ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Software Simulation of Temperature Distribution & Device Degradation of Integrated Circuits

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτριος Α. Ροδόπουλος

Επιβλέπων : Δημήτριος Σούντρης
Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Μάιος 2011

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Software Simulation of Temperature Distribution & Device Degradation of Integrated Circuits

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτριος Α. Ροδόπουλος

Επιβλέπων : Δημήτριος Σούντρης
Επίκουρος Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 16/06/2011.

.....	.....	.....
Κιαμάλ Πεκμεστζή Καθηγητής Ε.Μ.Π.	Δημήτριος Σούντρης Επίκουρος Καθηγητής Ε.Μ.Π.	Ιωάννης Ξανθάκης Καθηγητής Ε.Μ.Π.

Αθήνα, Μάιος 2011

.....

Δημήτριος Α. Ροδόπουλος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

To my grandparents,

Lt Gen Dimitrios Kapelaris and
Mrs Anastasia Kapelari

Table of Contents

Abstract

Advances in the modern semiconductor industry give rise to various reliability aspects of electronic design. On the one hand, device behaviour is dominated by stochastic phenomena that may vary even between the devices of the same technology. A timely example of such mechanisms is Bias Temperature Instability (BTI) and Random Telegraph Noise (RTN). On the other hand, the inevitable increase of functional block density in modern integrated circuits makes the temperature distribution a serious design constraint. As a result, fast, yet accurate thermal profiling is of vital importance both at design time and at runtime.

The first part of the current work deals with the time and workload dependent device variability of modern downscaled technologies. BTI and RTN are incorporated in circuit simulations of larger circuits, the parametric reliability of which is of major importance. There is a fundamental differentiation from the state of the art, since the atomistic approach towards BTI and RTN allows the observation of detailed workload dependency in the simulation results. This concept is materialized in a fully functional simulation framework of a 32 bit SRAM partition. Based on a real memory architecture, the workload of such a structure was extracted by realistic a realistic application and simulated on the circuit under test. Performance metrics of the circuit, where monitored during the simulation of different workloads. Each different workload, or RunTime Situation (RTS), is instantiated by a cumulative delay metric and a leakage energy value. This concept enables the clustering of RTSs into workload scenarios.

The second part of the current work deals with the numerical acceleration of the publicly available thermal simulator HotSpot-5.0. An extensive profiling of its source code, revealed a CPU intensive iterative method for the extraction of the transient solution. This method was replaced with a simplified equivalent that achieved the intended acceleration, without imposing any accuracy degradation. The accelerated version of the tool was successfully incorporated to a broader tool that performs hierarchical thermal profiling of Muli-Processor System-on-Chip (MPSoC) floorplans. That way, the application spectrum of such thermal analysis tools is significantly broadened.

Keywords: Atomistic Approach, Bias Temperature Instability, Circuit Simulations, Device-level modelling, Euler Method, Floorplan, Multi-Processor System-on-Chip, Ordinary Differential Equation, Power Traces, Random Telegraph Noise, RC Network, Runge-Kutta Method, Runtime Situations, Static Random Access Memory, Thermal Profiling, Workload Dependency

Acknowledgements

From NTUA, I have to thank Prof. Dimitrios Soudris for this inspiring collaboration. His executive and scientific insight has created a valuable academic experience. Dr. Antonis Papanikolaou has been a valuable co-supervisor. I thank him for his timely pieces of advice. I would also like to thank Dr. Kostas Siozios, being the first researcher I closely collaborated with for published material. This introductory work definitely encouraged me to pursue the current thesis. Finally, I am very grateful to Microlab, NTUA which allowed my participation in the 2011 IEEE ICICDT in Kaohsiung, Taiwan where I had the chance to present part of this thesis.

From IMEC vzw, I would like to thank Prof. Francky Catthoor for his restless guidance. His holistic perception introduced me to a powerful intellectual tool. I am looking forward to realizing its full capacity. Also, I would like to thank my co-supervisor Dr. Ben Kaczer. His continuous feedback was very helpful for my understanding of device-level concepts. I would like to express my appreciation to Dr. Stefan Cosemans for answering all my questions regarding circuit modelling and SRAM internal structure. Also, I would like to thank Dr. Maria Toledano-Luque for providing and explaining real device measurements for the calibration of our framework.

To my IMEC colleagues Swaraj Bandhu Mahato and Vinicius Valduga de Almeida Camargo, I would like to express my sincere gratitude for the fruitful cooperation. I would like to thank them for continuously stimulating this collaboration with their intellect and motivation. Working with them was an experience I will never forget.

Finally, I would like to express my endless love to my father Anastassios Rodopoulos, my mother Aikaterini Kapelari and my brother Aristeidis Rodopoulos. Without their support and encouragement, the current endeavour would be impossible.

List of Figures

List of Tables

List of Algorithms

Chapter 1: Introduction

1.1 Overview

The work presented in this text is partial fulfilment of the requirements for a Diploma by the School of Electrical and Computer Engineering of the National Technical University of Athens. This work can be divided in two separate parts, representing the involvement of the author in two different areas respectively.

The next three chapters of the text represent the work that was completed during a six month internship (October 2010 – April 2011) of the author in IMEC vzw, Belgium as a Master Thesis student. The author was involved in the creation of a fully functional simulation framework for a 32-bit SRAM partition. The simulations were focused on the parametric reliability of the circuit under test, under the influence of two major degradation mechanisms, namely Bias Temperature Instability (BTI) and Random Telegraph Noise (RTN). The circuit was implemented in a 32nm technology, which successfully places it within the current downscaling trends.

The final chapter deals with the acceleration of the HotSpot-5.0 software. This thermal simulator is used for describing the temperature distribution given the power traces, the thermal and geometric characteristics of an integrated circuit. It is of vital importance to keep the execution time as low as possible, without sacrificing any accuracy in the derived transient solution. After a series of profiling stages, we replaced a time consuming numerical method with a more optimal equivalent. The execution time of the software was greatly reduced with negligible accuracy degradation. This accelerated version has been used with the name Quick_Hotspot, to create a hierarchical thermal analysis tool. That way, the possible applications of thermal analysis, either at design time or at runtime have been further broadened.

1.2 Effects of BTI & RTN on the Performance of an SRAM Partition

Devices of older technologies exhibit a uniform degradation throughout their lifetime. In other words, one device is representative of the way all devices of the same technology behave under operating and/or stress conditions. However, while device dimensions reach nm lengths, the situation is completely different. We observe an extensive variability within the devices of the same technology. If we draw our attention to BTI and RTN, we observe that the threshold voltage evolves quite differently during the lifetime of deca-nanometre devices of the same technology.

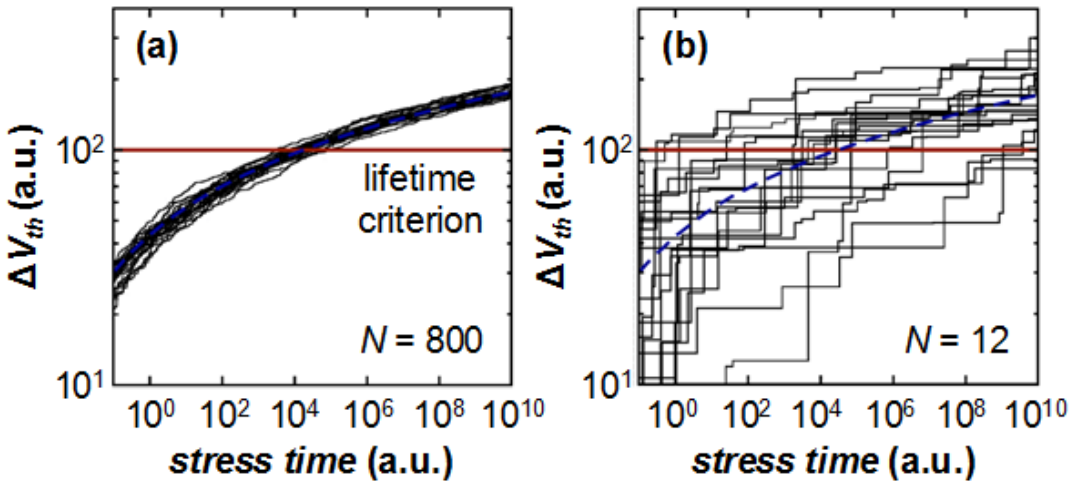

Figure 1.2.1: Expected threshold voltage degradation in devices of older technologies (a), in comparison to modern downscaled devices (b); the latter graph demonstrates the increased device variability of modern technologies [10]

As a result of time and workload dependent device variability, the reliability of embedded systems is greatly at stake. At this point it is usefully to define the two basic types of failures that can impede the reliability of embedded systems. A functional failure refers to an erroneous condition that causes the corruption of data that are transmitted or stored in an embedded system. On the contrary, parametric failures correspond to extreme fluctuations of the system's performance metrics, like its delay or its leakage energy. Degradation of a devices' threshold voltage may lead to either of these failures. However, the purpose of the text is to group circuit workloads based on the effect that they have on the performance metrics of a small digital system. Hence, the presented work is more focused on parametric reliability, rather than functional reliability.

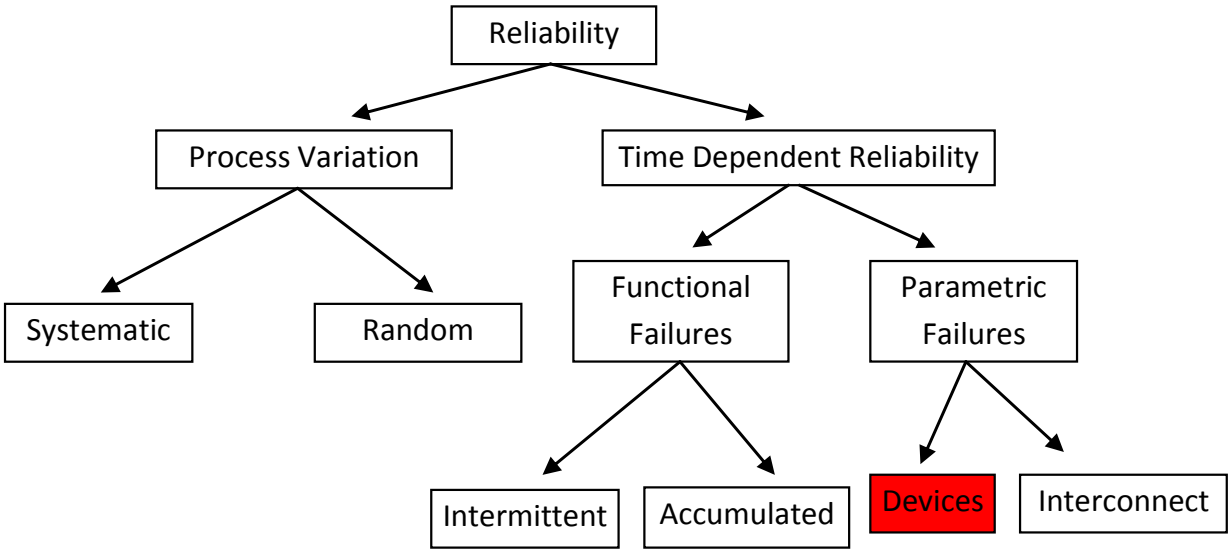

Figure 1.2.2: A guidance framework describing the landscape of reliability concerns; the focus of the current text is highlighted with red colour [3]

It goes without saying that parametric failures might cause functional hazards. For example, deadlines during embedded system activity may not be met because of parametric failures, thus causing crucial functional errors. It is important that we remove such tricky strides from our attention, since they increase the complexity of the reliability problem even further. It is also important that we remove from our scope any technology related variations, like the ones occurring because of random dopant fluctuations.

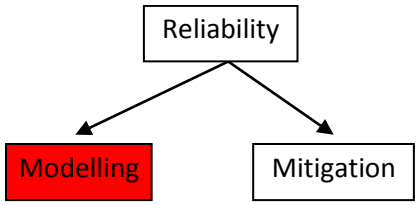

Figure 1.2.3: A guidance framework describing the landscape of possible approach towards reliability; the focus of the current text is highlighted with red colour [3]

Two basic approaches have to be maintained towards parametric reliability. On the one hand, there needs to be sufficient knowledge of the underlying mechanisms, as well as an understanding of their effects to circuit performance. The most elegant and reusable technique to achieve the above is circuit simulations. On the other, having created the required knowledge background, one needs to start reflecting on mitigation technique, which brings us to the mitigation approach. The current work is solely conentrated on simulations. However, the post processing of the simulation output, is perfrormed with mitigation techniques in mind. More analytically, the grouping of circuit workloads, based on the effect that they have on BTI and RTN behaviour is a key aspect that may enabled mitigation techniques based on workload tuning.

If we look into state of the art approaches towards modelling or mitigation of BTI or RTN, we see fairly crude approaches. Many attempts average out the device stress, using the concept of signal probability. Such simplifications are made at device level; hence no real workload dependency exists in these models. In other cases, there is poor incorporation of the BTI relaxation phase (when stress is removed from the device). As a result, the BTI estimation is overly pessimistic, thus leading to over-constrained designs. Finally, to our knowledge, there exists no unified approach towards BTI and RTN simultaneously.

Our proposed approach accounts for the workload that is applied to each device and monitors the occupancy of each oxide defect separately. The defect time scale interval is wide enough to incorporate both BTI and RTN behaviours. Finally, it is possible to decouple the stochastic component of the defect capture and emission from the model's workload dependency. Hence, our approach appears to be a suitable tool with which to explore parametric reliability in a realistic and detailed workload dependent way. What we essentially demonstrate is that our model contains a workload memory. In other words, workload that is applied to a circuit in the past will dictate the effect of BTI and RTN on the circuit performance at present and in the future. This quality of our approach enables BTI and RTN mitigation techniques workload tuning.

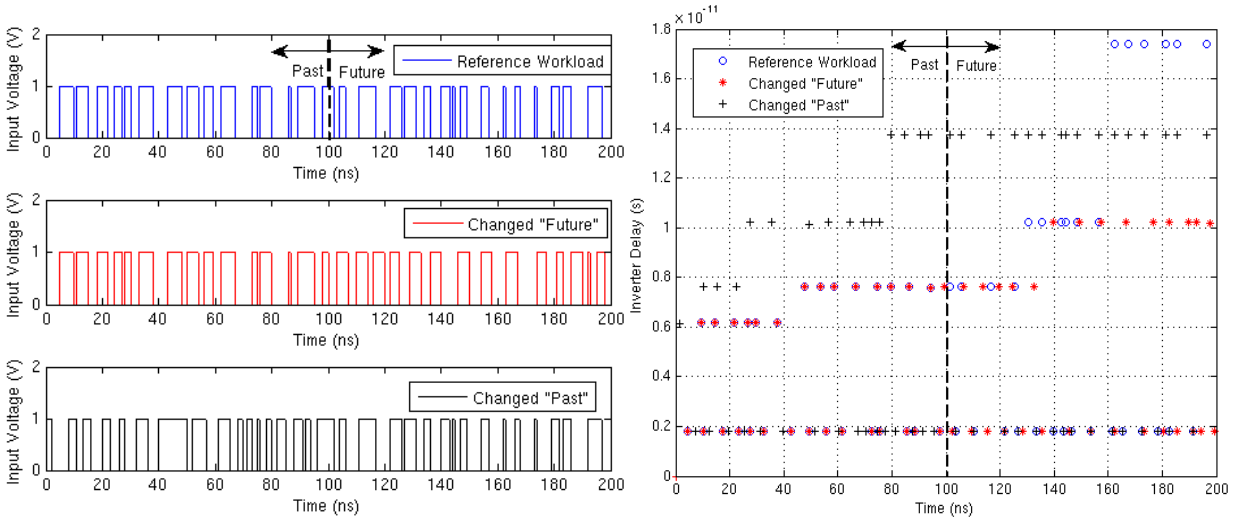

Figure 1.2.4: Runtime changes in the workload of an inverter, propagate to changes in the delay under the influence of BTI and RTN; increased delay variability was observed in SRAM partition simulations following the same principle [24]

In view of the increased variability of nm devices, an atomistic approach was developed to describe the activity of the oxide interface defects that is responsible for the two above mechanisms. That way, BTI and RTN can be explored in a time and workload dependent way. The principles of this atomistic model are described in Chapter 2. This atomistic approach has been software supported at device level, thus enabling the simulation of much larger circuits. These software support foundations are discussed in Chapter 3, as do the state of the art approaches for evaluation purposes. The complete framework for the simulations of the SRAM partition is presented in Chapter 4. Realistic workloads were used, in order to explore the impact that workload has on the BTI and RTN effect on the circuit's performance. As a result, realistic data streams have been extracted from a real application involving an FIR filter. These data have been parsed and applied to the SRAM partition. The monitoring of key performance metrics, like the read delay of the memory or its leakage energy, made it possible to observe performance fluctuations at runtime. Finally, an attempt was made to cluster different workloads, depending on the impact that the formers have on the memory's performance.

1.3 Acceleration of the Transient Solution on HotSpot-5.0

The advances in the semiconductor industry have lead to the incorporation of many functional blocks into single integrated circuits. A major issue that ensures the projected lifetime of these devices is the smooth distribution of temperature across the integrated circuit. The creation of hot spots on its surface is a major hazard that puts the chip's functional and parametric reliability at great stake. Furthermore, good knowledge of the chips thermal profile will impose realistic constraints for the design of the chip's packaging. Apart from thermal simulations at design time, a runtime extraction of the temperature distribution can enable workload tuning in order to alleviate hot spot creation across the chip's surface. In any case, a fast and yet accurate thermal simulator can be a useful tool to facilitate either post- or pre-silicon thermal analysis.

The academic society is quite familiar with the thermal simulator called HotSpot. We will focus on the fifth version of this software and will try to explore any further room for trade off between execution time and accuracy. The first stage of this exploration is the profiling of the code, using timing functions of reduced overhead. The profiling results will reveal the parts of the code where we need to concentrate our optimization efforts. A search through available numerical methods allows us to replace CPU intensive functions with optimized equivalents, thus leading to an overall acceleration of the thermal simulator. Finally, a realistic application of the accelerated tool presents hierarchical thermal profiling that can be equally used at design time or at runtime. This work is discussed in Chapter 5.

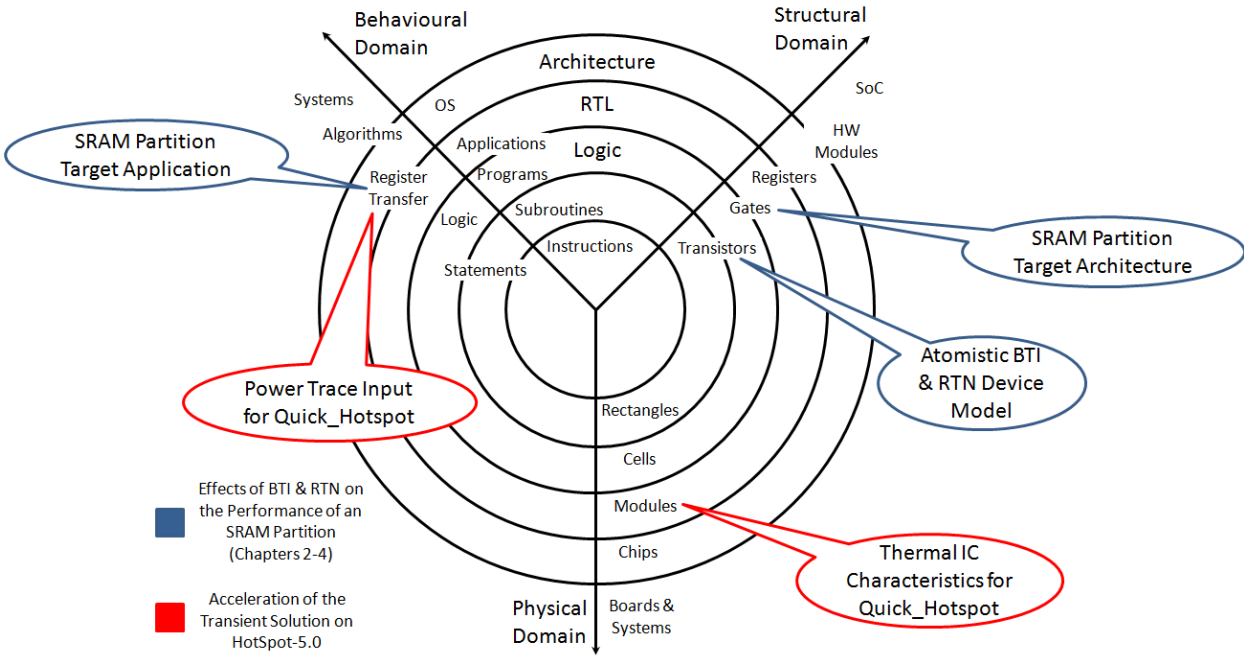

Figure 1.3.1: Placement of the work presented in current text, within the design abstraction levels described by the Gajski-Kuhn chart (based on figure found in [33])

1.4 Placement within the EDA Context

It is obvious that the contributions of the current text are closely related to the area of Electronic Design Automation (EDA). Be that design time or runtime simulators, we aim at the realistic representation of operation parameters for integrated circuit subsystems. Hence, it would be interesting to reflect upon the design domains where the contribution of this text resides. The first part of the text is connected on both the device and architecture level. The BTI and RTN atomistic model belong explicitly to the transistor level of the electronic design landscape. The software support of this model involves an enhanced transistor model that is able to monitor the defect activity of each simulated device. On the other hand, the test circuit we are using corresponds to a target architecture and target application. The former refers to the partition's ecosystem, within the system where the memory is implemented. The latter refers to the realistic activity of its memory cells.

Chapter 2: Device Level Fundamentals

2.1 Introduction

In this chapter, we will create the device level foundations required for any further discussion about simulations of much larger circuits. Since the focus of the current text is Bias Temperature Instability (BTI) and Random Telegraph Noise (RTN), a brief presentation of these mechanisms is required. It is also interesting to inspect the state of the art approach towards these mechanisms. In view of the assumptions made in these models, it will become evident that a need is present for an approach that follows the workload dependent reality more closely. The atomistic approach that is used in the current text really does fill the gap that exists in the current BTI and RTN modelling landscape. However, before delving into the atomistic model, we need to briefly present the experimental work upon which the former is founded.

