in digital systems. The issue of soft-errors in a digital system is normally started with an evaluation phase to recognize the most sensitive parts of a system which affect the most from soft-errors. The next step will be modifying those sensitive parts to enhance the robustness of the entire system with regard to soft-errors.

In this thesis, in particular we proposed a fast simulation-based soft-error evaluation framework to analyse the behaviour of a system in presence of soft-errors. An enormous CPU time is a common drawback in simulation-based fault studies, so it was shown that it is possible to decrease the CPU time required to conduct fault injections up to 67% as compared to conventional methods. This achievement helps the designer to rapidly evaluate a system during its development phase and recognize the weak parts.

A realistic soft-error model which can be used in simulation-based fault-injections in logic-gate level was also developed in this thesis. This model helps to decrease the total time required to conduct fault-injections at a gate-level even further while preserving the accuracy of experiments. This model is only deviates 15% with real physical soft-errors and therefore can predict a realistic behaviour of a system. Integration of this model in the framework discussed in the previous paragraph can introduce an accurate and fast fault-injection framework.

Furthermore, some efficient mitigation methods to diminish the impact of soft-errors in DSP processors were introduced. These methods impose low overhead on area and speed of the system while they mitigate a large portion of soft-errors. These achievements show that using unique characteristics of a design can potentially result in efficient and customized solutions with regard to soft errors. Finally, a soft-error mitigation method for a multicore system which is composed of DSP processors was researched in this thesis. The results show that 94% of soft-errors can be mitigated in a multicore environment while on average 9 extra clock-cycles are imposed on the system. In the case of no soft-error, no performance loss is imposed on the system.

The results of this thesis can be used to propose a complete flow to investigate the impact of soft-errors during the development phase of a digital system. The results of chapters 3 and 4 can be applied to build the framework to study the impact of soft-errors on a system, while chapters 5 and 6 can be used as examples to employ the unique features of a system to decrease its sensitivity with regard to soft-errors.

7.4 Future work

The results of this thesis can be used in different frameworks for mitigating the impact of soft-errors. For example, the framework of chapter 3 which uses the logic gate-level model of soft-errors described in chapter 4 can be integrated with automatic insertion of saboteurs to build a fully automated and fast fault-injection tool. This tool can be used in the soft-error sensitivity phase in an ASIC digital design flow. This is shown in Figure 7.1 with a grey background. The input of the tool will be the synthesized gate-level net list (the output of a logic synthesis tool) and the output will be the sensitivity of the design with regard to soft-errors. The information can be used to improve the designs to reduce their sensitivity to soft-errors.

Finally, applying and testing the developed approach in other DSP processors will be interesting. Moreover, the investigation of soft-errors in a multicore architecture when also inter-core communication, shared memory access and I/O are taken into account, can be considered challenging topics for future work.

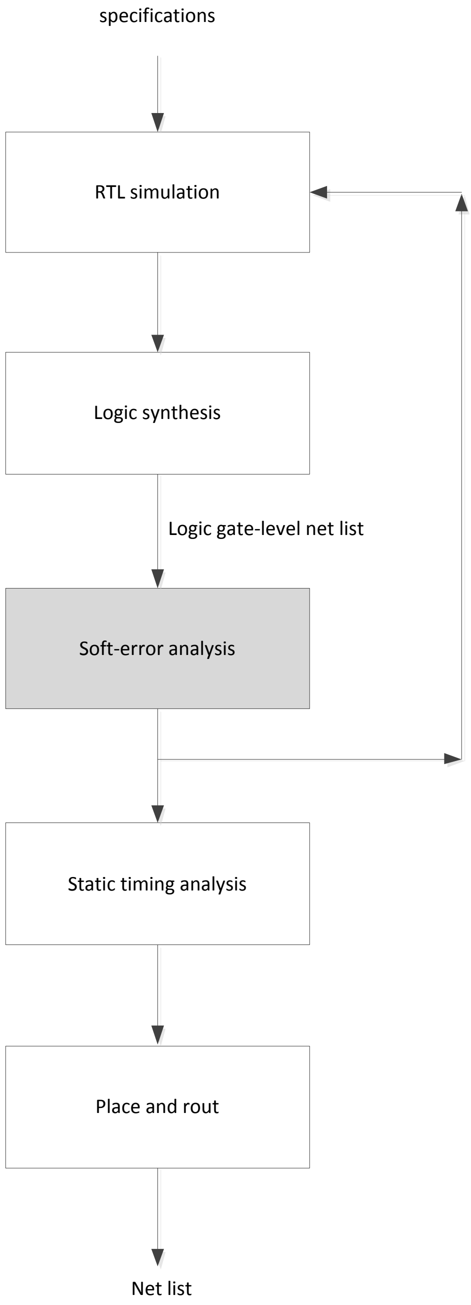


Figure 7.1 Soft-error analyses in an ASIC digital design flow.

List of our publications

[Roh11a] A. Rohani, H. G. Kerkhoff, ″Study of the effects of SET induced faults on submicron technologies,″ in IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), pp. 41-46, 2011.

[Roh11b] A. Rohani, H. G. Kerkhoff, ″A Technique for Accelerating Injection of Transient Faults in Complex SoCs,″ in Euro-micro Conference on Digital System Design (DSD), pp. 213-220, 2011.

[Roh12] A. Rohani, H. G. Kerkhoff, ″An online soft error mitigation technique for control logic of VLIW processors,″ in Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), pp. 85-91, 2012.

[Roh13a] A. Rohani, H. G. Kerkhoff, ″Rapid Transient Fault Insertion in Large Digital Systems,″ in Microprocessors and microsystems Journals, ISSN 0141-9331, Vol. 37, No. 2, pp. 147-154, , 2013.

[Roh13b] A. Rohani, H. G. Kerkhoff, ″Functional unit for a processor,″ European Patent, EP13191370.9, 2013.

[Roh13c] A. Rohani, H. G. Kerkhoff, E. Costenaro, D Alexandrescu, ″Pulse-length determination techniques in the rectangular single event transient fault model,″ in International Conference on Embedded Computer Systems: Architectures, Modelling, and Simulation (SAMOS), pp. 15-18, 2013.

[Roh14a] A. Rohani, H. G. Kerkhoff, "Two soft-error mitigation techniques for functional units of DSP processors," in IEEE European Test Symposium (ETS), pp. 1-6, 2014.

[Roh14b] A. Rohani, H. G. Kerkhoff, "A soft-error mitigation technique in a multicore architecture composed of DSP cores," to be submitted for 16th IEEE Latin-American Test Symposium (LATS), 2015.

BIOGRAPHY

Alireza Rohani obtained his B.Sc. degree in Computer Engineering from the Shahed University, Tehran, Iran, in 2006. Later, he obtained his M.Sc. degree in Computer Architecture from Amirkabir University of Technology, Tehran, Iran, in 2010. In June of 2010, he joined the Computer Architecture for Embedded System (CAES) group at the University of Twente to pursue his Ph.D. degree in the field of Dependability of modern processors. During this time, he worked on the TOETS research project and published several papers on international conferences and scientific journals.