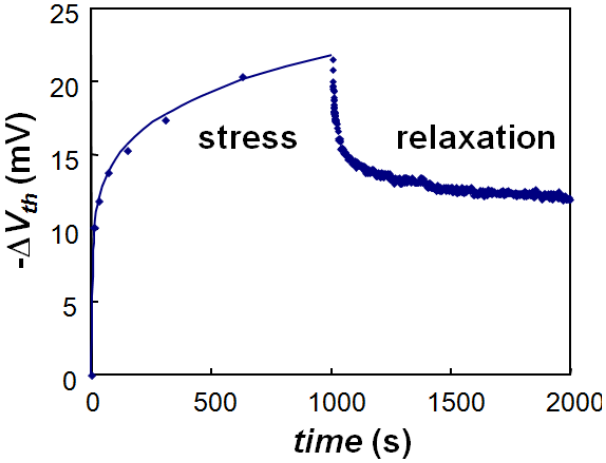

Figure 2.2.1: Example of the alteration between stress and relaxation phases, because of NBTI on a pMOSFET

2.2 Qualitative Description

Among other effects, BTI manifests itself as an increase of the absolute value of a device's threshold voltage under specific stress conditions. In case of a pMOSFET, the stress phase is defined as a negative gate voltage, with respect to the drain and source voltages (NBTI). A complementary definition exists for nMOSFETs, where the stress is defined as a positive gate voltage with respect to the drain and source voltages (PBTI). When the stress is removed (i.e. the voltage polarity is reversed), a partial recovery of the threshold voltage is observed. This mechanism is intensified at increased operating temperatures.

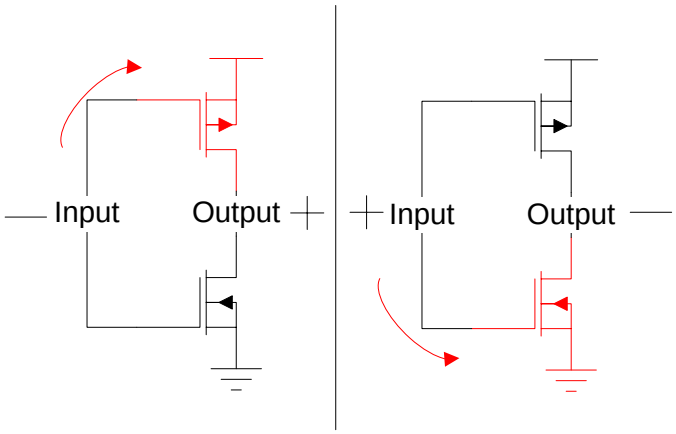

Figure 2.2.2: BTI stress phases in a simple inverter circuit; the affected branch is highlighted accordingly

Even if various models exist for BTI [19], no universally accepted approach exists towards its physical background. However, it appears that the phenomenon is related to the creation and activity of carrier traps.

As for RTN, it is manifested as a two state noise in the drain current of a device [14], which obviously propagates to the threshold voltage. The two state nature of this phenomenon also points towards the activity of carrier traps.

2.3 Insulator

As illustrated in [29], the term *threshold voltage* may have various meanings in the related literature. This thesis is not focused on the physical background of the MOSFET operation modes. Hence, we will restrict only to a high-level qualitative definition. We will define the threshold voltage as the V_{gs} that is required for strong inversion to commence. A quantitative equivalent can be found in [7] or [29] and is given by the following formula.

$$V_{th} = V_{FB} - 2\varphi_F - \frac{|Q_B|}{C_{ox}}$$

In the above expressions, V_{FB} is the flatband voltage, φ_F is the Fermi potential, Q_B is the depletion region charge and C_{ox} is the oxide capacitance. Threshold voltage is a parameter that may undergo fluctuations during the device operation. Some of the fluctuations remain unnoticed in a circuit's performance, whereas others lead to serious performance degradation. The main focus of the current text is the performance impact of V_{th} fluctuations caused by BTI and RTN. According to [7] the flatband voltage is a key quantity to threshold voltage variations.

$$V_{FB} = \varphi_{MS} - \frac{Q_f}{C_{ox}} - \frac{Q_{it}(\varphi_S)}{C_{ox}}$$

In the above equation, Q_f is the fixed trap density and Q_{it} is the trap density, which appears to be dependent on the surface potential φ_S. φ_{MS} is the difference between the bulk potential and the potential of the gate material, as described in [29]. Based on [7], threshold voltage variations may be caused by changes in either of the Q_f or Q_{it} charges. If we focus on the SiO_2/Si interface, we can identify bonds between silicon and various other atoms that exist in that area (oxide, silicon or hydrogen). The default crystal organization refers to bonds between Si and SiO_2. However crystal mismatches may occur.

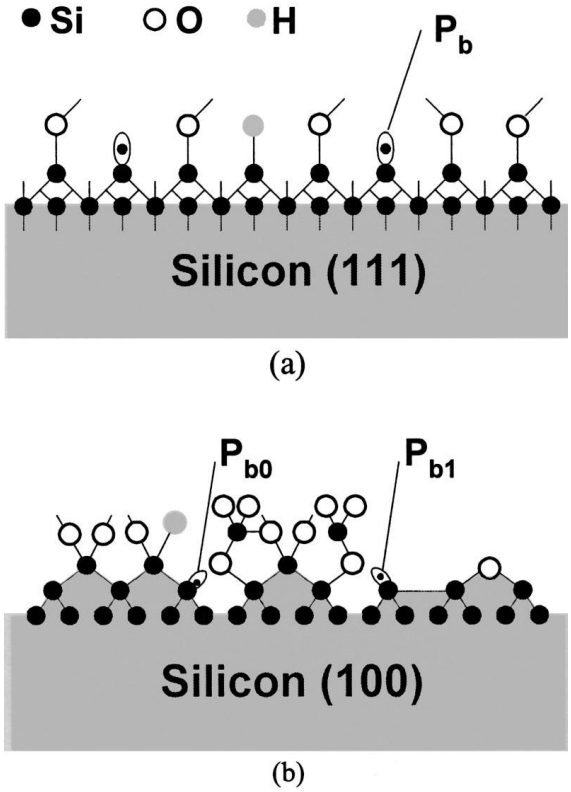

Figure 2.3.1: Interface traps for different wafer orientations [7]

Interface traps correspond to unpaired electrons of Si atoms [7]. Depending on the orientation of the wafer, we may distinguish different kinds of interface traps, namely P_b centers for (111) orientation and P_{b1} and P_{b2} centres for (100) orientation. These interface traps are carrier recombination/generation centres. Hence, the minority carriers of the channel are recombined in these centres, thus reducing the I_{DS} current during the device operation. A reduction of I_{DS} is translated to an increase of the absolute value of the threshold voltage. Depending on the position of the interface trap in the band gap, its behaviour may resemble either an acceptor or a donor; hence the different effect on either p- or n- enhanced channels. Larger numbers of interface traps may occur from the destruction of bonds between silicon and other atoms, like hydrogen.

If we move towards the bulk of the insulator, we come across what is referred to as border traps. The principle describing their behaviour is similar to that of interface traps, namely they also exchange carriers with the substrate of the device [38]. The nature of border traps is reported to be bonds that are vacant from oxygen atoms [39].

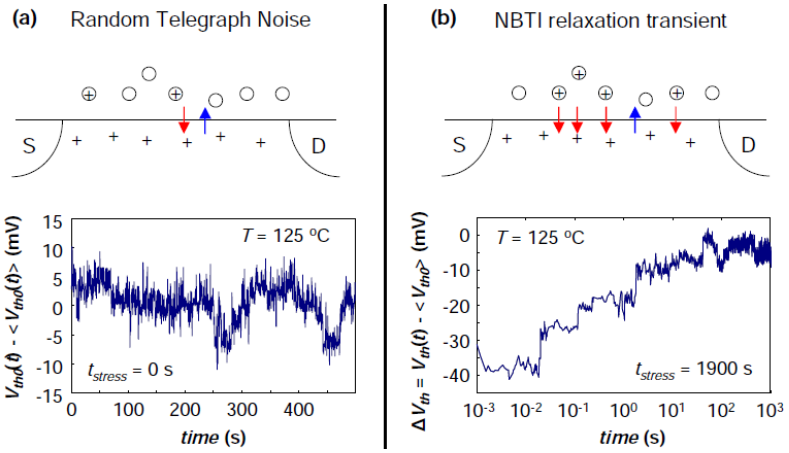

Figure 2.3.2: Connection between RTN (a) and NBTI(b) with oxide interface activity, as demonstrated in [12]

In the creation and recombination activity of such traps, originate the mechanisms of RTN and BTI. In the first case, the carrier exchange is brought to some sort of equilibrium, whereas in the second case, carrier emission is suppressed in favour of carrier capture. Depending on the frequency I_{DS} of the recombination/generation in the border traps, we can distinguish either fast or near permanent effects on V_{th}.

However, state of the art models on BTI, tend to focus more on the interface bond breaking, rather than the effect of trap activity.

2.4 The Reaction Diffusion Model

One of the predominant approaches towards BTI is the Reaction Diffusion (RD) model, which is mainly utilized for the explanation of NBTI. This phenomenon has been considered as a major reliability issue from a lot of researchers [18]. The basic principle of the model is that during the device operation, Si-H bonds

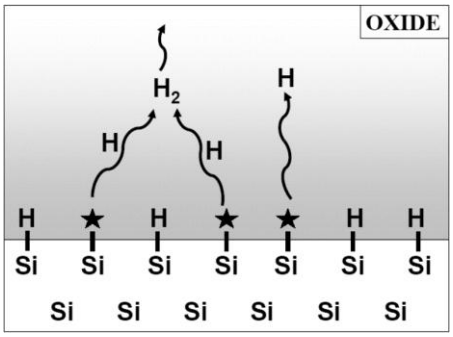

Figure 2.4.1: Artist's impression of the Si-H bond breaking and the diffusion of H towards the bulk of the gate stack [1]

gradually break, since they are weaker that the SiO_2-Si counterparts. Carriers are trapped in the dangling bonds, thus reducing the I_{DS}, hence the V_{th}. The interface trap creation is presented to be proportional to the lifetime of the device in the time scale, as demonstrated in [1]. According to the RD model, the number of interface traps N_{IT} at any given time is given a combination of a diffusion component and a reaction component.

$$\frac{dN_{IT}}{dt} = \underbrace{k_F(N_0 - N_{IT})}_{\text{Diffusion}} \underbrace{-k_R N_H(0) N_{IT}}_{\text{Reaction}}$$

Assuming $\left\{\frac{dN_{IT}}{dt} \ll 1 \text{ and } N_{IT} \ll N_0\right\}$, the above equation can be written as $\frac{k_F N_0}{k_R} \approx N_H(0) N_{IT}$.

The diffusion part of the above equation contains the Si-H bond dissociation constant k_F and the total number of Si-H bonds N_0. The inverse part of the phenomenon, i.e. reaction, contains the reverse Si-H annealing constant k_R and the diffused H that is still close enough to the SiO_2/Si interface $N_H(0)$. The rest of the hydrogen is considered to diffuse away from the interface.

Depending on the form of the detached hydrogen (atoms, molecules, ions), we can use either diffusion (Fick's Law) or drift equations (dependent on the field E_{ox}) [1]. Based on these equations we can distinguish the diffusion of two types of particles.

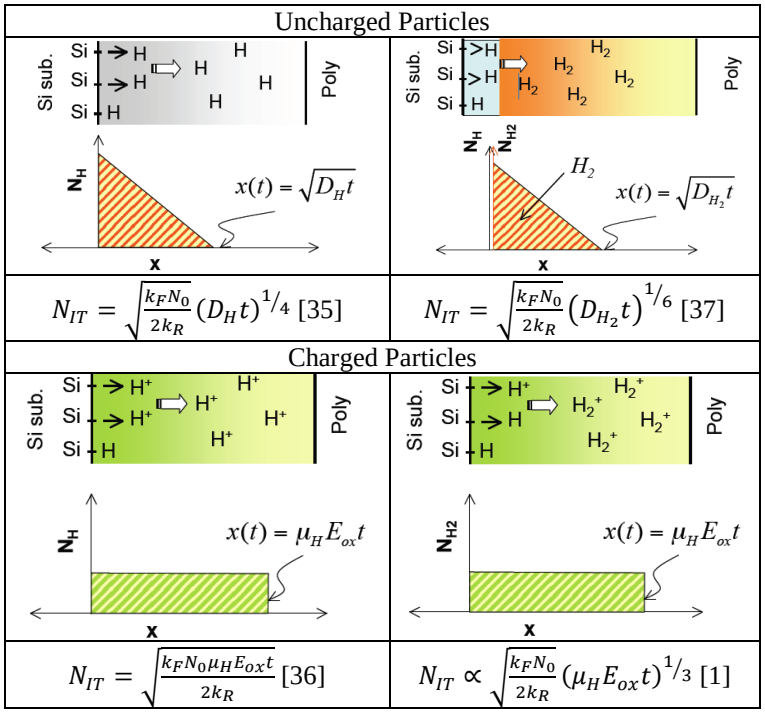

Figure 2.4.2: Possible particles diffused in the gate bulk [1]

$$\begin{cases} \text{neutral particles (H atoms, } H_2\text{): } \dfrac{dN_{IT}}{dt} = D_H \dfrac{d^2 N_H}{dx^2} \Rightarrow N_{IT} = \displaystyle\int_0^{\sqrt{D_H t}} N_H(x,t)dx \\ \text{charged particles } (H^+, H_2^+)\text{: } \dfrac{dN_{IT}}{dt} = N_H \mu_H E_{ox} \Rightarrow N_{IT} = \displaystyle\int_0^{\mu_H E_{ox} t} N_H(x,t)dx \end{cases}$$

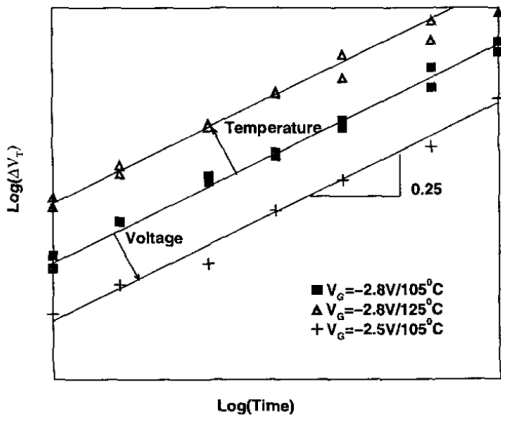

Figure 2.4.3: Evident $^1/_4$ exponent dependence in threshold voltage shifts throughout the device lifetime; temperature dependence is also visible [37]

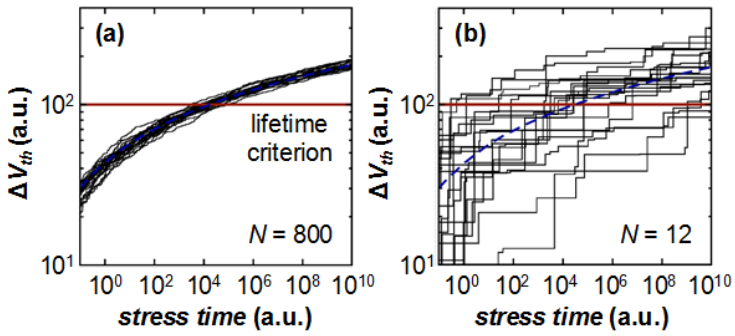

Figure 2.4.4: Expected threshold voltage degradation in devices of older technologies (a), in comparison to modern downscaled devices (b); an approach based on reaction and diffusion mechanisms is deemed unrealistic for the second case [10]

Having established the change in the number of interface traps N_{IT}, we can determine the impact on V_{th} based on the following equation [21], which can be corrected for mobility degradation with the parameter m.

$$\Delta V_{th}(E_{ox},t) = (1+m)\frac{qN_{IT}(E_{ox},t)}{C_{ox}}$$

ΔV_{th} or ΔN_{IT} curves across the device lifetime, point towards hydrogen atoms as the core reason for threshold voltage degradation. The main reason is the $^1/_4$ exponent that dominates these results. A cross examination with the activation energy which is required for this process to take place, further supports the H atom diffusion claim. The amorphous nature of the SiO_2, may cause deviations from the initially expected exponent value [1].

We need to put the above perception of BTI to a technology context. Devices of older technologies have sufficient size and a large number of defects, thus exhibiting uniform reliability behaviour [12]. As the device dimensions decrease, the stochastic defect behaviour becomes gradually more evident. As a result, an approach like the one presented above appears to fit less the downscaling trend of recent technologies. Finally, a closer look at the activity of border traps will reveal that the mechanism of BTI resembles greatly that of RTN and 1/f noise in a way that the RD model is unable to interpret [40]. From the above realizations, it appears imperative to move towards a more atomistic modelling of BTI.

2.5 Other Causes of Transient V_{th} Variations

Since the focus of the current text is parametric reliability under transient V_{th} variations, it is of vital importance to include in our analysis any mechanisms that cause V_{th} variations. Once identified, we want to remove them from our final degradation model. Indeed, if they do not

cause any performance degradation, we should not incorporate them. Such a mechanism is Drain Induced Barrier Lowering (DIBL). During various simulations that have been performed, we have come across the modulation of the threshold voltage of a device by the V_{ds}. Such a modulation can be noticed in the simulation of a simple SRAM cell. We focus on the upper left pMOSFET. Its gate voltage is obviously $\bar{Q}$, whereas the drain voltage is Q. A swing following the voltage at Q is visible in the simulation results. The various spikes of the voltage at $\bar{Q}$ are as a result of the cell's switching (since the read and write activity of the cells is simulated). It is interesting that such dynamic alterations are propagated to the threshold voltage as well.

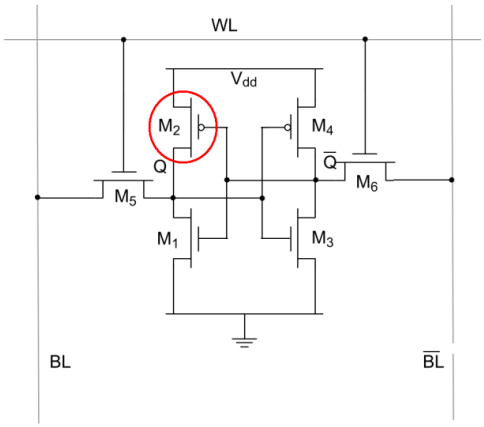

Figure 2.5.1: Circuit of the SRAM cell in question

The reason for the above modulation appears to be DIBL. As explained in [29], the drain voltage causes variation in the threshold voltage of a device. As observed in the simulation results, upon removal of the increased drain voltage, the threshold voltage recovers completely. As a result, DIBL does no accumulate any degradation in the transient response of the device. Given that the drain and gate voltages are complementary for the cross coupled inverters of a traditional cell, it is important to be able to distinguish V_{th} variations that are caused by drain voltages and that have no effect on the performance of the device. Similar effects of DIBL can be identified in the results presented in [10].

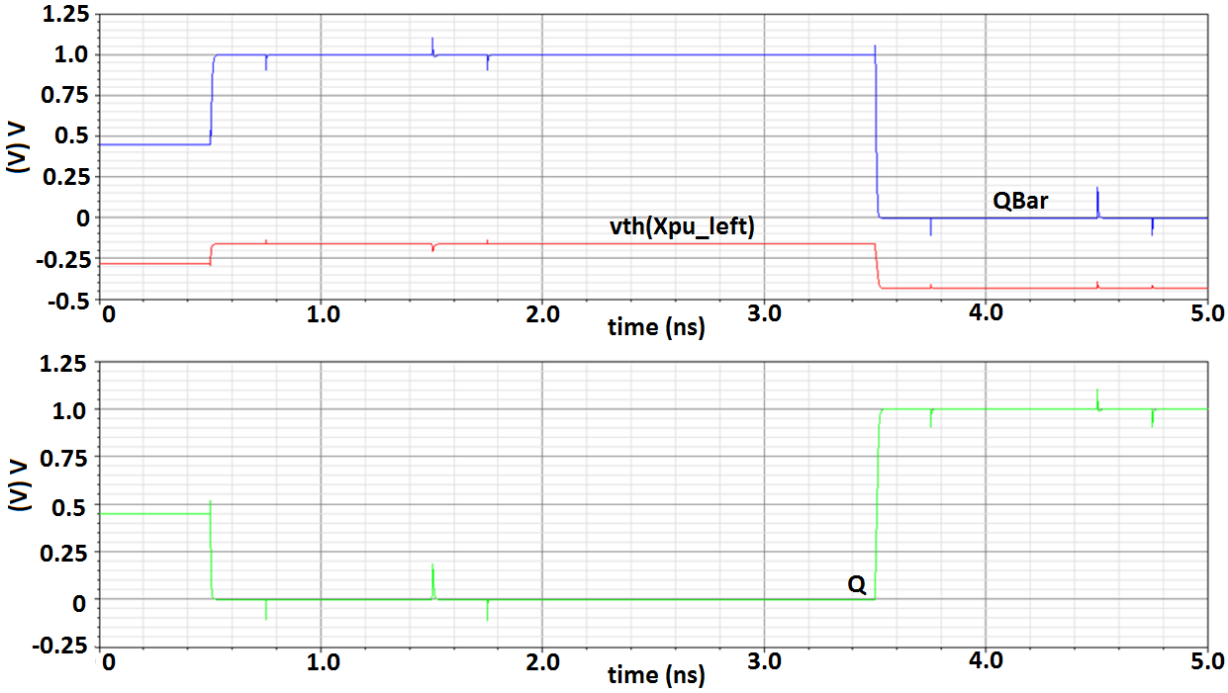

Figure 2.5.2: Simulation output that demonstrates V_{th} modulation due to DIBL

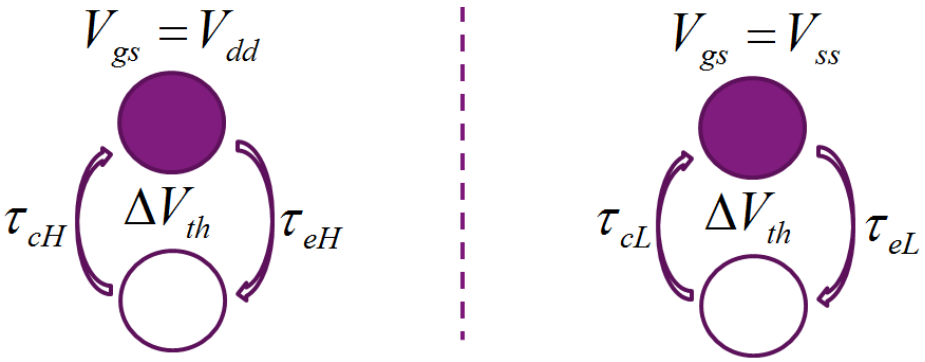

Figure 2.6.1: Conceptual schematic of the two state atomistic defect model, based on [10]

2.6 Atomistic BTI Model

A suitable two state model for defects found in the oxide interface has been presented in [10]. An oxide defect can be either occupied or not. It also has a pre defined impact on the device's threshold voltage, ΔV_{th}.

The rate at which this defect captures and emits minority carries is determined by time constants. Since we are handling digital systems, it is plausible to assume a pair of time constants (for a capture or an emission event) for different logic level (high or low), hence $\{\tau_{cH}, \tau_{cL}, \tau_{eH}, \tau_{eL}\}$. Time constants with small values correspond to frequent capture and emission of minority carriers, hence RTN. Larger time constants correspond to quasi permanent capture, hence BTI. A single device can be assumed containing an arbitrary number of defects, each one having a different set of the four aforementioned constants.

Based on the experimental results of [27], we possess the distributions that describe the values $\{\tau_{cH}, \tau_{cL}, \tau_{eH}, \tau_{eL}, \Delta V_{th}\}$ as well as the average number of defects for a given devices. The time constants have been shown to distribute uniformly across the logarithmic scale [11]. As a result, an interval between 10^{-9}s up to 10^{9}s would be wide enough to incorporate both RTN and BTI behaviours. The trap density can be either extracted from the data presented in [27], or can be adjusted by the user so that the above time constant interval is properly accounted for (traps with all kinds of time constants within the interval). Furthermore, we can assume scaling rules, to apply the same distributions to devices of different dimensions than the ones of the measured MOSFETs. Finally, the distribution of average ΔV_{th} can be extracted directly from [27], both for pMOSFETs or nMOSFETs.

Given a set of defect parameters and a time step equal to Δt, we can define the probability of it capturing or emitting a minority carrier based on the following equation [10].

$$P_{p,v} = \frac{\tau_{\bar{p},v}}{\tau_{e,v} + \tau_{c,v}} \left\{ 1 - exp\left[-\frac{1}{\tau_{e,v}} + \frac{1}{\tau_{c,v}} \right] \Delta t \right\}$$

Based on this assumption it becomes possible to incorporate the defect activity in an in situ process that monitors the occupancy of a device's defects (i.e. their state and the probability of their switching to another state). Further information regarding the software support of this idea can be found in Chapter 3.

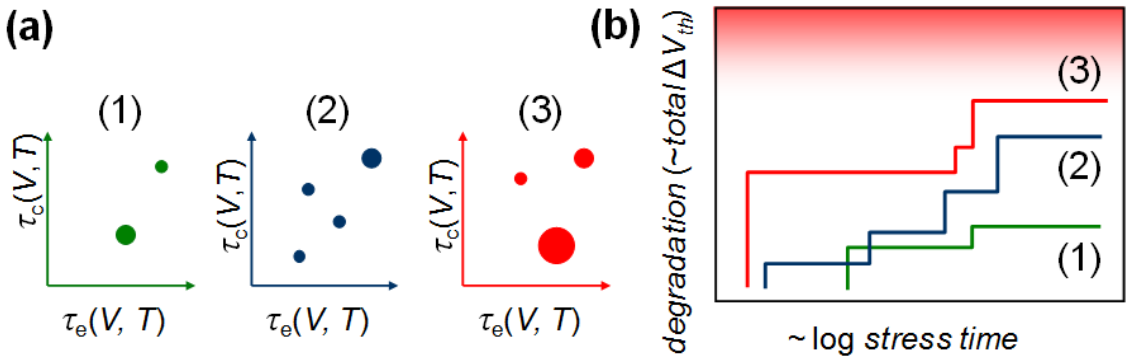

Figure 2.6.2: Simplified diagram illustrating the two state defect model; different devices are populated with defects of sets of $\{\tau_{cH}, \tau_{cL}, \tau_{eH}, \tau_{eL}, \Delta V_{th}\}$ (a); this variability propagates to the V_{th} degradation during the device lifetime [12]

2.7 Conclusions

In this chapter, we have presented the device level fundamentals that are necessary for a detailed BTI and RTN simulation. We have looked into the most popular approach towards BTI and identified its weaker point, namely its inability to account for devices with few defects and for workload dependence, because of its purely statistical averaging nature. In view of this insufficiency, we presented the novel atomistic approach towards BTI and RTN. Being supported by experimental data, the latter allows a partly stochastic - partly deterministic monitoring of the oxide defect occupancy. By assigning a set of defect parameters $\{\tau_{cH}, \tau_{cL}, \tau_{eH}, \tau_{eL}, \Delta V_{th}\}$ to an arbitrary number of traps of a single device, we can simulate BTI and RTN throughout the device lifetime. This simplified two state model can effectively be software supported, as discussed in the next chapter.

Chapter 3: Circuit Modelling

3.1 Introduction

The purpose of this chapter is to present the work that has enabled the creation of a complete parametric reliability simulation framework. The simulations of an SRAM partition are enabled by the software supported BTI and RTN model that is presented in the previous chapter. This chapter will present the components of this software support. It is also useful to reflect briefly on the state of the art simulation approaches. That way, it will become obvious that the proposed approach is both novel and meaningful for modern downscaled technologies.

3.2 RelXpert

During the readings for state of the art reliability simulation approaches, one is bound to come across the industry standard tool called RelXpert. This tool focuses on the simulation of NBTI and Hot Carrier Injection (HCI).

From the available documentation, it is clear that this tool shares a similar goal to the scope of the current work. The actual intension of the RelXpert-related researchers is quite self-explanatory: "The key is to find an NBTI age formula such that the device degradation due to the NBTI effect and the NBTI age can be physically established and modeled" [34].

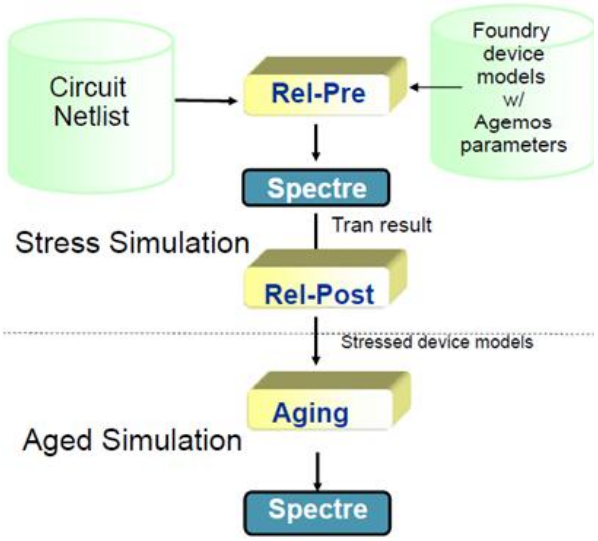

Figure 3.2.1: Conceptual flowchart of the RelXpert simulation methodology [31]

As for the algorithm itself, RelXpert performs two iterations of the same simulation, one for a fresh netlist (called *Stress Simulation*) and one for the netlist including the degraded components (called *Age Simulation*) [34]. The first round will determine the stress of the circuit's components. Based on that stress, some key parameters of typical MOS models (like BSIM) are changed, thus mirroring degradation. As a result the circuit devices become "aged". The second simulation will yield the behavior of the "aged" circuit. It appears that this tool averages out the degradation of sensitive parameters. The theoretical foundation of RelXpert is clearly the RD model. RelXpert uses averaging of statistically sampled data to degrade some parameters of typical MOS models (like BSIM) [13]. It is noteworthy that the parameter shift performed for "aging" purposes is fixed in the time scale after the first simulation and does not change during the second one.

An attempt to gradually simulate the device degradation is also reported [31]. The idea is to split the stress duration into intervals and perform a pair of simulations (*Stress* and *Age Simulation*) at the end of each interval.

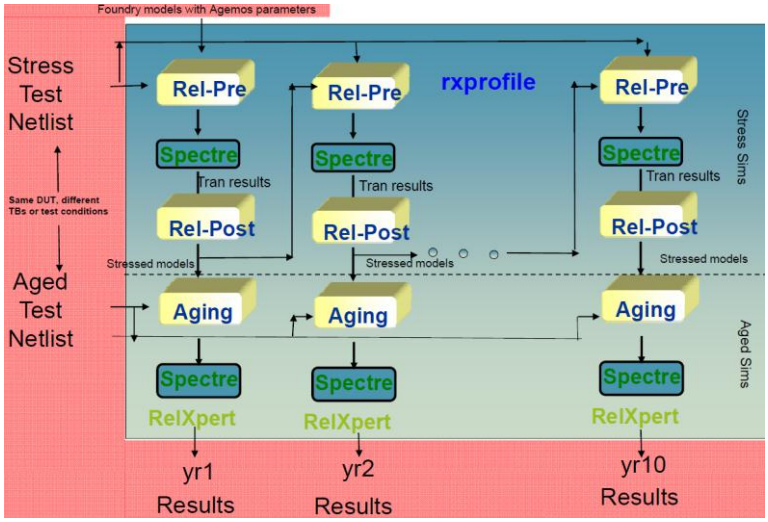

Figure 3.2.2: Flowchart of gradual NBTI simulation using the RelXpert framework [31]

3.3 Other Simulation Approaches

Both in terms of modeling and mitigation, we observe a tendency to average out either the stress of the devices or their degradation. This means that any consequent simulation approach will be unable to follow the degradation mechanisms in a transient way. In the current section we will present some noteworthy approaches towards reliability simulation that create the landscape of modeling (and mitigation) of mechanisms like BTI or RTN. Since the current text emphasizes on the reliability of memory structures, our focus will be maintained accordingly.

We observe that the gradual NBTI simulation framework based on RelXpert strides over very vast intervals of circuit lifetime, proportional to a year [31]. An approach that employs gradual modeling of increased granularity is that of [19]. This approach provides insight to the spatial and temporal reliability variations of a circuit under test. The degradation mechanisms that are studied are NBTI and HCI.

The behaviour of the circuit under test is coded into a vector of performance parameters (e.g. gain, bandwidth, etc.), $\boldsymbol{P}^i$. The degradation of each device is calculated based on the degradation under hypothetical DC stress. This degradation is extrapolated over stress duration T_{str} and included in "aged" device models. The behavior of the degraded circuit will now be

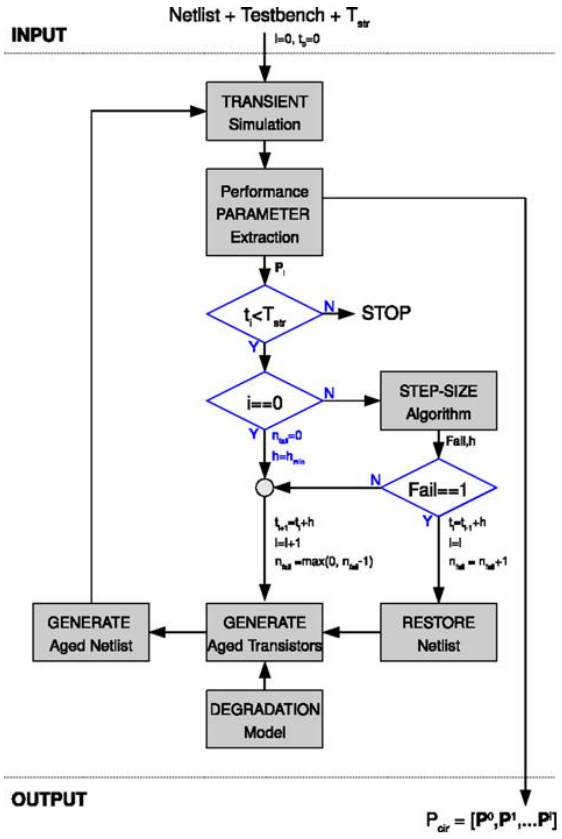

Figure 3.3.1: Flowchart of NBTI and HCI simulation framework presented in [19]

$\boldsymbol{P}^{str}$. In order to enable the simulation of analog or mixed signal systems, the framework splits the T_{str} interval in steps with duration equal to t_i. For each step, the above set of computations is performed and the respective "aged" circuit is fed as input to the next iteration for t_{i+1}. Output of this iteration is a matrix $\boldsymbol{P}^{cir} = [\boldsymbol{P}^0 \quad \boldsymbol{P}^1 \quad ...]$ of the performance parameters for the circuit under test, over T_{str} lifetime. In this approach, NBTI degradation under time varying stress is calculated based on integrated equations derived for DC stress voltages.

Up to this point we have seen that no approach accounts for the past stress of a device. However, based on the stress and relaxation alterations, it appears vital to incorporate the past stress of a device when considering the evolution of BTI degradation. An approach that sufficiently addresses this point can be found in [15] and mainly addresses NBTI. Aiming at a reduced computational overhead, this work differentiates from the usually employed RD model. It also avoids numerical simplifications, since it does not focus on the microscopic mechanisms of NBTI. On the contrary, it treats the degradation $D(t)$ of a device as a convolution of a stress function $g(t)$ (incorporating node voltages, temperature and other NBTI parameters) with the response function of this degradation mechanism $h(t)$ as follows.

$$D(t) = h(t) \otimes g(t) = \int_0^{\tau} h(\tau) \quad \underbrace{g(t-\tau)}_{\text{memory component}} \quad d\tau$$

Assuming a unit step $g(t)$, one can easily obtain the system response $h(t)$, solely due to NBTI degradation. Sadly, no further information is given regarding the way that the contributors perceive the $h(t)$ degradation response. The latter is simply left as a user defined parameter.

The convolution concept is the one that represents some sort of stress and degradation memory. It is added in parallel to the existing RelXpert framework (cyan blocks of Figure 3.3.2). This work is compared to industry standard reliability tools (RelXpert) and achieves a more realistic simulation, especially for the relaxation part of NBTI. After a simulation of a netlist under test, the voltage waveforms are extracted and the rate of aging is estimated at each simulation step. It is important to note that there is no in situ simulation of NBTI. In contrast, a feedback loop is required, in order to correct the simulation output according to the convolution based model.

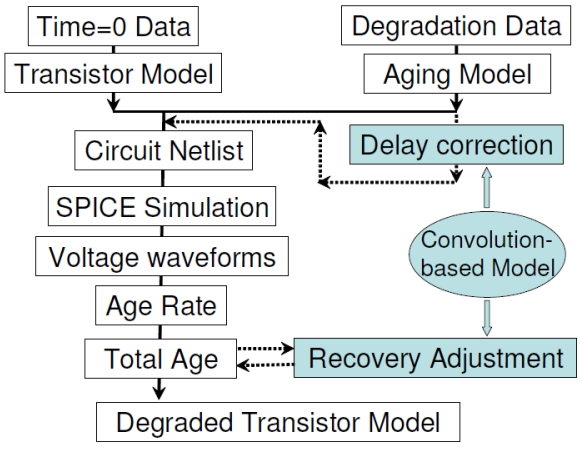

Figure 3.3.2: Simulation framework of the approach presented in [15]

Another approach that is more focused on the target structures of the current text (i.e. memories) is a mitigation method of NBTI induced SNM degradation, as presented in [2]. This work emphasizes on the functional reliability of SRAM cells, namely the ability of the cell to keep its

value. Fluctuations of the cell's delay because of NBTI activity are referred to as insignificant. The device stress is perceived through the concept of signal probabilities, i.e. the ratio of zeros and ones that are found in the simulated digital system.

This work employs the metric of SNM (static noise margin), which is the minimum DC voltage that can cause the cell's stored value to flip. Variations of the threshold voltage are implemented as current sources in industry standard circuit simulators. Based on the operating conditions, the signal probability and the idle time of a cell, the respective SNM is calculated.

The concept of signal probability is very well illustrated in [17]. In this work, we can identify the exact concept of the device stress being averaged out. For a digital circuit, an arbitrary bit stream is turned into an equivalent AC digital signal with the same ratio of high and low voltage duration. For a time step equal to t_0, the signal probability is defined as a simple ratio.

$$SP = \frac{p}{q}$$

The impact of NBTI on the threshold voltages comes from the solution of the reaction diffusion model, as presented in [16]. That way, it is possible to find the relationship between the signal probability and the threshold voltage impact after a specific duration of pre-defined stress and relaxation alterations.

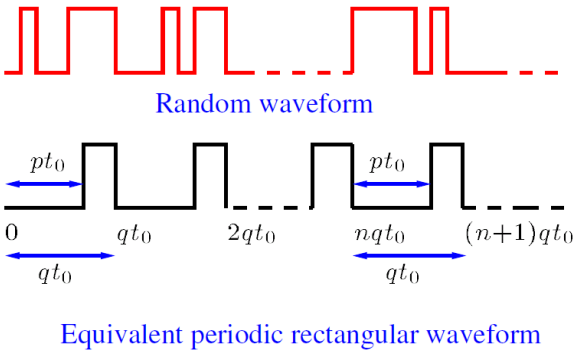

Figure 3.3.3: Illustration of the signal probability, as presented in [17]

A work that focuses on parametric reliability, namely the degradation of the performance metrics of a circuit instead of functional failures is [21]. This work concentrates on combinatorial circuits and studies the performance degradation in comparison to the threshold voltage degradation because of NBTI. Threshold voltage degradation is estimated based on signal probability and projections are made to describe the consequent degradation of the circuit's delay. This work also presents an algorithm that aims at reducing performance degradation, by altering the size of the employed devices. Surprisingly, the performance degradation is reported to have a weak correlation to the signal probability of the circuit's main inputs. The interpretation of this result is that the series connection of gates in combinatorial circuits tends to average out the impact of NBTI since, for a device which is under stress, there are other devices that follow and may be in the relaxation phase.

Another NBTI oriented approach that utilizes signal probability is [32]. This work aims at identifying critical gates in the presence of NBTI degradation. The effect of NBTI is assumed as a shift of the threshold voltage. The latter is calculated with an analytical formula.

3.4 What is missing in the State of the Art

Based on the above analysis, we can summarize the insufficiencies of the presented reliability simulation approaches, in the following points.

1. In many approaches, the stress of the simulated devices is averaged out using signal probability. The irregularity of the signals that are typically applied to devices of digital systems is not properly accounted for.
2. Statistical averaging occurs in early stages of the utilized models (for example at the device input level, as noted in the previous point). This course of action may reduce the credibility of the final method, since possible errors at core parts of the methodology tend to be accumulated in the final results.
3. The majority of the state of the art focuses on NBTI. There exists reduced work on PBTI.
4. In some cases, there is poor incorporation of the recovering part of BTI. This may lead to overly pessimistic NBTI estimation. From a designer's point of view, this may lead to over constrained and suboptimal designs.
5. The majority of the state of the art work tends to focus on functional reliability, rather than parametric reliability.
6. Some of the inspected approaches tend to be suitable only for specific types of systems (e.g. performance analysis on combinatorial circuits). Hence, the claims made about NBTI degradation and the effects of the latter cannot be generalized in other types of systems (e.g. memories).

The proposed framework emerges quite timely in the above landscape because it succeeds in accounting for detailed workload dependency. A detailed structure of the framework can be found in [24] and [41]. Neither the device degradation, nor their stress is averaged out at any of the proposed framework's steps. Furthermore, the framework incorporates both NBTI and PBTI. With the wide variety of time scale defect behaviour, RTN is also included. The two state defect kinetics that are employed account realistically for both stress and relaxation of the devices. The framework is compatible with any standard industry simulator and can be considered as an enhancement of typical device models (e.g. BSIM4).

3.5 Inverter Simulations

This section will present some brief simulation results using the aforementioned framework. We only consider NBTI, for simplification purposes. PBTI could also be easily included. The purpose of this section is to provide conclusive proof of the atomistic model's workload dependency. A more detailed test case with a wide variety of results and claims will be presented in the next chapter.

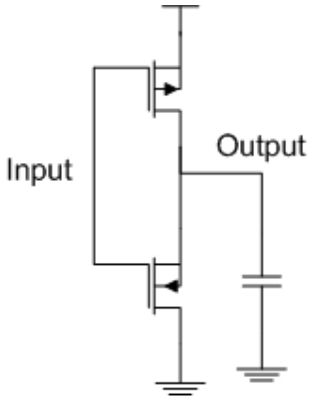

Figure 3.5.1: The circuit of a simple inverter

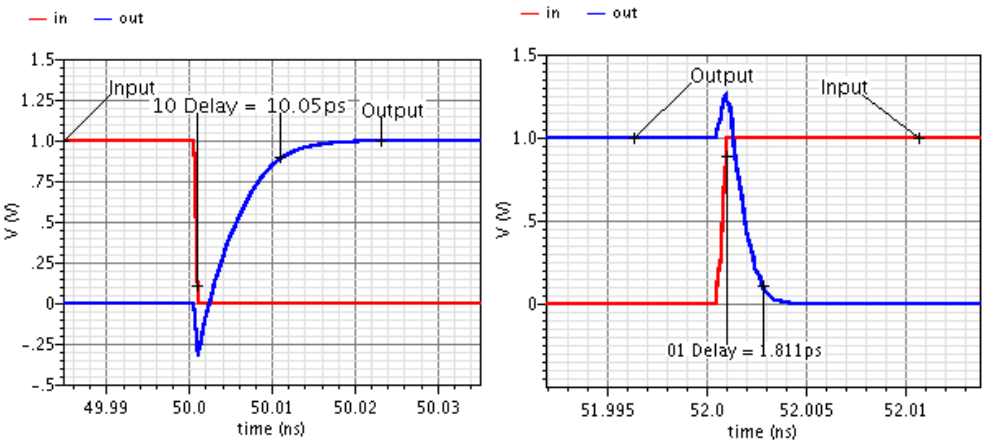

Figure 3.5.2: Instances of delay measurements on the inverter under test

We simulate a simple inverter and measure its delay. Since the pull-up branch is affected by NBTI, we expect larger delays for measurements during a 1-0 transition of the input. On the contrary, 0-1 transitions of the input are expected to lead to roughly steady and smaller delay measurements.

Based on a bit sequence of 200ns (Reference Workload), we change either the first ("Past") or the last ("Future") 100ns of the sequence. The results demonstrate a distinct proof of the model's detailed workload dependency, which is not averaged out by a pure stochastically based device model. The simulation output of the Changed "Future" is identical to the "Reference" output up to the point that the bit sequence is changed. In contrast, if we only change the "Past", the simulation output is entirely different.What we demonstrate here is that the atomistic BTI model has a deterministic workload memory. The observed NBTI effect at any time strongly depends on the specific workload that has preceded the current operation state.

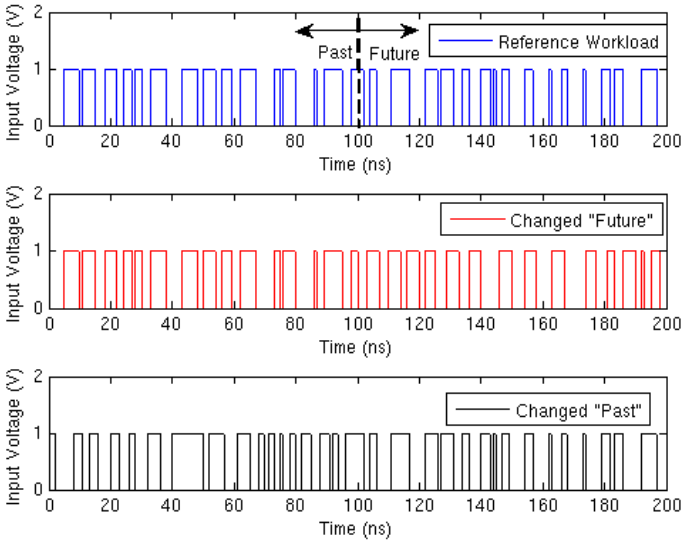

Figure 3.5.3: Runtime changes on the inverter's workload [24]

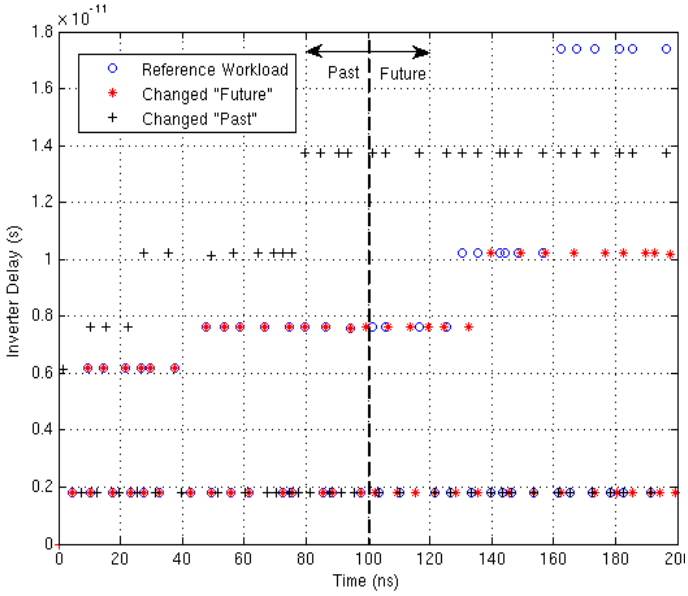

Figure 3.5.4: Delay measurements on the inverter [24]

3.6 Conclusions

This chapter presented the work that enables any further attempt of parametric reliability analysis. The created framework allows a software support of the atomistic defect model that includes both BTI and RTN in circuit simulations. This support is achieved through netlist annotation with oxide defect parameters, based on two-state kinetics. At runtime, the simulation is accompanied by a transient monitoring of the defects' occupancy. This allows a dynamic update of the devices' threshold voltage. It has been conclusively proved that if we suppress the stochastic component of the model, we can observe detailed workload dependency in the simulation results. Finally, comparison with other simulation approaches places the proposed framework in a much needed place of the state of the art landscape.

Chapter 4: Test Case

4.1 Introduction

The current chapter aims to present an operational example of a parametric reliability simulation framework. The circuit under test is a 32 bit SRAM partition. We need to inspect the circuit under test, the target architecture and the target application where the partition is supposed to be used. A specific part of the framework is dedicated to the creation of all the input and control signals. The input that is applied to the circuit under test was extracted by a realistic application. The simulation of the SRAM partition under the above input is accompanied by measurement of key performance metrics. That way, the analysis of the circuit's parametric reliability is achieved. Having addressed the above subjects, the current chapter will be completed with an extensive presentation of the simulation results.

4.2 Detailed Circuit Description

The organization of the circuit is based on already taped out circuits presented in [5] and [6]. We need to stress that the writing functionality is beyond the scope of the current text. Even though our test case supports the writing operation, the netlists are implemented in an "artifact" way. In contrast, the main focus of the test case is the reading performance of the memory. As a result, the respective circuitry is implemented as realistically as possible.

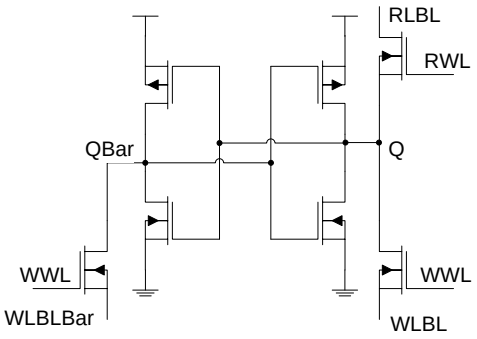

Figure 4.2.1: Circuit of the SRAM cell

The SRAM partition is composed of two identical groups of 16 bits. Naturally, each cell corresponds to a single bit. For the cell itself, a variation of the traditional 6 transistor organization is implemented. The SRAM cell requires two cross coupled inverters that hold the stored value at any given time. For the implementation of the reading functionality, we require a single reading local bit line (RLBL). The cell communicates with the RLBL through a pass transistor connected to its Q node. For the implementation of the writing functionality, we require two voltage sources that will impose the value to be written, as well as its complement. The voltage sources correspond to the nodes WLBL and $\overline{\text{WLBL}}$. Two pass transistors connect these nodes with Q and $\bar{Q}$. As a result, we require separate read and write word pulses, for the respective

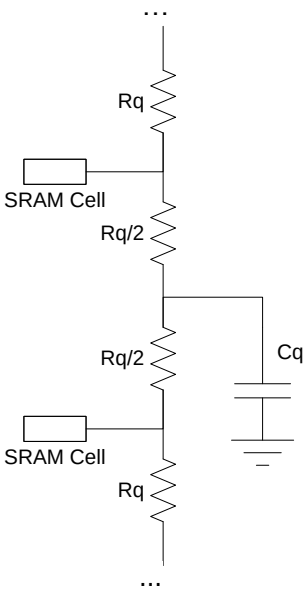

Figure 4.2.2: Equivalent of the RLBL

operations.

The RLBL is a large transmission line on which all 16 cells of a group are connected. In order to keep the circuit as realistic as possible, we need to populate it with parasitic resistances and capacitances. Hence, we will assume a parasitic resistance quantum between the nodes of two consecutive cells and a parasitic capacitance for the entire line. Obviously, only one cell from the group can be read at any given time. During the reading of a cell, the RLBL voltage swings down to V_{SS} if logic 0 is read. In other words, the RLBL is discharged through the pass transistor of the read cell. In the opposite case, the RLBL is kept to its pre-charge voltage.

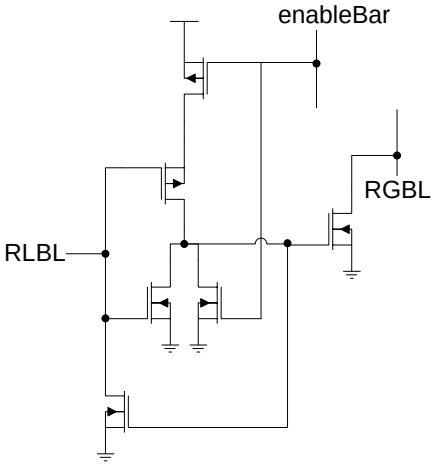

Figure 4.2.3: Circuit of the Read Buffer

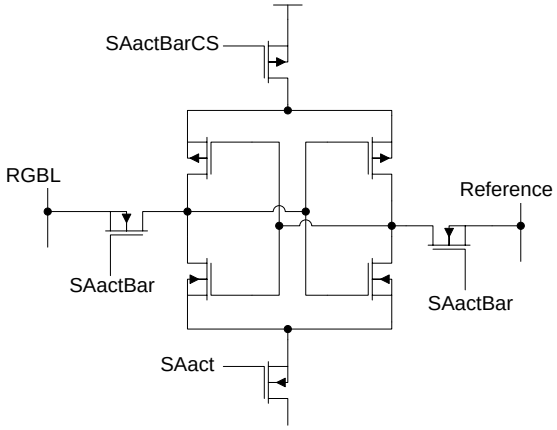

Figure 4.2.4: Circuit of the sense amplifier

As stated earlier the writing functionality is much simpler. Hence, the WLBL and $\overline{\text{WLBL}}$ nodes are common for the entire group. Since only one cell of the group can be written at any given time, the value to be written is imposed on the WLBL and $\overline{\text{WLBL}}$ and a pulse is applied to the gates of the respective pass transistors.

The partition is composed of two identical groups of the above organization. As far as the writing functionality is concerned, we distinguish two different sets of writing sources, i.e. $\{\overline{\text{WLBL0}}, \text{WLBL0}\}$ and $\{\overline{\text{WLBL1}}, \text{WLBL1}\}$. As for the reading functionality, we need to connect the reading local bit lines to some sort of common node. Hence, the end of each of the RLBL0 and RLBL1 is connected to a reading global bit line RGBL through group-specific read buffers.

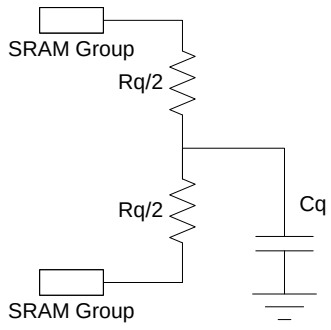

Figure 4.2.5: Equivalent of the RGBL

The RGBL is a large transmission line which is pre-charged to a specific voltage. It is hence populated with parasitic resistances and capacitances in the same way that the RLBLs are (obviously the values of the parasitic resistance quantum and of the parasitic capacitance are bound to be different).

The read buffer is a sub-circuit that is able to drive the higher load that is posed by the RGBL. The buffer is capable of sensing the downward swing of the respective RLBL, thus triggering a

transistor with increased dimensions that will drive the RGBL. In case no downward swing occurs, the read buffer does not operate and the RGBL is left to its pre-charged state. The operation of each read buffer is controlled by an activation signal.

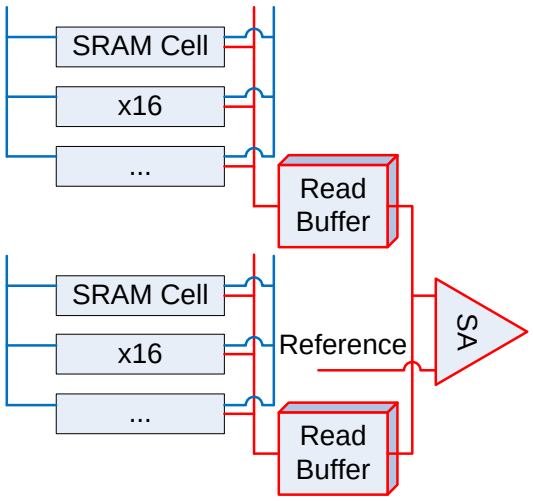

Figure 4.2.6: Organization of the SRAM partition

As is the case for almost every memory circuit, once a value is read, we need to ensure that the output of the memory is kept steady for a significant duration. This is achieved through the use of a sense amplifier. Given the single bit line structure of our memory circuit, the end of the RGBL will be connected to the input of a sense amplifier. The other input is connected to a reference voltage. That way, if the voltage at the end of the RGBL is found less than the reference, the sense amplifier latches to logic 0. In the opposite case, the sense amplifier latches to logic 1. The operation of the sense amplifier is controlled by two control signals.

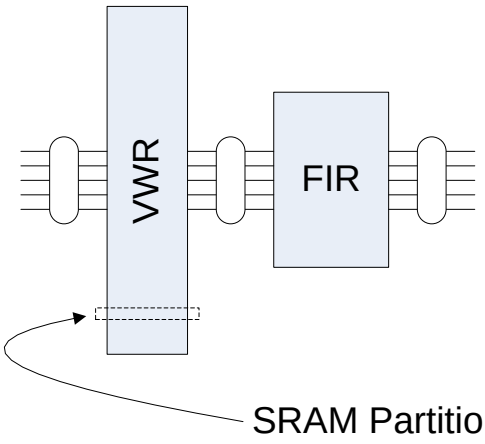

Figure 4.2.7: The ecosystem of the SRAM partition

The above circuits are used to create the entire SRAM partition (Figure 4.2.6; red colour indicates reading path and blue colour indicates writing path). It is noteworthy that the control signals of the cells (e.g. pulses for the read or write word lines), the enable signals for the buffers and the sense amplifier have to be externally defined. The usual decoder that controls the memory addressing is not implemented. That would seriously increase the device inventory, thus the computational complexity of any further simulation.

4.3 Target Architecture & Target Application

The current section will put the circuit presented above in the context of the target architecture and the target application. The target architecture will dictate the circuitry that is supposed to surround the circuit under test. It will also indicate the way that the input of the partition will be organized. The target application will indicate the control signals that should be applied to the partition. That way, the application will be realistically modeled for the circuit under test.

The architecture in question is a Very Wide Register (VWR), with a word length of 480 bits. The VWR is organized in two groups, of 480×16 bits. Since the simulations that were performed

are on the device level, it is plausible to select only one partition of the VWR as a representative unit of the entire architecture. This partition is the circuit under test.

The way that we utilize the aforementioned VWR, will dictate the exact activity that should be simulated on the SRAM partition. In the context of this test case, we assume that the above VWR is connected to the input of a digital FIR filter. The transmitted data are raw video (non-encoded). A realistic assumption of the SRAM activity is that the VWR is read wide-word-by-wide-word and rewritten with new values after all wide-words have been read. This high level description will be of vital importance for the creation of the partition's control signals. Given the organization of our partition (which is a column of the VWR), we can assume that after all cells have been written with data, we read the cells one by one. It would be realistic to leave a global retention interval between the consecutive read operations which may correspond to peripheral activity (e.g. decoding of VWR address etc.). Given this high level description, we can summarize the read/write activity of the SRAM partition in a few lines of pseudo code.

```
<Loop 1> {                          /* <Loop 1> iterations are  */
                                    /* user defined             */
  for (i=0; i<=31; i++) {           /* Write all cells first    */
    write(cell[i]);
    i++;
  }
  for  (i=0; i<=31; i++) {          /* Read cell-by-cell        */
    read(cell[i]);
    for (j=1; j<=10; j++) {         /* Global Retaining         */
      nop                           /* between reads            */
    }
    i++;
  }
}
```

Algorithm 4.3.1: Pseudo-code description of the target application

4.4 Stimuli & Input Setup

First, we need to account for realistic SRAM activity. Based on the target application as described in the previous section, we need to access the wide-word stream that is fed to the input of the FIR filter. These wide words have to be pre-stored in the VWR, then read and later transmitted to the input of the filter. Since we only simulate a single column of the VWR, we need to parse the filter input, and keep the bits that are stored (and later read) only in one column.

It is important that the addressing scheme is kept the same during the hypothetical operation of our digital system. The only thing that may change is the information that is stored and later read from the SRAM cells. A single stream of input for a given VWR column will be referred to as a *RunTime Situation* (RTS). The FIR filter input can thus provide us with 480 different RTSs. A

simple script in Perl is implemented to complete this parsing. Based on the available input, the duration of each RTS is roughly 12μs. Before defining exactly the control and input signals, we have to be reminded of the possible operation modes of an SRAM cell, i.e. Read, Write and Retain. The first two are quite self explanatory, whereas the third corresponds to complete cell inactivity, when a cell retains its value. We can safely assume that when a cell of the partition is written or read, all the others are in Retain operation mode. That way, we can break the SRAM activity into segments, each one corresponding to one of the above operation modes. Based on the above architecture and application target, we need to devise a way to setup both the control signals and the input that is to be written in the partition cells. The three different operation modes can be translated to three key commands to the memory.

`Read X Y`	Read the value stored in cell X found in group Y
`WriteW X Y`	Write value W in cell X found in group Y
`Retain`	Keep all cells at retention mode

Table 4.4.1: The three commands that are used for the control of the entire SRAM partition

Possible values of the **`x`** parameter can be from 0 to 31 (32 cells are included in the partition), whereas the **`y`** parameter can be either 0 or 1 (cells are organized in two groups). Obviously the parameter **`w`** can be either 0 or 1, since it corresponds to a single bit. In view of the above assumption, we can describe the activity of the entire SRAM partition based on these three commands. A script has been developed in Perl, for the parsing of an input file of SRAM activity. Certainly, the input file must not violate the internal consistency of the memory. For example, a cell cannot be read without previously being written. Based on the above processes, a total of 79 piece-wise linear voltage source files are created per RTS. This input is enough to simulate the activity of the entire SRAM partition for a user-defined duration.

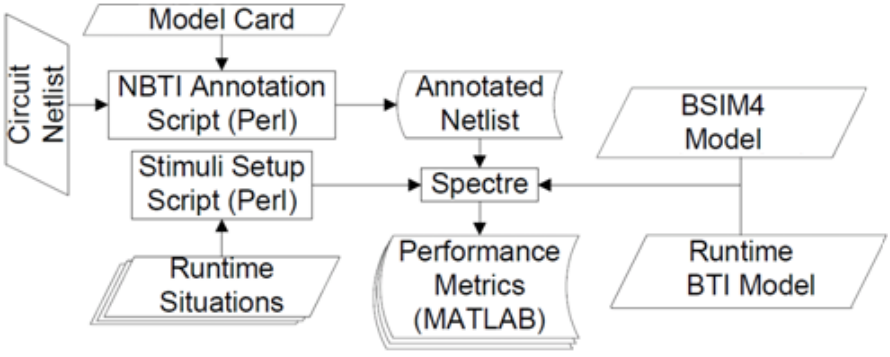

Figure 4.4.1: Complete framework state for test case simulations [24]

4.5 Simulation & Metric Extraction

Having covered the netlist annotation in Chapter 3, as well as the stimuli and input setup methodology in the previous section, we now have a complete view of the simulation framework

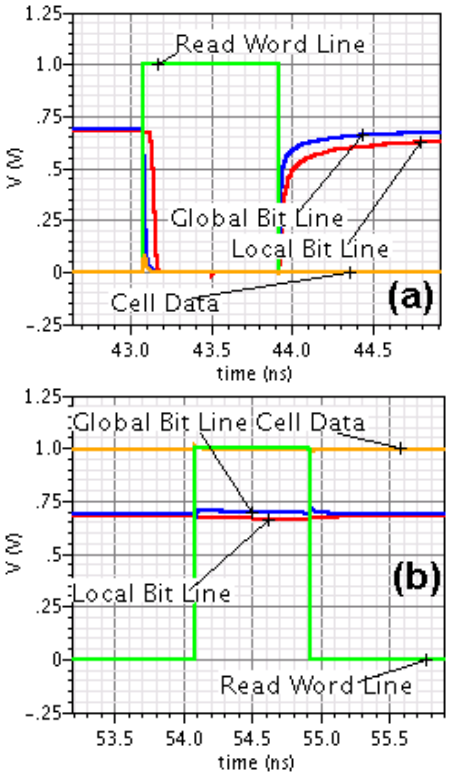

Figure 4.5.1: Instances of the reading operation [24]

effect on the cells and read buffers. In the current SRAM organization, this metric can be applied only to cases when the read value is logic 0. That is because in case of reading logic 1, there is no voltage swing in the end of the global bit line.

For the delay measurements, we use the built in capabilities of an industry standard simulator, which measures the duration from a trigger event up to a target event. This capability of the simulator cannot credibly detect a voltage difference between the inputs of the sense amplifier, when no swing occurs at the end of the global bit line. As a result, the delay measurements are restricted only to cases when logic 0 is read. This restriction does not affect the validity of any claim made in the current text but simply mirrors the particularity of the test case circuit.

for our test case. The objective of this section will be to present the logic behind the extraction of performance metrics, which enables parametric reliability analysis of the test case circuit. In this text, we will focus on read delay and leakage energy.

As far as delay is concerned, we measure the time from the activation of a read word line up to the point where the voltage difference at the inputs of the SA, is enough for the latter to sense. Based on [4], such a sufficient voltage difference can be assumed as 50mV. It is important that we do not incorporate the sense amplifier in the delay measurement, since it may be a tunable component. It also exhibits no impact to the preceding circuit. As a result the read delay measurement we are implementing mirrors the BTI

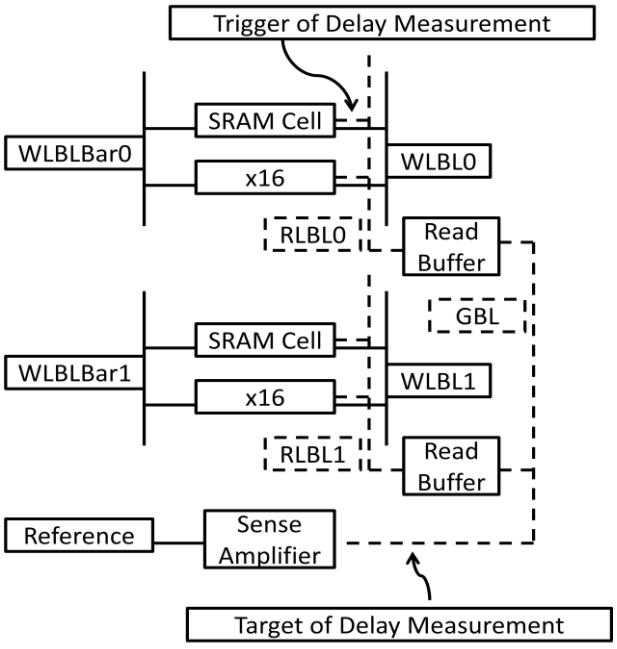

Figure 4.5.2: Read delay measurement across the read critical path

Based on the target application, there exist some global retention periods during the activity of the SRAM partition. This is a very good chance during which to measure *leakage energy*. During global retention instances, we give the circuit's currents a chance to settle. As a result, any dynamic effect is decaying, so leakage dominates the transient power measurement. Leakage energy is extracted using transient power measurements. We parse the power measurements during global retention instances and we numerically integrate them to get the leakage energy.

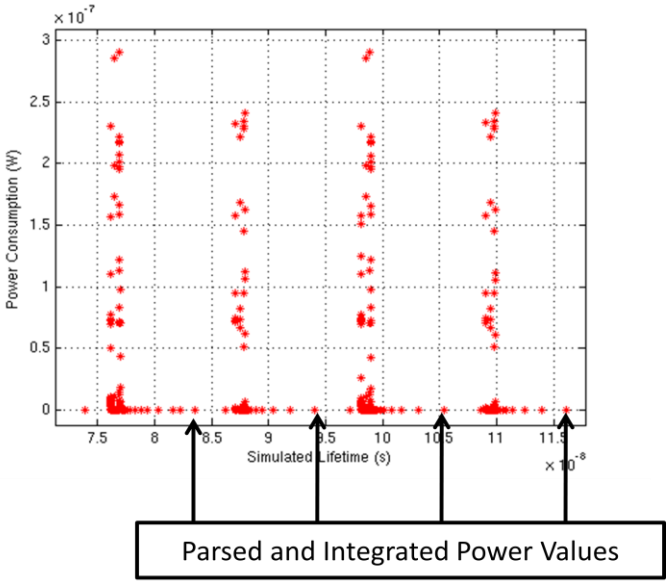

Figure 4.5.3: Power measurements leading to leakage energy estimation from the simulation output

Another interesting transient metric is the *BTI impact* on a specific device. For the definition of the BTI impact on the critical MOS, we need the $\Delta V_{th}[i]$ values, for all the device's defects. We define as O, the set of defects that are occupied during a specific time instance. We also define as T, the set of all defects belonging to this device. Hence, a valid definition of BTI impact is now obvious. It is equally obvious that this metric can take only discrete values, depending on the BTI parameters that have been assigned to that device during pre-processing.

$$\text{BTI Impact} = \frac{\sum_{i \in O} \Delta V_{th}[i]}{\sum_{i \in T} \Delta V_{th}[i]} \times 100\%$$

4.6 Parametric Fluctuations

For the purposes of this analysis, we test the SRAM partition netlist under 2 different RTSs of 12μs duration. In the first case, we test the circuit assuming no initial conditions for the devices' defects, i.e. all defects are non-occupied at the beginning of the simulation. In the second case, we initialize the occupancy of the traps using an AC BTI model [10]. Only NBTI is included in this case, but PBTI can easily be included as well. For the time being, we will ignored the leakage energy and focus only on the read delay of the partition.

The time zero simulation accounts exclusively for RTN, since the slower traps have no time to become occupied. However, the netlist that assumes initial conditions for the defects (even through the simplification of AC stressing) incorporates both RTN and NBTI. The initial AC stressing is assumed to have lasted 10^7s and the respective duty cycle (for the $-V_{gs}$) be 80%. Such an initial stressing is enough for some of the slow defects to become occupied. We also stress a reference netlist with the same pair of RTSs, thus having a control of our simulation experiment. The NBTI parameters are the same as the experimental values that calibrate the atomistic model.

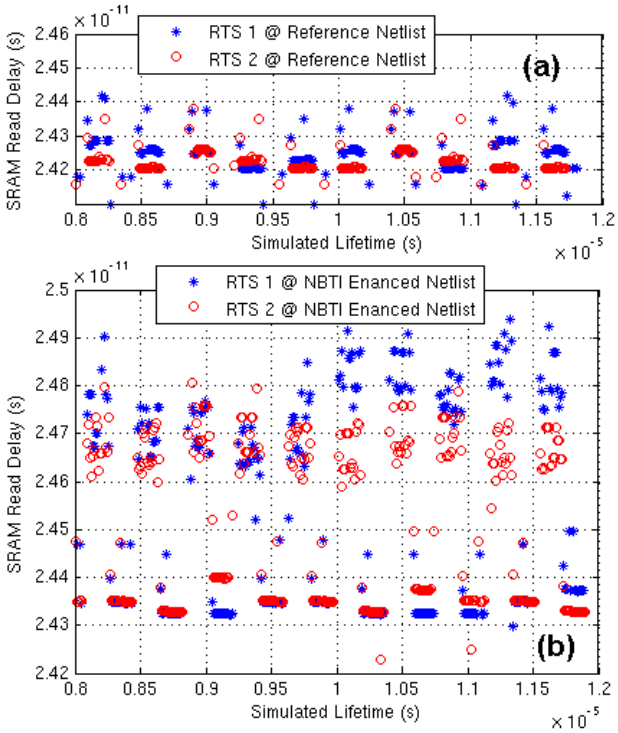

Figure 4.6.1: Runtime delay fluctuations of an SRAM partition for two different RTSs. No initial stressing is assumed (time zero simulation). The delay measurements are almost identical in the Reference netlist (a). For a netlist enhanced with defect activity (b), the fluctuations exhibit greater runtime variability and also differ between the two workloads [24].

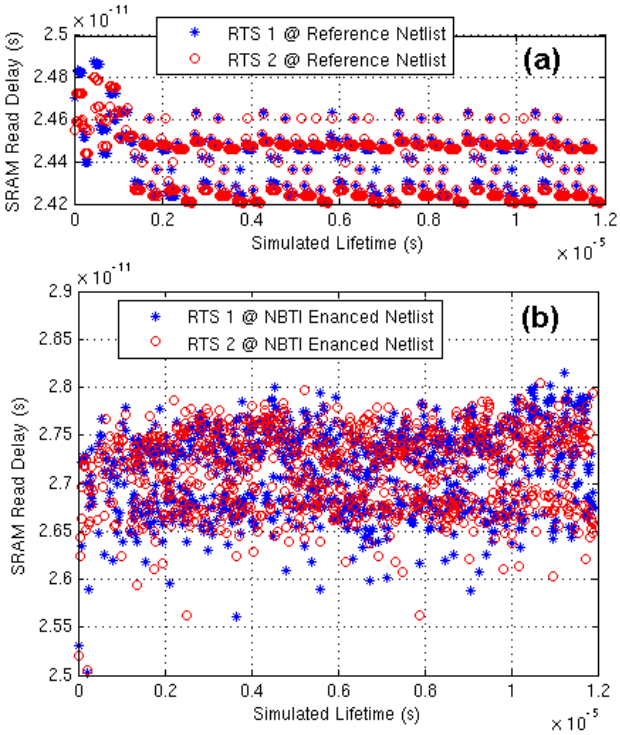

Figure 4.6.2: Runtime delay fluctuations of an SRAM partition for two different RTSs. Initial AC stressing is assumed. When NBTI is not considered (a), the fluctuations are concentrated in a very small interval. In a netlist enhanced with defect activity (b), the delay fluctuations cover a much wider interval [24].

What we observe in this case, is that the presence of NBTI widens, in any case, the interval of measured delay. When assuming only RTN, the fluctuations are concentrated in a wider interval than the reference case. With the incorporation of NBTI (initialized netlist), the interval widens even further and surpasses the usually accepted 10% margin. The transient evolution of the delay measurements is different for each RTS in the latter case. This morphological difference indicates that an attempt of clustering various RTSs is vital, if we are to study the impact that BTI has on the circuit's performance.

4.7 Conclusions

This chapter presented the simulation results on a test case circuit of a 32 bit SRAM partition. We have successfully accounted for realistic workloads, after considering the target application and the target architecture of the SRAM partition. We have customized the BTI and RTN simulation framework, in order to extract performance metrics of the circuit under test. Measurements of the circuit's read delay and leakage energy have been implemented. The customized framework encourages the extraction of distance metrics with respect to the delay behaviour. Such capabilities will likely enable the clustering of various RTSs, based on the delay impact because of BTI and RTN degradation. In view of the current framework's status, the clustering of RTSs to workload scenarios appears to be quite feasible.

Chapter 5: Accelerating HotSpot-5.0

5.1 Introduction

This chapter is based on the fifth version of the architecture-based compact thermal modelling tool called HotSpot, which is publicly available through [9]. An accurate, yet fast, thermal modelling tool is of vital importance on both design and research applications. With that in mind, we aim at the acceleration of the existing implementation. By maintaining the same level of accuracy, while the execution time is much less, HotSpot is tuned into a very powerful tool with a wider variety of applications. For instance, it can be utilized in ad hoc thermal modelling and thermal management applications.

We will attempt to distinguish, whether there is any further room for compromise between the current model's accuracy and its execution time. It is important for the optimization to take place in such a level that any processor-specific changes (use of acceleration engines etc.) do not affect the validity of our claims. Before exploring the employed acceleration techniques, we will present the principles of thermal simulation. Next, the profiling methodology will be analyzed, which will lead us to the most CPU intensive parts of this software. A brief analysis of the available numerical methods, will point towards the optimal acceleration technique. Finally, we will observe the accelerated version both through proof-of-concept simulations as well as in a realistic application.

5.2 Thermal Simulation Principles

HotSpot uses an RC model with which it represents the transient and steady state thermal behaviour of the various blocks found on any IC [8]. The creation of this model is based on the duality between electrical and thermal quantities [8], [20]. Given the thermal characteristics of an IC, this software is capable of creating a thermal equivalent, which can be solved for a user defined power consumption profile.

Thermal Network		Electrical Network	
Heat Flow-Power	P (W)	Current Flow	I (A)
Temperature Difference	T (K)	Voltage	V (V)
Resistance	R_{th} (K/W)	Resistance	R (Ω)
Rate of Inertia-Capacitance	C_{th} (J/K)	Capacitance	C (F)
Time Constant	$\tau_{th} = R_{th}C_{th}$ (s)	Time Constant	$\tau = RC$ (s)

Table 5.2.1: Duality between electrical and thermal quantities

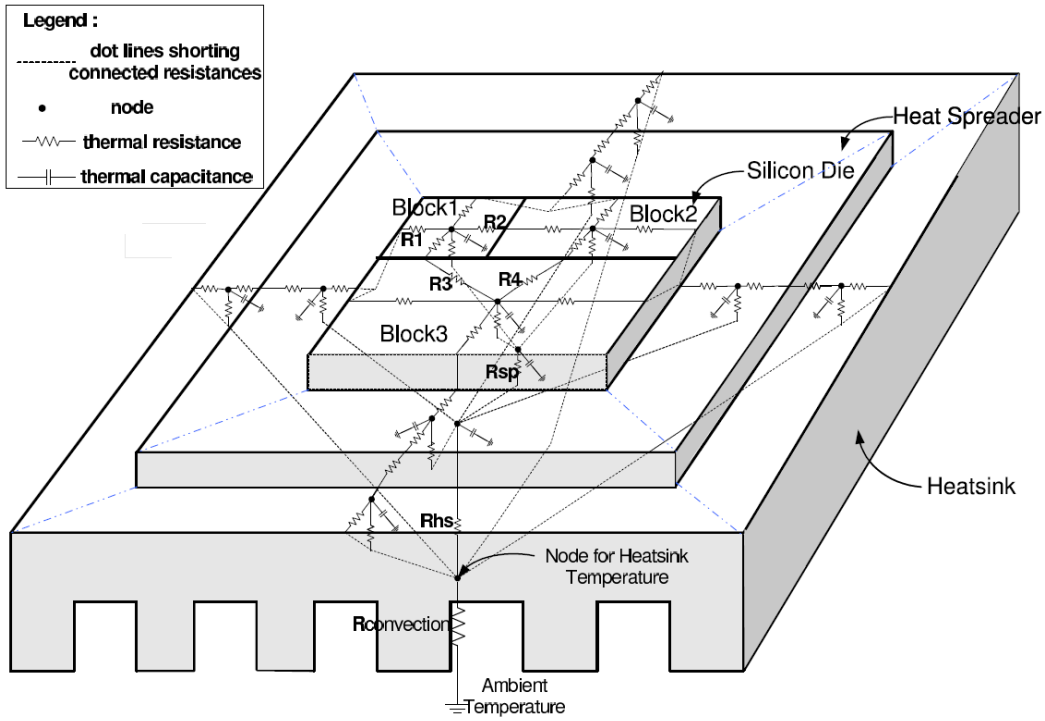

Figure 5.2.1: Thermal simulation principles: Each block of an IC is connected to others through thermal resistances and capacitances [26]

Taking into consideration the physical dimensions and thermal constants of the chip's functional blocks, the above duality yields an ordinary differential equation which describes the way in which thermal energy is spread throughout the chip.

$$\frac{d\bar{T}}{dt} + \boldsymbol{C}\bar{T} = \boldsymbol{A}^{-1}\bar{P}$$

where $\boldsymbol{A}$ and $\boldsymbol{C}$ represent the chip's thermal characteristics [9]. It is important that the current chapter deals exclusively with the transient solution of the above problem. HotSpot offers a steady state solution option, which reduces the ordinary differential equation to a simple algebraic relation, since it is obvious that $\frac{d\bar{T}}{dt} = 0$ at steady state.

In order to find the transient solution, HotSpot-5.0 employs a numerical iterative method, with adaptive step sizing. Being this an iterative process, the program needs to read a power consumption vector every t_{int} and given the previous temperature vector it seeks to calculate the new one. Given a time step t_s, HotSpot performs a set of calculations until a time interval equal to t_{int} is covered. The step t_s is susceptible to change, hence the adaptive step sizing. That way, HotSpot can inspect in greater detail parts of the solution that change rapidly, while taking vast strides in smoother parts of the solution. For each iteration (with internal step $t_s = h$) the previous temperature and power vectors ($\bar{T}_n$ and $\bar{P}_n$ respectively) are needed. That way, the necessary parameters of the numerical method can be calculated.

5.3 Execution Profiling

In this section we will present the methodology followed in order to identify CPU intensive instances during the program's execution. After a series of trial and error iterations, we placed timing functions in key parts of the source code. That way the duration of critical execution parts could be measured. The code was split into the following parts.

- Parsing and Initial Configuration: During this part we have the processing of the commands entered by the user. This information is combined with the specifications found in the configuration file.

- R Model Population: An array with the vertical and lateral thermal resistances between the functional blocks is created during this part. The population of the R model is sub-categorized as follows.
  - Model Sanity Check
  - Gx's and Gy's Calculation
  - Shared Lengths Calculation
  - Packaging Incorporation
  - Chip Edges
  - G Initialization
  - Overall Rs Calculation
  - Peripheral Nodes Incorporation
  - B Array Incorporation
  - Array Copying
  - Array LUP Decomposition

- C Model, Names, Temperature and Power Arrays Initialization: During this part, we have extensive memory allocation and preparation of power and temperature arrays. The functional blocks' names are also printed during this stage. Finally, the network of thermal capacitors is created.

- Numerical Part: The iterative numerical process for the solution of the ordinary differential equation is carried out during this stage.

For the profiling procedure, nine different floor-plans with their respective power traces were used. The total execution time for each benchmark is the sum of the execution times of each one of the above steps (plus any overhead of trifling procedures, like freeing any memory blocks etc). For the total execution time, we used built-in timing commands of the terminal. Finally, we can safely assume that the overhead introduced by the timing functions inside the source code is very small to be taken into consideration.

It is important to note that no accelerating engine is utilized during the profiling procedure. That is because our intention is to make the algorithm faster, while maintaining the same level of accuracy and not make processor-specific optimizations. We identify the LUP decomposition as well as the numerical part as the two dominant tasks of the transient solution. The focus of the current text is to optimize the second of the two stages, since it is the most CPU intensive. Optimization of the LUP decomposition is beyond the scope of this thesis.

Benchmark Name	tseng	apex2	alu4	apex4	ex5p
Number of Blocks	1217	1901	1530	1281	1108
Profiling Stages					
Parsing & Initial Configuration	0.03507	0.09076	0.06411	0.04437	0.04162
R Model Population	207.90791	1225.88951	510.54963	339.47135	212.61873
➢ Sanity Check	0.00007	0.00017	0.00023	0.00010	0.00011
➢ Gx's and Gy's Calculation	0.00043	0.00079	0.00103	0.00050	0.00043
➢ Shared Lengths Calculation	0.13489	0.41395	0.27474	0.19144	0.13599
➢ Packaging Incorporation	0.00001	0.00001	0.00001	0.00001	0.00001
➢ Chip Edges Incorporation	0.00025	0.00047	0.00036	0.00032	0.00025
➢ G Initialization	0.18894	0.61105	0.38454	0.28838	0.20784
➢ Overall Rs Calculation	0.22253	0.67702	0.43228	0.30996	0.22390
➢ Peripheral Nodes Incorporation	0.00000	0.00000	0.00000	0.00000	0.00000
➢ B Array Incorporation	0.44147	1.68490	0.90139	0.69288	0.47198
➢ Array Copying	0.44251	19.71515	0.41254	0.32405	0.27276
➢ **Array LUP Decomposition**	**206.47619**	**1202.56026**	**508.12696**	**337.66305**	**211.30481**
C Model, Names, Temperature and Power Arrays Initialization	0.48230	29.16401	4.34094	0.50904	0.38705
Numerical Part	**297.40017**	**906.18673**	**580.50909**	**400.49556**	**310.50477**
Total	**506.26747**	**2220.04100**	**1096.27189**	**741.07585**	**524.02856**

Benchmark Name	diffeq	misex3	seq	s298
Number of Blocks	1577	1403	1808	1940
Profiling Stages				
Parsing & Initial Configuration	0.06734	0.06241	0.07850	0.11604
R Model Population	534.58509	454.24514	832.53052	1247.79978
➢ Sanity Check	0.00012	0.00010	0.00015	0.00019
➢ Gx's and Gy's Calculation	0.00059	0.00056	0.00068	0.00079
➢ Shared Lengths Calculation	0.26161	0.23596	0.34541	0.42027
➢ Packaging Incorporation	0.00001	0.00001	0.00001	0.00001
➢ Chip Edges Incorporation	0.00039	0.00034	0.00046	0.00047
➢ G Initialization	0.39490	0.36115	0.52949	0.60874
➢ Overall Rs Calculation	0.43352	0.38285	0.56745	0.69750
➢ Peripheral Nodes Incorporation	0.00000	0.00000	0.00000	0.00000
➢ B Array Incorporation	0.91402	0.85107	1.35366	2.61172
➢ Array Copying	0.42286	0.38312	0.55911	19.79258
➢ **Array LUP Decomposition**	**532.15639**	**452.02943**	**829.16008**	**1223.60415**
C Model, Names, Temperature and Power Arrays Initialization	2.87897	0.66723	42.80352	29.74521
Numerical Part	**561.74322**	**525.10873**	**731.00479**	**944.14868**
Total	**1100.10970**	**980.78258**	**1611.73478**	**2270.33875**

Table 5.3.1: Profiling results for nine benchmarks on the default HotSpot-5.0 implementation; the most CPU intensive execution stages have been confirmed and highlighted; stage durations are presented in seconds

5.4 Numerical Approach

In order to solve the transient problem described by the aforementioned ordinary differential equation, HotSpot employs the 4th order Runge-Kutta method. Given the vectors $\bar{T}_n$ and $\bar{P}_n$, we seek to calculate $\bar{T}_{n+1}$, where $\bar{k}_i$ are the slopes of the temperature graphs for each node.

$$\left.\begin{array}{c} \bar{k}_1 = f(\bar{P}_n, \bar{T}_n) = \boldsymbol{A}^{-1}\bar{P}_n - \boldsymbol{C}\overline{T_n} \Rightarrow\ \bar{t}_1 = \bar{T}_n + \dfrac{h}{2} \times \bar{k}_1 \\ \bar{k}_2 = f(\bar{P}_n, \bar{t}_1) = \boldsymbol{A}^{-1}\bar{P}_n - \boldsymbol{C}\overline{t_1} \Rightarrow\ \bar{t}_2 = \bar{T}_n + \dfrac{h}{2} \times \bar{k}_2 \\ \bar{k}_3 = f(\bar{P}_n, \bar{t}_2) = \boldsymbol{A}^{-1}\bar{P}_n - \boldsymbol{C}\overline{t_2} \Rightarrow\ \bar{t}_3 = \bar{T}_n + h \times \bar{k}_3 \\ \bar{k}_4 = f(\bar{P}_n, \bar{t}_3) = \boldsymbol{A}^{-1}\bar{P}_n - \boldsymbol{C}\overline{t_3} \end{array}\right\} \Rightarrow$$

$$\bar{T}_{n+1} = \bar{T}_n + \frac{h}{6} \times (\bar{k}_1 + 2 \times \bar{k}_2 + 2 \times \bar{k}_3 + \bar{k}_4)$$

The above set of calculations is performed three times for the sake of adaptive step sizing every new $t_s = h$. Thus, it appears to be the core of the numerical solution performed by HotSpot. If we combine this observation with the results from the profiling phase, this set of equations is rightly considered one of the first areas to be examined for any execution time vs. accuracy trade-offs. We find that the error of the method employed by the original HotSpot implementation is $O(h^5)$ [22]. Given the very small value of the step, even though it is dynamically changing (adaptive step sizing), it appears logical to increase this error in favour of a faster execution. Thus, we conclude to the following simpler set of equations for a single iteration.

$$\bar{k}_1 = f(\bar{P}_n, \bar{T}_n) \Rightarrow \bar{T}_{n+1} = \bar{T}_n + h\bar{k}_1$$

The error of the above implementation (namely the Euler method, instead of the fourth order Runge-Kutta) is $O(h^2)$ [22]. Given the small values of the time step, we can safely assume that the accuracy of the method remains at a similar level in comparison to the original implementation. Before attempting a complexity analysis using segments from the source code, we will attempt a presentation of the proposed numerical approach idea in a more simplistic way. As we know from the mathematical background the vectors $\bar{k}_i$, represent the slopes of the temperature graphs for each node. As a result, we can easily calculate them having formed the problem's differential equation.

$$\frac{d\bar{T}}{dt} = \bar{k}_i = \boldsymbol{A}^{-1}\bar{P}_n - \boldsymbol{C}\overline{T_n}$$

Let us assume that the slope calculation is performed by a function which receives the power and temperature vectors as input, thus we define `evaluate_slope(Pn,Tn)`. We also assume that the vector $\bar{k}_1$ is available at the beginning as is the time step for each iteration (namely, $t_s = h$). We define `rk4(Pn,Tn,k1,h)`, as the function that will return the vector $\bar{T}_{n+1}$. From the above analysis, the pseudo code for the default `rk4_default(Pn,Tn,k1,h)` can be easily extracted. The

```
rk4_default(Pn,Tn,k1,h) {

  t1 = Tn + h/2.0 * k1;

  k2 = evaluate_slope(Pn,t1);
  t2 = Tn + h/2.0 * k2;

  k3 = evaluate_slope(Pn,t2);
  t3 = Tn + h * k3;

  k4 = evaluate_slope(Pn,t3);

  Tnew = Tn + h * (k1 + 2*k2 + 2*k3 + k4)/6.0;
  return Tnew;
}
```

```
proposed_approach(Pn,Tn,k1,h) {

  Tnew = Tn + (h * k1);

  return Tnew;
}
```

Algorithm 5.4.1: Default and proposed numerical approach expressed in pseudo code

proposed approach is equally easy to express in the form of pseudo code. Given that each $\bar{T}_{n+1}$ has to be calculated several times and for several cases in order to achieve adaptive step sizing, it is important that this critical code of the iterative calculations be fast and yet accurate. We must note that all the above variables represent vectors apart from $t_s = h$, which is the current iteration step, thus being a single positive number. The computational effort required in the proposed approach is seriously decreased since the iteration requires four times less the original number of operations. Given the level at which the alterations are performed, we can safely assume that any additional acceleration methods (e.g. processor-specific acceleration engines) will only further improve the execution performance.

5.5 Complexity Analysis & Source Code Alterations

In this section, we will look closer to the actual source code of HotSpot. We will identify the key function that has to be altered and will introduce the proposed alteration. Finally, a simple complexity analysis was deemed necessary to further support our acceleration claims. Since the source code of HotSpot-5.0 is rather lengthy, we will concentrate on the function **`rk4_core`** which was identified as the specific function implementing the 4th order Runge-Kutta method.

This being a purely algorithmic complexity analysis, we will deprive any presented function from processor-specific acceleration methods. Furthermore, we will not look into memory access and memory allocation times but only into purely mathematical operations. A basic function we must consider is the **`matvectmult`**, which multiplies a square matrix with a column vector. We add simple comments to show the complexity of any noticeable steps. Evidently, we must look for the dominating complexity throughout these steps. The complexity of function commands, will be presented in comments syntax.

```
void matvectmult(double *vout, double **m, double *vin, int n) {
  int i, j;
  for (i = 0; i < n; i++) {           /* O(n) */
    vout[i] = 0;                      /* O(n) */
    for (j = 0; j < n; j++)           /* O(n^2) */
      vout[i] += m[i][j] * vin[j];    /* O(n^2) */
    }
  }
}
```

Algorithm 5.5.1: The function **matvectmult** found in HotSpot-5.0

```
void slope_fn_block(block_model_t *model, double *y, double *p, double *dy) {
   int n = model->n_nodes;
  double **c = model->c;
  int i;
  double *t = dvector(n);

  matvectmult(t, c, y, n);                  /* O(n^2) */

  for (i = 0; i < n; i++)                   /* O(n) */
    dy[i] = p[i]-t[i];                      /* O(n) */

  free_dvector(t);
}
```

Algorithm 5.5.2: The function **slope_fn_block** found in HotSpot-5.0

```
void rk4_core(void *model, double *y, double *k1, void *p, int n, double h, double
*yout, slope_fn_ptr f) {
  int i;
  double *t, *k2, *k3, *k4;
  k2 = dvector(n);
  k3 = dvector(n);
  k4 = dvector(n);
  t = dvector(n);

  for(i=0; i < n; i++)
    t[i] = y[i] + h/2.0 * k1[i]; /* O(n) */
  (*f)(model, t, p, k2);         /* O(n^2) *
  for(i=0; i < n; i++)
    t[i] = y[i] + h/2.0 * k2[i]; /* O(n) */
  (*f)(model, t, p, k3);         /* O(n^2) */
  for(i=0; i < n; i++)
    t[i] = y[i] + h * k3[i];     /* O(n) */
  (*f)(model, t, p, k4);         /* O(n^2) */

  for (i =0; i < n; i++)
    yout[i] = y[i] + h * (k1[i] + 2*k2[i] + 2*k3[i] + k4[i])/6.0; /* O(n) */

  free_dvector(k2);
  free_dvector(k3);
  free_dvector(k4);
  free_dvector(t);

}
```

Algorithm 5.5.3: The function **rk4_core** found in HotSpot-5.0

It is evident that the function **matvectmul** follows an $O(n^2)$ complexity, where n is the common dimension between the multiplied matrix and vector. In this case, the n parameter is the number of nodes found in the thermal network. The function **matvectmul** is essential for the calculation of the slopes of the temperature function for each thermal node. The latter task is performed by the function **slope_fn_block** [9]. The function **slope_fn_block** (which is called by reference in **rk4_core** as **f**) has the same complexity, namely $O(n^2)$, where n has the same meaning as above. Finally, we calculate the complexity of the **rk4_core** function in a similar way. As a result, the complexity of the function **rk4_core** in the original implementation is also $O(n^2)$, where n the number of nodes in the thermal equivalent circuit. Our approach reduces drastically the required computational effort, by simplifying the function **rk4_core**. The complexity of the replacement function **euler_core** was reduced to $O(n)$, where n the number of nodes in the thermal equivalent circuit. Obviously, our intended approach is the implementation of Euler's numerical method rather than the fourth order Runge-Kutta method which was originally utilized. A reduced complexity does not qualify as an acceptable solution to our acceleration problem. It is vital to consider the underlying numerical error as well as the overall performance of the altered source code, in a series of benchmarks.

```
void euler_core(void *model, double *y, double *k1, void *p, int n,
double h, double *yout, slope_fn_ptr f) {

  int i;

  for (i =0; i < n; i++) yout[i] = y[i] + (h * k1[i]); /* O(n) */

}
```

Algorithm 5.5.4: The function **euler_core**, which is proposed to replace **rk4_core**

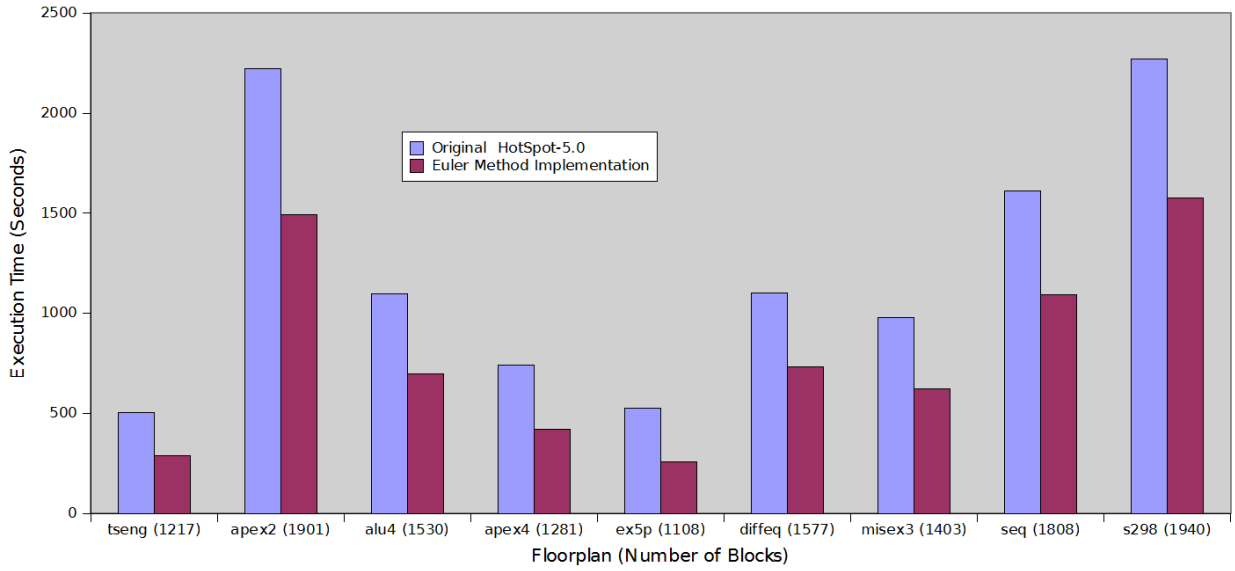

Figure 5.5.1: Execution time for nine benchmarks comparing the default and the numerically optimized versions

Benchmark Name	**tseng**	**apex2**	**alu4**	**apex4**	**ex5p**
Number of Blocks	1217	1901	1530	1281	1108
Profiling Stages					
Parsing & Initial Configuration	0.05237	0.09115	0.06365	0.07130	0.03663
R Model Population	227.06458	1206.25284	566.81006	334.95655	196.46724
➢ Sanity Check	0.00008	0.00017	0.00010	0.00009	0.00009
➢ Gx's and Gy's Calculation	0.00050	0.00075	0.00059	0.00050	0.00046
➢ Shared Lengths Calculation	0.13417	0.41335	0.27008	0.18832	0.14047
➢ Packaging Incorporation	0.00001	0.00001	0.00001	0.00001	0.00001
➢ Chip Edges Incorporation	0.00026	0.00049	0.00035	0.00030	0.00024
➢ G Initialization	0.24589	0.61557	0.37207	0.29092	0.19995
➢ Overall Rs Calculation	0.22455	0.67741	0.43227	0.30835	0.22629
➢ Peripheral Nodes Incorporation	0.00000	0.00000	0.00000	0.00000	0.00000
➢ B Array Incorporation	0.47995	3.33825	0.90672	0.65763	0.44595
➢ Array Copying	0.25400	19.55677	0.51642	0.32518	0.21384
➢ **Array LUP Decomposition**	**225.72439**	**1181.36879**	**564.31078**	**333.18463**	**195.23936**
C Model, Names, Temperature and Power Arrays Initialization	0.54130	32.78961	5.48549	0.71735	0.34254
Numerical Part	60.11105	193.25667	124.40264	85.47268	61.28075
Total	288.38408	1491.72987	697.70546	422.07990	258.52480

Benchmark Name	**diffeq**	**misex3**	**seq**	**s298**
Number of Blocks	1577	1403	1808	1940
Profiling Stages				
Parsing & Initial Configuration	0.07901	0.06344	0.09690	0.08448
R Model Population	604.16245	505.41729	901.19450	1226.08093
➢ Sanity Check	0.00017	0.00018	0.00014	0.00029
➢ Gx's and Gy's Calculation	0.00059	0.00056	0.00070	0.00132
➢ Shared Lengths Calculation	0.30961	0.23621	0.34606	0.42022
➢ Packaging Incorporation	0.00001	0.00001	0.00001	0.00001
➢ Chip Edges Incorporation	0.00033	0.00033	0.00045	0.00049
➢ G Initialization	0.39131	0.33162	0.52028	0.61335
➢ Overall Rs Calculation	0.44129	0.38460	0.57457	0.67863
➢ Peripheral Nodes Incorporation	0.00000	0.00000	0.00000	0.00000
➢ B Array Incorporation	0.96120	0.82940	1.35027	1.65129
➢ Array Copying	1.84087	0.93420	2.06790	18.52872
➢ **Array LUP Decomposition**	**600.21634**	**502.66573**	**896.32433**	**1203.53025**
C Model, Names, Temperature and Power Arrays Initialization	4.43278	1.74777	21.19745	68.39291
Numerical Part	120.70472	113.25028	166.68681	192.04378
Total	731.38462	621.20606	1094.42386	1577.51120

Table 5.5.1: Profiling results for nine benchmarks of the accelerated version (Euler method); stage durations are presented in seconds

5.6 Optimization Results

In this section, we will present the execution time for nine different benchmarks (Table 5.5.1). A comparison between the default HotSpot-5.0 and our intended approach will be performed. It is noteworthy that the complexity of the benchmarks is quite high (large number of thermal blocks). However, this was intentionally chosen, in order to stress the proposed numerical method in extreme cases.

At the same time, we compared the temperature output between the default and numerically accelerated version. The result was a maximum difference equal to 0.03℃ [25]. It is important that we have managed to reduce the CPU intensity of this software, by keeping the trade-off with accuracy in acceptable margins.

The next step is to verify the stage of the source code that would require further optimization, apart from the use of the Euler method. Again, we place timing functions of reduced overhead in key parts of the numerically optimized source code and monitor the duration of various segments of the code. What we identify is that the role of the numerical part (i.e. the one that deals with the solution of the ordinary differential equation) has been greatly reduced. The profiling indicates that the LUP decomposition is a candidate for future acceleration attempts.

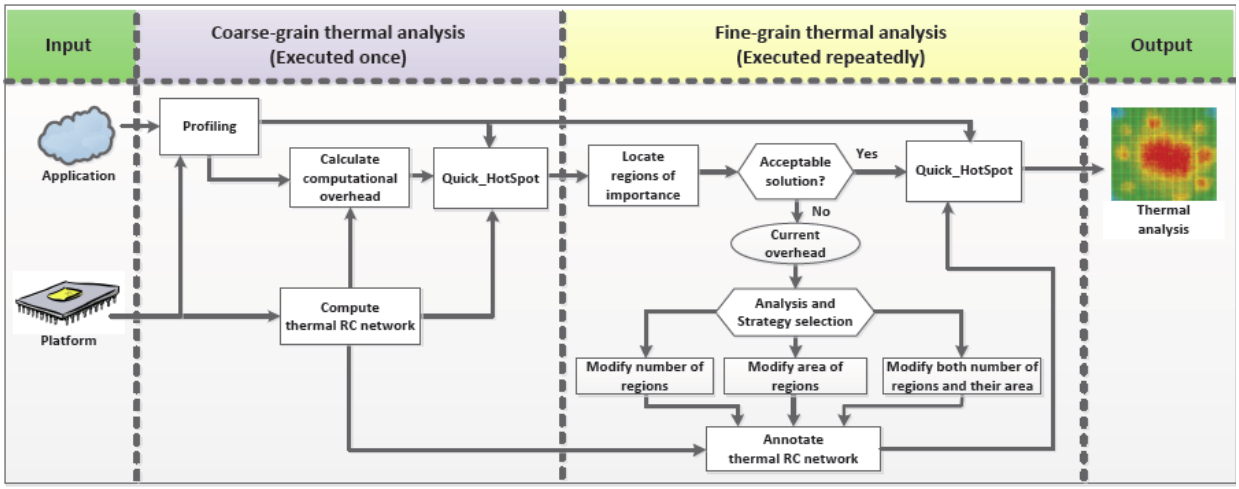

Figure 5.7.1: Incorporation of the accelerated thermal simulator (Quick_Hotspot) into a hierarchical thermal analysis tool

5.7 Applications of the Accelerated Version

In [25], we can see an application of the above software, incorporating the optimized numerical (Euler) method. The enhanced thermal simulator, which is named Quick_Hotspot, has been included in a hierarchical thermal analysis tool (Figure 5.7.1). The purpose of this tool is to analyze the temperature distribution across a user defined IC in different levels of granularity.

During an initial coarse-grain phase, we obtain a crude view of the thermal distribution across the chip. This phase is executed only once and it requires the power sources found in the integrated circuit. Such values are naturally dependent on the mapped application. Also, a template of the RC thermal equivalent network is created, upon which any further thermal analysis will be based. For the latter, we require the thermal and geometric constants of the chip's functional blocks. Finally, regions of importance can be created during this early step, by grouping hardware resources that operate over a certain temperature threshold.

On the granularity of the above network lies the tuneable accuracy of the proposed framework. Depending on whether the level of accuracy is acceptable, it is possible to annotate the aforementioned template, so that a fine-grain thermal analysis can be achieved. This task is performed during the second, iterative, phase of the tool.

In case the achieved detail is not satisfactory, we can by modify the number of critical regions, or modify the area of critical regions. Also a combination of these two techniques is possible. A feedback loop enables this sort of granularity calibration.

Having settled to a sufficiently detailed thermal profile, we find the transient solution to the temperature distribution problem and we present the thermal map at a given point of the chip's operation.

5.8 Conclusions

In this chapter, we have presented the optimization approach of a popular thermal simulator called HotSpot-5.0. Initially, a simple profiling of the software allowed us to concentrate on the most CPU intensive tasks. Having distinguished the solution of the ordinary differential equation as such a part, we explored various numerical techniques that could achieve a faster yet accurate solution. The Euler method was picked as a suitable replacement of the 4^{th} order Runge-Kutta, the latter being the default implementation. The execution of the altered source code revealed a significant acceleration, whereas the deviation from the original tool is negligible. Hence, it is obvious that we achieved a significant acceleration of the thermal simulator. That way, we have broadened the simulator's application spectrum.

An example of a hierarchical thermal analysis tool that uses our accelerated version demonstrates that the field of application may vary from runtime thermal managers to design time detailed thermal analysis tools. Certainly, further improvements of the source code, for instance optimization of the LUP decomposition part, may yield much more optimistic versions.

Chapter 6: Conclusion

6.1 Overview

In the previous chapters, we have presented the involvment of the author in two major areas. On the one hand, a fully functional framework has been created to simulate the activity of an SRAM partition under the effect of BTI and RTN. These two phenomena have been accounted for based on a novel atomistic approach that describes the occupancy of defects in a deterministic way. Performnace metrics of the SRAM partition have been monitored at runtime and memory workloads can be grouped together, based on the impact of BTI and RTN on the delay and leakage energy [41].

On the other hand, a thermal analysis tool (HotSpot-5.0) has been studied extensively for the purporse of reducing its execution time. Profilling functions have been used to identify the most CPU intensive code segments. Finally, a numerical method was replaced by an optimized equivalent, thus leading to reduced execution time with no noticeable accuracy degradation. Finally, it has been demonstrated in practice that the accelerated version of the above tool (referred to as Quick_Hotspot) can be used in frameworks that support hierarchical thermal analysis.

Having completed the two above tasks, it is possible to identify the potential for future work in both areas. This reflection will be the main task of the current chapter, alongside a brief evaluation of the achieved results. For the sake of consistency, we will split the following chapter according to the two main axes of this text.

6.2 Effects of BTI & RTN on the Performance of an SRAM Partition

In this area of focus, we have successfully proved that the atomistic quality of the BTI and RTN device level model can propagate to the circuit level as detailed workload dependency. Simulations of the SRAM partition have proved that under the effects of BTI and RTN, the read delay of the memory spreads in an alarmingly wide interval. There lies clear motivation for the development of mitigation techniques for BTI and RTN.

Steps towards enhanced parametric reliability, especially for embedded systems, can be supported by the workload dependency of the simulation framework. Workload dependency can be emphasized by suppressing the completely stochastic component of the model (via random number generation manipulation). It has been demonstrated that the workloads applied to the SRAM partition, can be grouped based on their impact on delay and/or leakage energy. Such groups of RTSs can be considered as workload scenarios that constitute different operation modes for a digital system (e.g. an embedded system). As a result, the creation of microcontrollers that can switch between such operation modes appears to be an appealing

concept. Certainly, the calibration of an underlying model is required. For this purpose, extensive simulations are required, in order to test many different functional blocks (i.e. memories, combinatorial circuits etc.), for many different BTI and RTN instantiations, under a wide variety of workloads. In order to enable such a workload scenario characterization, it is imperative to enhance the current framework with some new key qualities.

- Regarding the device level model, it is very important to calibrate it further we new measurements on real devices. It is important to clarify the way that various defect parameters of the device are scaled with the device dimensions (e.g. trap density).

- The *in-situ* defect monitoring script is implemented in quite a high level language (Verilog-A). This requires compilation for every different defect instantiation and is generally evaluated as highly suboptimal solution. A use of a lower level language (e.g. C) would allow platform specific optimizations and easier profiling. Hence, execution time obstacles could be easier alleviated.

- The annotation framework could be enhanced in the following way: We need to try different initial stressing for the same defect instantiation.

- Since we are unable to simulate more than ~10^{-5} seconds, we need a technique to inspect small parts of the partition's lifetime, while we can speed through the latter using the analytical model. Certainly, the use of the AC analytical model [10] does not fit the workload dependent qualities that differentiate the atomistic approach from the state of the art. For example, any attempt to fit the AC stress to the percentage of stored ones or zeros is very crude and certainly out of the question. Still, based on the current framework, the aforementioned way appears as the only computationally feasible idea.

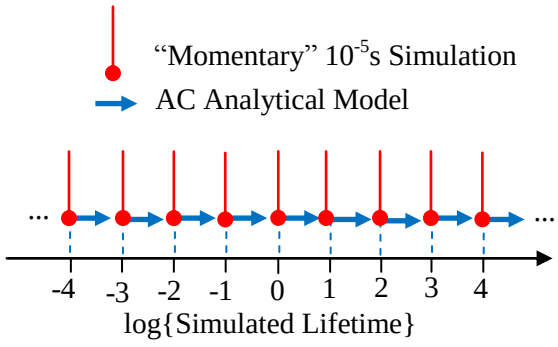

Figure 6.2.1: The analytical model can speed through the partition's lifetime, while the simulator can stop per decade and inspect transient delay fluctuations for different RTSs (e.g. 50 RTSs per momentary simulation should suffice).

- On parallel, methods to accelerate the simulation (which is the most CPU intensive task) would be hugely helpful. We need to maintain the differentiation from the averaging out state of the art, hence the AC model mentioned above should only be a short term solution.

- Fortunately, the netlist annotation framework is generic enough to be applied to any netlist. However, the tools that create various RTSs and inspect performance metrics are highly netlist specific. It would be interesting to increase the scope of this part of the framework.

That way, various types of circuits and workload can be easily studied without the need for a time consuming “context switch”.

6.3 Acceleration of the Transient Solution on HotSpot-5.0

In the results presented in the respective chapter, we have demonstrated a satisfactory acceleration of HotSpot-5.0, with negligible accuracy degradation. However, the profiling of the accelerated version showed that the LUP decomposition part of the software remains as the most CPU intensive segment. Hence, it is imperative to explore the numerical optimization of the respective code.

It is vital to remain in the same course of action and not attempt any processor-specific optimizations. Our goal is to make the algorithm faster, so any localized acceleration will only add up to the already achieved reduction of CPU time.

Finally, having increased the potential applications of this thermal analysis tool, the most appealing potential is to use it for runtime thermal management purposes.

References

[1] Alam M. A., “On the Reliability of Micro-electronic Devices: An Introductory Lecture on Negative Bias Temperature Instability,” in *Nanotechnology 501 Lecture Series*, September 2005. Available at http://www.nanohub.org/resources/?id=193.

[2] Calimera, A.; Macii, E.; Poncino, M.; , "Analysis of NBTI-induced SNM degradation in power-gated SRAM cells," *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on* , vol., no., pp.785-788, May 30 2010-June 2 2010

[3] Catthoor F., “Introduction to overall reliability research topics and cooperation network”, Reliability Mini-Workshop, 2011 IMEC, Belgium

[4] Cosemans S., “Variability-aware design of low power SRAM memories”, PhD thesis 2009, KULeuven

[5] Cosemans Stefan; Wim Dehaene; Francky Catthoor; , "A Low Power Embedded SRAM for Wireless Applications," *Solid-State Circuits Conference, 2006. ESSCIRC 2006. Proceedings of the 32nd European* , vol., no., pp.291-294, Sept. 2006

[6] Cosemans, S.; Dehaene, W.; Catthoor, F.; , "A 3.6 pJ/Access 480 MHz, 128 kb On-Chip SRAM With 850 MHz Boost Mode in 90 nm CMOS With Tunable Sense Amplifiers," *Solid-State Circuits, IEEE Journal of* , vol.44, no.7, pp.2065-2077, July 2009

[7] D. K. Schroder et al., "Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing", Journal of Applied Physics, vol.94, no.1, p.1 (2003).

[8] K. Skadron, et. al., “Temperature-aware microarchitecture: Extended discussion and results”, Technical Report CS-2003-08, University of Virginia, Computer Science Department, 2003.

[9] K. Skardon et al., HotSpot-5.0. [Online]. LAVA Lab: University of Virginia, 2010.

[10] Kaczer B. *et al.*, “Atomistic approach to variability of bias-temperature instability in circuit simulations”, *Reliability Physics Symposium (IRPS), 2011 IEEE International* (accepted)

[11] Kaczer, B.; Grasser, T.; Martin-Martinez, J.; Simoen, E.; Aoulaiche, M.; Roussel, P.J.; Groeseneken, G.; , "NBTI from the perspective of defect states with widely distributed time scales," *Reliability Physics Symposium, 2009 IEEE International* , vol., no., pp.55-60, 26-30 April 2009

[12] Kaczer, B.; Grasser, T.; Roussel, P.J.; Franco, J.; Degraeve, R.; Ragnarsson, L.; Simoen, E.; Groeseneken, G.; Reisinger, H.; , "Origin of NBTI variability in deeply scaled pFETs," *Reliability Physics Symposium (IRPS), 2010 IEEE International* , vol., no., pp.26-32, 2-6 May 2010

[13] Green Keith; Mu F.; Kapila G.; Reddy V. (2010, November 24). *Simulation of Circuit Reliability with RelXpert* [Online]. Available: http://www.cdnusers.org/Portals/0/cdnlive/na_download/Tuesday/Track3/1230/1230_Green.pdf

[14] Kolhatkar, J.S.; Vandamme, L.K.J.; Salm, C.; Wallinga, H.; , "Separation of random telegraph signals from 1/f noise in MOSFETs under constant and switched bias conditions," *European Solid-State Device Research, 2003. ESSDERC '03. 33rd Conference on* , vol., no., pp. 549-552, 16-18 Sept. 2003

[15] Kufluoglu, H.; Reddy, V.; Marshall, A.; Krick, J.; Ragheb, T.; Cirba, C.; Krishnan, A.; Chancellor, C.; , "An extensive and improved circuit simulation methodology for NBTI recovery," *Reliability Physics Symposium (IRPS), 2010 IEEE International* , vol., no., pp.670-675, 2-6 May 2010

[16] Kumar, S.V.; Kim, C.H.; Sapatnekar, S.S.; , "An Analytical Model for Negative Bias Temperature Instability," *Computer-Aided Design, 2006. ICCAD '06. IEEE/ACM International Conference on* , vol., no., pp.493-496, 5-9 Nov. 2006

[17] Kumar, S.V.; Kim, C.H.; Sapatnekar, S.S.; , "NBTI-Aware Synthesis of Digital Circuits," *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE* , vol., no., pp.370-375, 4-8 June 2007

[18] Kumar, S.V.; Kim, K.H.; Sapatnekar, S.S.; , "Impact of NBTI on SRAM read stability and design for reliability," *Quality Electronic Design, 2006. ISQED '06. 7th International Symposium on* , vol., no., pp.6 pp.-218, 27-29 March 2006

[19] Maricau, E.; Gielen, G.; , "Efficient Variability-Aware NBTI and Hot Carrier Circuit Reliability Analysis," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.29, no.12, pp.1884-1893, Dec. 2010

[20] Mircea S., "Advanced Digital Integrated Circuits ; Lecture 20: Thermal Design," *EECS Instructional and Electronics Support*, May 04, 2005. [Online]. Available: http://bwrc.eecs.berkeley.edu/classes/icdesign/ee241_s05/Lectures/Lecture20_Thermal.pdf. [Accessed: July 24, 2010].

[21] Paul, B.C.; Kunhyuk Kang; Kufluoglu, H.; Alam, M.A.; Roy, K.; , "Negative Bias Temperature Instability: Estimation and Design for Improved Reliability of Nanoscale Circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.26, no.4, pp.743-751, April 2007

[22] Press W., et.al., Numerical Recipes in C; The Art of Scientific Computing, 2nd ed. Cambridge: Cambridge Univ. Press, 1997.

[23] Quick Hotspot tool, available at http://proteas.microlab.ntua.gr/ksiop/software.html

[24] Rodopoulos D. et al., "Time and Workload Dependent Device Variability in Circuit Simulations", International Conference on Integrated Circuit Design and Technology, 2011 IEEE International (accepted)

[25] Siozios Kostas, Rodopoulos Dimitris and Soudris Dimitrios, «Quick_Hotspot: A Software Supported Methodology for Supporting Run-Time Thermal Analysis at MPSoC Designs,» accepted presentation in 2nd PARMA Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures, co-located with ARCS 2011 - Architecture of Computing Systems, 23 February 2011, Lake Como, Italy.

[26] Skadron, K.; Stan, M.R.; Huang, W.; Sivakumar Velusamy; Karthik Sankaranarayanan; Tarjan, D.; , "Temperature-aware microarchitecture," *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on* , vol., no., pp. 2- 13, 9-11 June 2003

[27] Toledano-Luque, M. et al., "From mean values to distributions of BTI lifetime of deeply scaled FETs through atomistic understanding of the degradation", *VLSI Technology Symposium, 2011* (accepted)

[28] Toledano-Luque, M. et al., *"*Response of a Single Trap to AC Negative Bias Temperature Stress", *Reliability Physics Symposium (IRPS), 2011 IEEE International* (accepted)

[29] Tsividis Y., "Operation and Modeling of the MOS Transistor", 2nd Edition, McGraw-Hill, New York, 1999.

[30] Valduga de Almeida Camargo Vinicius, "NBTI Simulator User Guide".

[31] Wang X. et al., "Reliability Simulation on the Virtuoso Analog Design Environment", Cadence Design Systems, available online at http://www.cadence.com/cdnlive/library/Documents/2009/NA/100509%20-%20Track1-3%20-%20Xiao%20Wang%20-%20Cadence_Final.pdf

[32] Wenping Wang; Zile Wei; Shengqi Yang; Yu Cao; , "An efficient method to identify critical gates under circuit aging," *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on* , vol., no., pp.735-740, 4-8 Nov. 2007

[33] Weste, Neil and Harris, David, *Principles of CMOS VLSI Design - A Circuits and Systems Perspective, 3rd Edition* Addison-Wesley, MA, May 2004

[34] Zhihong Liu; McGaughy, B.W.; Ma, J.Z.; , "Design tools for reliability analysis," *Design Automation Conference, 2006 43rd ACM/IEEE* , vol., no., pp.182-187, 0-0 0

[35] K. O. Jeppson and C. M. Svensson, "Negative bias stress of MOS devices at high electric fields and degradation of MNOS devices," *J. Appl. Phys.*, vol. 48, no. 5, pp. 2004–2014, May 1977.

[36] Ogawa, S. and Shiono, N., Generalized diffusion-reaction model for the low-field charge-buildup instability at the Si-SiO2 interface. Physical review B. v51. 4218-4230.

[37] Chakravarthi, S.; Krishnan, A.; Reddy, V.; Machala, C.F.; Krishnan, S.; , "A comprehensive framework for predictive modeling of negative bias temperature instability," *Reliability Physics Symposium Proceedings, 2004. 42nd Annual. 2004 IEEE International* , vol., no., pp. 273- 282, 25-29 April 2004

[38] T. Grasser, H. Reisinger, P-J. Wagner and B. Kaczer, " Time-dependent defect spectroscopy for characterization of border traps in metal-oxide-semiconductor transistors", Phys. Rev. B, Vol. 82, No. 24, pp. 5318-5327 (2010)

[39] Lenahan P and Conley J 1998 What can electron paramagnetic resonance tell us about the Si/SiO_2 system *J. Vac. Sci. Technol.* B 16 2134-53

[40] T. Grasser , B. Kaczer , W. Goes , T. Aichinger , P. Hehenberger and M. Nelhiebel "A two-stage model for negative bias temperature instability", *Proc. Int. Rel. Phys. Symp.,* , 2009.

[41] Dimitrios Rodopoulos, "Time and Workload Dependent Device Reliabilityin Circuit Simulations", Master Thesis Internship-Thesis Report, October 2010-April 2011, IMEC vzw, Belgium

Publications

Conference Papers

[CP-1] **D. Rodopoulos**, S. B. Mahato, V. Valduga de Almeida Camargo, B. Kaczer, F. Catthoor, S. Cosemans, G. Groeseneken, A. Papanikolaou, D. Soudris, "Time and Workload Dependent Device Variability in Circuit Simulations", International Conference on Integrated Circuit Design and Technology, April 2011 IEEE International

[CP-2] Siozios Kostas, **Rodopoulos Dimitrios** and Soudris Dimitrios, «Quick_Hotspot: A Software Supported Methodology for Supporting Run-Time Thermal Analysis at MPSoC Designs,» accepted presentation in 2nd PARMA Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures, co-located with ARCS 2011 - Architecture of Computing Systems, 23 February 2011, Lake Como, Italy.

Journals

[JP-1] Siozios Kostas, **Rodopoulos Dimitrios** and Soudris Dimitrios, «Quick_Hotspot: A Software Supported Methodology for Supporting Run-Time Thermal Analysis at MPSoCs», Microprocessors and Microsystems, Elsevier (invited paper)

[JP-2] Siozios Kostas, **Rodopoulos Dimitrios** and Soudris Dimitrios, «On Supporting Rapid Thermal Analysis», IEEE Computer Architecture Letters (accepted)

Appendix A

"Time and Workload Dependent Device Variability in Circuit Simulations"

Presented in the

IEEE International Conference on IC Design and Technology

3 May 2011

Kaohsiung, Taiwan

Time and Workload Dependent Device Variability in Circuit Simulations

D. Rodopoulos[1,2], S. B. Mahato[1,4], V. Valduga de Almeida Camargo[1,5], B. Kaczer[1], F. Catthoor[1,3], S. Cosemans[1,3], G. Groeseneken[1,3], A. Papanikolaou[2], D. Soudris[2]

[1]IMEC, Leuven, Belgium [2]NTUA, Greece [3]KU Leuven, Belgium [4]TU Munich, Germany & NTU, Singapore [5]UFRGS, Brazil

email: drodo@microlab.ntua.gr

***Abstract*—Simulations of an inverter and a 32-bit SRAM bit slice are performed based on an atomistic approach. The circuits' devices are populated with individual defects, which have realistic carrier-capture and emission behaviour. The wide distribution of defect time scales, accounts for both fast (Random Telegraph Noise – RTN) and near-permanent (Bias Temperature Instability – BTI) defects. The atomistic property of the model allows the detection of workload dependency in the delay of both circuits.**

***Index Terms*—Bias-temperature instability (BTI), circuit simulations, parametric reliability, random telegraph noise (RTN), static random access memory (SRAM), workload dependency**

I. Introduction and Related Work

Most of the existing BTI simulation approaches, either in terms of modeling or mitigation, employ only the averages of BTI-related parameters. Versions of the reaction-diffusion model appear to be quite popular [5]-[9]. BTI is translated into degradation parameters, which are added to each transistor, thus altering its transient response. The parameters are constant and do not change throughout the simulation of the "aged" circuits [5].

In some cases, averaging of the stress applied on each device is employed [8]. In other approaches, there is poor incorporation of the recoverable part of BTI [9]. This suppression can result in over-estimation of the impact of the BTI mechanism [5]. In a reliability aware design context, this can lead to over-constrained designs.

Devices of older technologies have sufficient size and a large number of defects, thus exhibiting uniform reliability behaviour. As the device dimensions decrease, the stochastic defect behaviour becomes gradually more evident [1], [2].

As previously demonstrated, it is possible to monitor the occupancy of *individual* oxide defects [1], [2]. If this atomistic model is applied to any circuit, it is possible to identify the effect of the defect activity on the operation parameters of the circuit, like the circuit's delay or leakage energy. That way, the circuit's parametric reliability can be studied.

In view of the above, it is increasingly important to shift to a combined deterministic-stochastic view of reliability related phenomena, such as Bias Temperature Instability (BTI) or Random Telegraph Noise (RTN). The deterministic quality refers to the model's workload dependency, which is based on the gate voltage dependent trap behaviour (not just on the average duty cycle of the ones or zeros). The stochastic component mirrors the probabilistic nature of oxide defect activity. Only that will allow observing and analyzing true workload dependent behaviour. Moreover, that analysis can be the basis of mitigation approaches based on workload tuning. Enabling these important objectives by providing the basic modeling framework is the main differentiator of this paper.

The next section covers the atomistic defect model. The third section contains information on the modules used to illustrate the novel model. Finally, the simulation results are presented along with some brief conclusions.

II. The Atomistic BTI Model

A. Theoretical Model

The atomistic model that is used in the simulations is consistent with the device downscaling trend toward nm dimensions, since it treats each defect of each device separately. Each device is enhanced with a random number of traps, extracted from a Poisson distribution (an average of 10^{12} traps per cm^2 is used). Each trap is characterized by a different threshold voltage impact ($\Delta V_{th} > 0$) and a different set of time constants $\{\tau_{cH}, \tau_{cL}, \tau_{eH}, \tau_{eL}\}$ resembling the capture or emission times for high (H) or low (L) voltages at the device's gate (V_{gs}). Our setup determines each defect's occupancy transitions *during the circuit simulation* using

$$P_{p,v} = \frac{\tau_{\bar{p},v}}{\tau_{e,v} + \tau_{c,v}} \left\{ 1 - \exp\left[-\frac{1}{\tau_{e,v}} + \frac{1}{\tau_{c,v}} \right] \Delta t \right\}, \quad (1)$$

where the process p ($\bar{p}$ being the complement) can be either a capture (c) or an emission (e) event. Δt is the simulation step and v can be either of the $\{H, L\}$ voltage levels.

For every simulation step, a random number is compared to the process probability (1). If the random number is found smaller than the probability, the respective process occurs. When a defect becomes *occupied* (i.e. a carrier is captured in the trap), the respective ΔV_{th} value is added to the runtime V_{th} value of the device. Further details on the model and its assumptions can be found in [2].

The distributions of the *defect time scales* are taken from experiments. The interval of time constants is sufficient to account for a wide variety of defect behaviours. Hence, it is possible to observe fast trapping and detrapping events for a single defect (corresponding to *RTN*). It is also possible to observe near-permanent behaviours with larger time constants (corresponding to *BTI*).

B. Netlist Annotation Framework

The simulations presented in the current paper account only for NBTI, but PBTI can also be easily included. For the purpose of the simulations, a framework is required with which to populate the netlist under test with the defects, their time constants and their V_{th} impact. This annotation is performed during pre-processing (Fig. 1). Any sub-circuit is expanded down to device level. Each device is annotated with NBTI parameters alongside the parameters provided by the PTM model [3]. The transient part of the atomistic model is added on top of a Verilog-A implementation of the BSIM4 model.

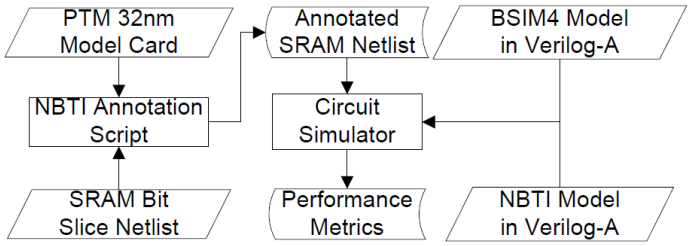

Fig. 1. Generic flowchart of the netlist annotation framework.

C. Workload Dependency

In commercial reliability tools like RelXpert [5], each device is replaced by an aged equivalent, based on the input stress provided to the simulator. Then, a simulation of the "aged" circuit is performed, thus indicating circuit degradation. There is no information provided about the *runtime defect activity*. Such approaches are unable to present a transient view of reliability phenomena. The purpose of this paper is to differentiate the atomistic BTI model from the state-of-the-art BTI approaches. *The main differentiator is the time-dependent workload dependency*. Hence, *all simulations need to assume time and workload as the only independent variables*.

The NBTI model requires a random number per iteration, which is compared to the V_{gs} dependent probability for a capture or emission event. The workload for each device corresponds to the imposed V_{gs} at any given time.

In the default implementation, no correlation is present between the generated random numbers and the imposed V_{gs}. If we bind the random number generation to the imposed V_{gs} (Fig. 2), we decouple the workload dependency from the model's stochastic component.

The initial seed values for the random number generation should remain the same, irrespective of the imposed workload.

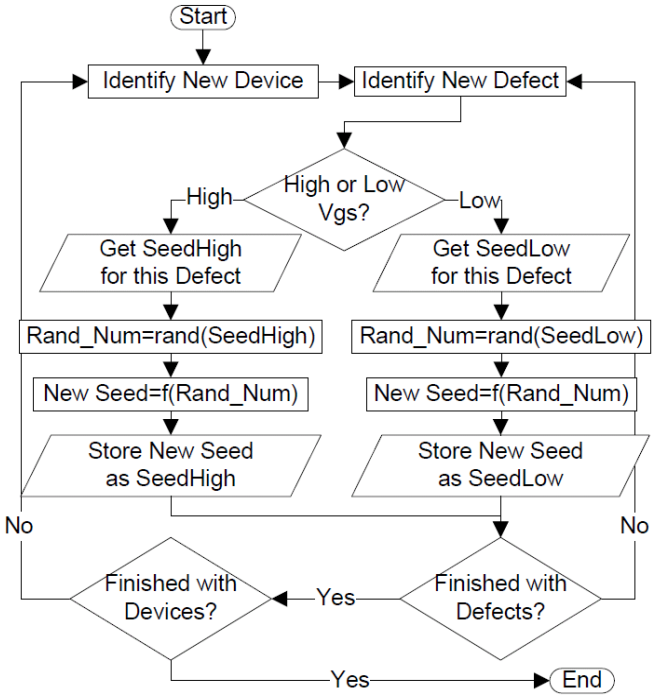

Fig. 2. Customized flowchart of random number generation.

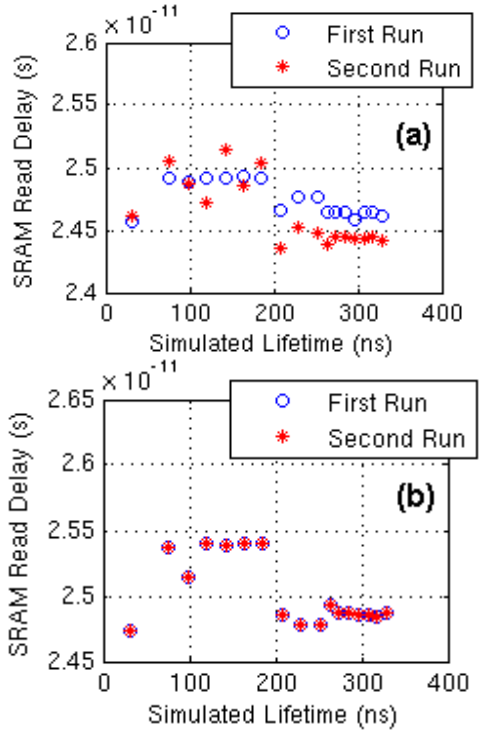

Fig. 3. Two simulations of the same workload: (a) Default implementation with an evident stochastic component and (b) If workload dependency is emphasized, the simulation outputs coincide.

The annotation of the netlist with NBTI parameters has to be performed just once.

In order to clarify the need for the above configurations, we simulate the same SRAM partition workload two consecutive times (Fig. 3). As output of the simulations, we consider the

observed delay for the SRAM read operation. If we use the above configurations, we inspect the same output in both runs, since *the workload is the same for both* (Fig. 3b). In the opposite case, the stochastic component is deeply routed in the simulations, so the output is different (Fig. 3a).

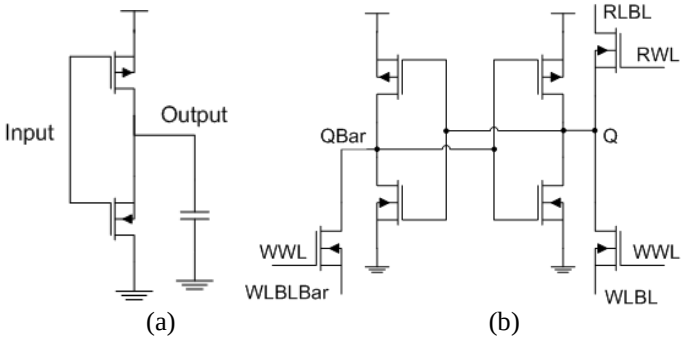

Fig. 4. Schematic of the inverter (a) and of an SRAM cell (b).

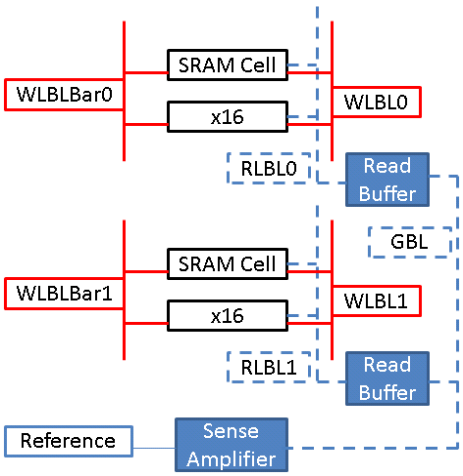

Fig. 5. Schematic of the SRAM Partition (reading path is dashed).

III. Case Studies

A. Inverter

In this case, we use a simple inverter (Fig. 4a) and measure its delay. Since the pull-up branch is affected by NBTI, we expect larger delays for measurements during a 1-0 transition at the input.

B. SRAM Bit Slice

This circuit is an SRAM bit slice of 32 bits, divided in two groups of SRAM cells, based on [4].

The two complementary nodes of each SRAM cell (Fig. 4b) are connected to voltage sources through access transistors. Writing is implemented by applying a word pulse to the respective transistors while maintaining the necessary voltage values at WLBL and WLBLBar. For the reading operation, the two groups of cells have single local read bit lines (RLBL0 and RLBL1) which are connected to a global bit line (GBL) through read buffers (Fig. 5). The GBL is then connected to a sense amplifier (SA). The RLBLs and the GBL are always pre-charged to a specific voltage and will be discharged to V_{ss}, when a stored 0 is read.

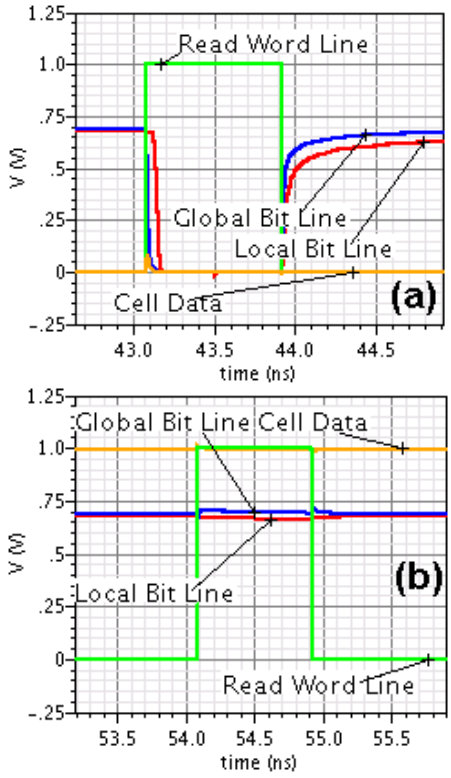

Fig. 6. Instances of the reading operation, either for logic 0 (a) or logic 1 (b). Evidently, only the second case allows a delay measurement.

The performance metric is the *delay of the read operation*. We measure the time from the activation of a word line up to the point where the voltage difference at the inputs of the SA, is enough for the latter to sense (Fig. 5). In the current SRAM organization, this metric can be applied only to cases when the read value is logic 0 (Fig. 6).

```
while(user_defined) {

  /* Write cells consecutively */
  for (i=0; i<=31; i++) {
    write(cell[i]);
  }

  /* Read cells consecutively */
  for (i=0; i<=31; i++) {
    read(cell[i]);

    /* Global Retention in-between */
    for (j=1; j<=10; j++) {
      nop
    }
  }
}
```

Fig. 7. Algorithmic description of the simulated SRAM activity.

C. Stimuli Setup

In the inverter case, we define a specific sequence of bits applied at the input of the inverter as a runtime situation (RTS). The frequency of this sequence is 1GHz.

In case of the SRAM partition, each of the cells can be in one of the Read, Write or Retain operation modes. If a cell is

not being read or written, it retains its value. All operation modes are assumed to have the same duration. As long as the consistency of the cells' internal state is maintained (e.g. one cannot read from a cell that has not been previously written), an arbitrary sequence of operation modes can exist, with a user defined frequency of 1GHz. The SRAM activity can be described by a handful of pseudo-code lines (Fig. 7). In the SRAM partition case, a different sequence of values written in the cells indicates a different runtime situation (RTS).

Each control signal or stimulus is defined as a piece-wise linear source, which is set according to the inspected RTS. This setup is performed during pre-processing and is incorporated into the annotation framework presented in section II (Fig. 8).

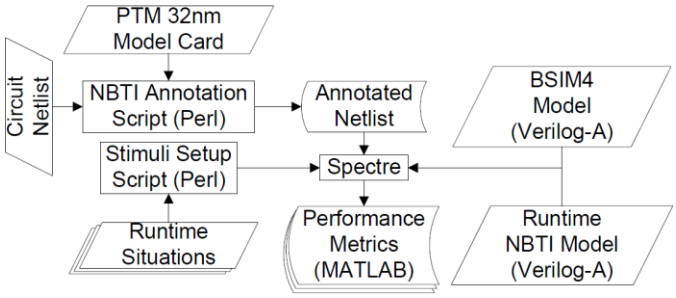

Fig. 8. Customized flowchart of the simulation framework (including automated stimuli setup).

IV. Simulations

A. Inverter

Based on a bit sequence of 200ns (Reference Workload), we change either the first ("Past") or the last ("Future") 100ns of the sequence (Fig. 9). *The results demonstrate a distinct proof of the model's detailed workload dependency*, which is not averaged out by a pure stochastically based device model. The simulation output of the Changed "Future" is identical to the Reference Workload output up to the point that the bit sequence is changed. In contrast, if we only change the "Past",

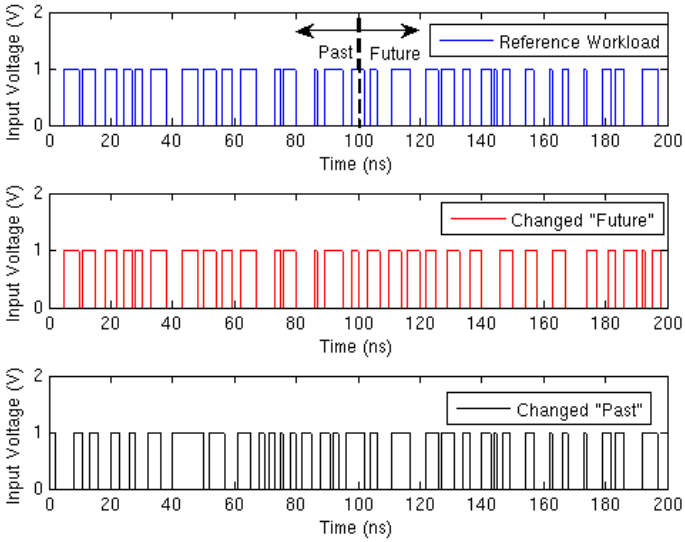

Fig. 9. Runtime changes on the inverter's workload.

the simulation output is entirely different (Fig 10).

What we demonstrate here is that *the atomistic BTI model has a deterministic workload memory. The observed NBTI effect at any time strongly depends on the specific workload that has preceded the current operation state. We can also analyze this on a cycle-by-cycle deterministic basis, which is unique compared to existing circuit-level models.*

B. SRAM Bit Slice

Two entirely different runtime situations are simulated (Fig. 11). Each workload is applied to the NBTI enhanced netlist and to a reference netlist without any oxide defects. No initial stressing of the devices is assumed; all defects are not occupied at the beginning of the simulation. The delay fluctuations differ between the two workloads, in the NBTI Enhanced case. On the contrary, the delay fluctuations are almost identical between the two RTSs, for the Reference netlists.

In another round of simulations, we use the analytical model presented in [2] to stress all the devices of the netlist with an AC signal (80% duty cycle for $-V_{gs}$) for 10^7 seconds. That way, we initialize the trap occupancy in the NBTI Enhanced netlist. A Reference netlist, without defect activity, is also used. Two different RTSs are simulated in both netlists (Fig. 12). We observe that the delay fluctuations in the NBTI Enhanced netlist occupy a much wider interval, than the Reference counterpart. The increased variability of the NBTI Enhanced netlist surpasses the usually accepted 10% criterion.

V. Conclusions

By monitoring separately every defect of every simulated device, we can see the workload dependent nature of the atomistic BTI model, propagating to the delay of the aged circuit. Even if we change the workload at runtime, we immediately see a difference in the evolution of the circuit's delay. When NBTI is accounted for, the delay measurements

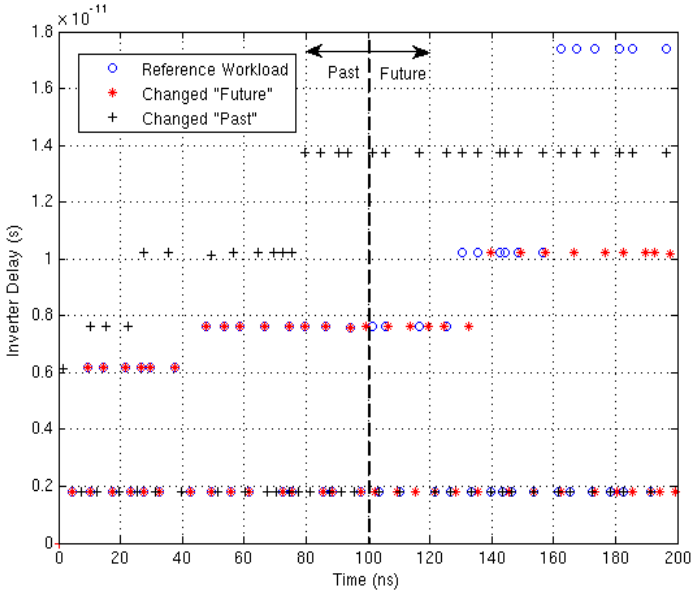

Fig. 10. Delay measurements on the inverter.

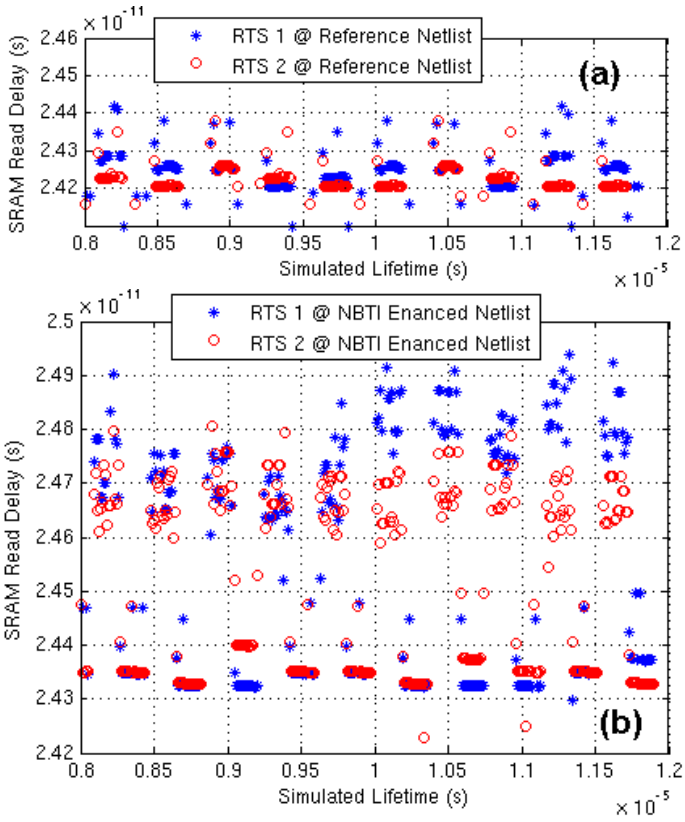

Fig. 11. Runtime delay fluctuations of an SRAM partition for two different RTSs. No initial stressing is assumed (time zero simulation). The delay measurements are almost identical in the Reference netlist (a). For a netlist enhanced with defect activity (b), the fluctuations exhibit greater runtime variability and also differ between the two workloads.

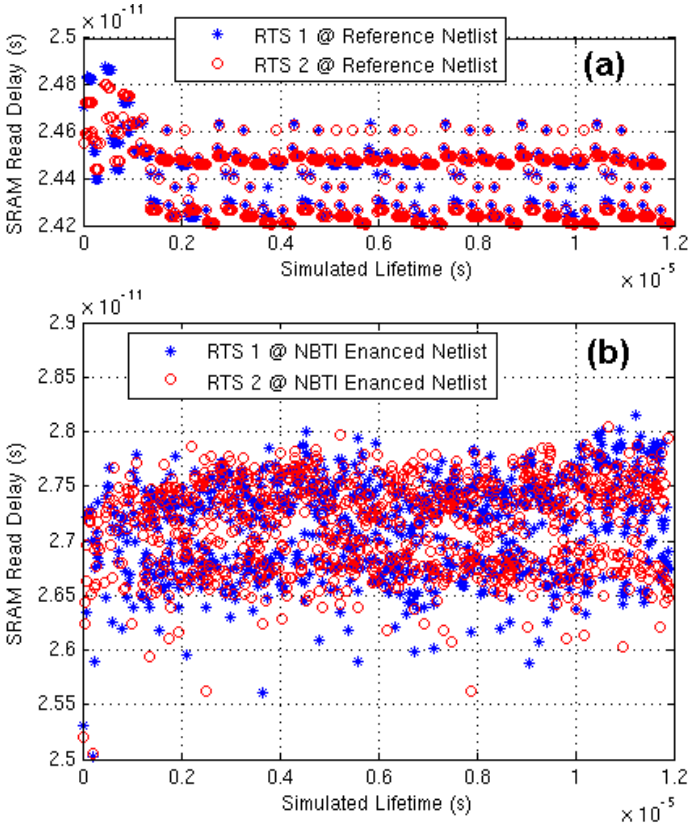

Fig. 12. Runtime delay fluctuations of an SRAM partition for two different RTSs. Initial AC stressing is assumed. When NBTI is not considered (a), the fluctuations are concentrated in a very small interval. In a netlist enhanced with defect activity (b), the delay fluctuations cover a much wider interval.

of the SRAM partition are distributed in a much broader interval.

As a result, the two test cases of this paper illustrate and substantiate the deterministic component of our model. This approach allows the modelling of individual trap behaviour based on the actual input stimuli sequence (and not just on the duty cycle of the ones or zeros as seen in other approaches).

The challenge that lies in view of these conclusions is to start searching for correlations between the imposed workload and its observed impact on performance metrics. That way, a more realistic view of the parametric reliability of larger circuits can be obtained. The current defect model incorporates both the stochastic and the workload dependent nature of oxide traps. Hence, it is a suitable tool with which to explore parametric reliability in a realistic and detailed workload dependent way.

ACKNOWLEDGMENTS

Part of this work was carried out in IMEC's Industrial Affiliation Program funded by IMEC's core partners. V.V.A.C. thanks CNPQ Brazil for financial support.

Discussions with Prof. T. Grasser about the physical aspects of gate oxide charge trapping are gratefully acknowledged.

REFERENCES

[1] Kaczer, B.; Grasser, T.; Roussel, P.J.; Franco, J.; Degraeve, R.; Ragnarsson, L.; Simoen, E.; Groeseneken, G.; Reisinger, H.; , "Origin of NBTI variability in deeply scaled pFETs," *Reliability Physics Symposium (IRPS), 2010 IEEE International* , vol., no., pp.26-32, 2-6 May 2010

[2] Kaczer B. *et al.*, "Atomistic approach to variability of bias-temperature instability in circuit simulations", *Reliability Physics Symposium (IRPS), 2011 IEEE International* (accepted)

[3] http://ptm.asu.edu/ .

[4] Cosemans, S.; Dehaene, W.; Catthoor, F.; , "A 3.6 pJ/Access 480 MHz, 128 kb On-Chip SRAM With 850 MHz Boost Mode in 90 nm CMOS With Tunable Sense Amplifiers," *Solid-State Circuits, IEEE Journal of* , vol.44, no.7, pp.2065-2077, July 2009

[5] Zhihong Liu; McGaughy, B.W.; Ma, J.Z.; , "Design tools for reliability analysis," *Design Automation Conference, 2006 43rd ACM/IEEE* , vol., no., pp.182-187, 0-0 0

[6] Calimera, A.; Macii, E.; Poncino, M.; , "Analysis of NBTI-induced SNM degradation in power-gated SRAM cells," *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on* , vol., no., pp.785-788, May 30 2010-June 2 2010

[7] S. Khan, S. Hamdioui, Temperature Impact on NBTI Modeling in the Framework of Technology Scaling, Digest of the 2nd Design For Reliability (DFR 10), PISA, Italy, Jan 2010.

[8] Kumar, S.V.; Kim, C.H.; Sapatnekar, S.S.; , "NBTI-Aware Synthesis of Digital Circuits," *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE* , vol., no., pp.370-375, 4-8 June 2007

[9] Paul, B.C.; Kunhyuk Kang; Kufluoglu, H.; Alam, M.A.; Roy, K.; , "Negative Bias Temperature Instability: Estimation and Design for Improved Reliability of Nanoscale Circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.26, no.4, pp.743-751, April 2007

Appendix B

"Quick_Hotspot: A Software Supported Methodology for Supporting Run-Time Thermal Analysis at MPSoC Designs"

2nd PARMA Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures

co-located with ARCS 2011 - Architecture of Computing Systems

23 February 2011

Lake Como, Italy

Quick_Hotspot: A Software Supported Methodology for Supporting Run-Time Thermal Analysis at MPSoC Designs

Kostas Siozios, Dimitris Rodopoulos and Dimitrios Soudris
School of Electrical and Computer Engineering, National Technical University of Athens, Greece

Abstract

Detailed thermal analysis and exploration has recently received significant attention since it is straightforward-related to numerous reliability issues. Furthermore, thermal profiling is a critical challenge for supporting efficient power management, especially to multi-processor system-on-chips (MPSoCs). This problem becomes even more important if we take into account the computational complexity of existing thermal profiling and analysis approaches. Among others this limitation imposes that thermal analysis is performed solely at design time. However, such a static exploration does not take into account constraints posed during application execution that lead to temperature variations. Hence, new algorithms and software tools able to provide accurate yet fast thermal analysis are upmost required. In this paper, we introduce a new software supported methodology for performing thermal analysis at run-time with different levels of granularity. Additional performance improvement is feasible by applying thermal analysis only to device regions with blocks that operate under high power densities. For demonstration purposes we show how this methodology is applied to an Altera Stratix-based FPGA device. Experimental results prove the efficiency of the proposed methodology, since the average execution time ranges between 41% and 78%, as compared to state of the art relevant solution, without any accuracy degradation at the derived thermal profile.

1 Introduction

Shrinking silicon technologies, increasing logic densities and clock frequencies leads to a rapid elevation in power density, while according to "*A-power*" law [1] the temperature stress is going to become more severe for technology nodes at 65nm and below.

Accurate and detailed thermal profiling for integrated circuits (ICs) is an important design challenge, mainly for three reasons: (i) temperature is closely related to reliability degradation, hence, there is an urgent need for solutions able to estimate such degradation, (ii) the information regarding spatial locations of power sources is not enough for applying efficiently cooling techniques (due to the complexity of existing devices), whereas the increased packaging cost prevents consumer products from being designed for the worst case scenario, and (iii) leakage current increases exponentially with temperature, causing a positive feedback loop between leakage power and temperature.

Recently, the accurate thermal analysis has been recognized as an important design problem that has to be tackled as soon as possible [4]. To make matters worse, new design technologies (e.g., three-dimensional integration) will depreciate thermal issues.

Thermal monitoring and profiling in integrated circuits is treated with two alternative ways: either by inserting thermal sensors in selective spatial locations of the chip's surface and read their output, or by inserting power sensors and compute thermal profiling through algorithmic approaches. Both of these solutions have advantages and disadvantages that should be carefully taken into consideration during the design of a new architecture. More specifically, the insertion of thermal sensors alleviates the requirement for performance overhead at run-time in order to compute temperature values but it allows monitoring only a limited area of the device. On the other hand, the power sensors impose that an algorithmic approach will be used at run-time to compute thermal profiling, but the retrieved solution can describe accurately the temperature variations across the entire architecture.

Up to now, in order to support thermal analysis at run-time, numerous techniques that try to alleviate the consequences posed by increased temperature values have been proposed. These solutions span from hardware level (e.g., thermal-aware floor-planning [5]), to low-power design methodologies at circuit level (e.g. [6]), as well as strategies for dynamic power/thermal management (e.g. [7], [8]).

The previously mentioned techniques can prevent target architecture from failure due to thermal stress. However, the majority of them is applied statically only at design time. In addition to that, even though some of these solutions can be extended to support thermal management techniques at run-time, usually the employed strategies are based on pre-defined scenarios.

This work was supported by the HiPEAC Grand entitled "On Providing Dynamic Reliability Improvement in FPGA

Hence, almost the majority of existing techniques exhibit serious limitations regarding their efficiency, as they do not take into consideration constraints posed at run-time that alter on-chip temperature values. For instance, the operating conditions for many devices (e.g. portable, multimedia, wireless, etc.) depend mainly on the usage. Regarding these devices, scenario based thermal management cannot be though as an acceptable solution.

As we will depict later, this static temperature exploration mainly occurs due to increased computational complexity for performing detailed thermal analysis during application's execution. Consequently, run-time thermal analysis and exploration, which is performed in conjunction to application's execution, is upmost required.

In this paper, we introduce a novel software supported methodology for performing fast and accurate thermal analysis, exploration and management of MPSoCs. The main differentiation of this solution, as compared to existing approaches is the significantly reduced computational complexity. This comment in conjunction to the continue increasing number of processing cores in existing MPSoCs, enables the opportunity to perform run-time thermal analysis even in such architectures.

Since the complexity of the proposed solution is significantly lower as compared to state of the art approaches found in relevant references (e.g. *Hotspot* [8]), it can efficiently support techniques for run-time thermal analysis and/or management due to:

- existing MPSoCs include a number of processing cores which can support the thermal analysis task.
- there is no need for performing thermal analysis very often since temperature variations is not a temporally rapid procedure.
- the computational complexity, and hence the execution time, of proposed solution is significant lower as compared to existing approaches.
- it is possible to perform focused (or localized) thermal analysis only at regions with increased power densities rather than computing temperature values for the whole chip, in order to further speedup the execution time.

Note that even though the last bullet leads to mentionable performance improvement, however it should be applied very carefully since it might result to loss of accuracy. We have to mention that, this performance improvement does not affect the detail of derived thermal profiles. More specifically, based on experimental results we achieve thermal analysis of complex MPSoCs about 31% faster than *Hotspot* version 5.0 (latest available version) [8], while the maximum temperature error is only 0.03º C.

The contributions of this work are summarized, as follows:

- Introduction of a novel methodology for detailed thermal profiling and exploration of MPSoCs under run-time constraints.
- Introduction of a new algorithm, as well as the open-source tool that software supports the proposed methodology. This algorithm is based on the Hotspot engine.
- The introduced tool, named *Quick_Hotspot*, achieves faster thermal analysis and exploration without any loss of accuracy, as compared to state-of-the-art solutions

The rest of the paper is organized as follows: Section 2 describes the proposed methodology for run-time thermal analysis, whereas the software tool that supports this task is discussed in section 3. In section 4 we give a number of qualitative and quantitative comparison results that prove the efficiency of our proposed methodology. Finally, conclusions are summarized in section 5.

2 Proposed Thermal Analysis and Exploration Methodology

The proposed thermal analysis and exploration methodology seeks to accurately compute on-chip temperature values during the continuous execution of an MPSoC. This methodology, shown in Figure 1, consists of two steps, namely (i) the coarse-grain, and (ii) the fine-grain thermal analysis, while it is applicable either as a monitoring agent at an embedded operating system (OS), or as a stand-alone tool for thermal analysis.

2.1 Coarse-grain thermal analysis

During the first step of the proposed methodology (coarse-grain thermal analysis), we are primarily interested in retrieving a rough and fast estimation of the temperature variation across the chip's area. In other words, the task we try to accomplish during coarse-grain thermal analysis deals with the identification of regions of importance, where detail thermal analysis should be applied (during the second step of proposed methodology).

More specifically, having as input the application's placement and routing (P&R), as well as the total power consumption (static and dynamic) for all the hardware resources of target architecture (both logic and interconnection infrastructure), it is possible to compute how these power sources are distributed over the target architecture. This task does not introduce mentionable computational complexity, since it only parses and manipulates placement, routing and power data.

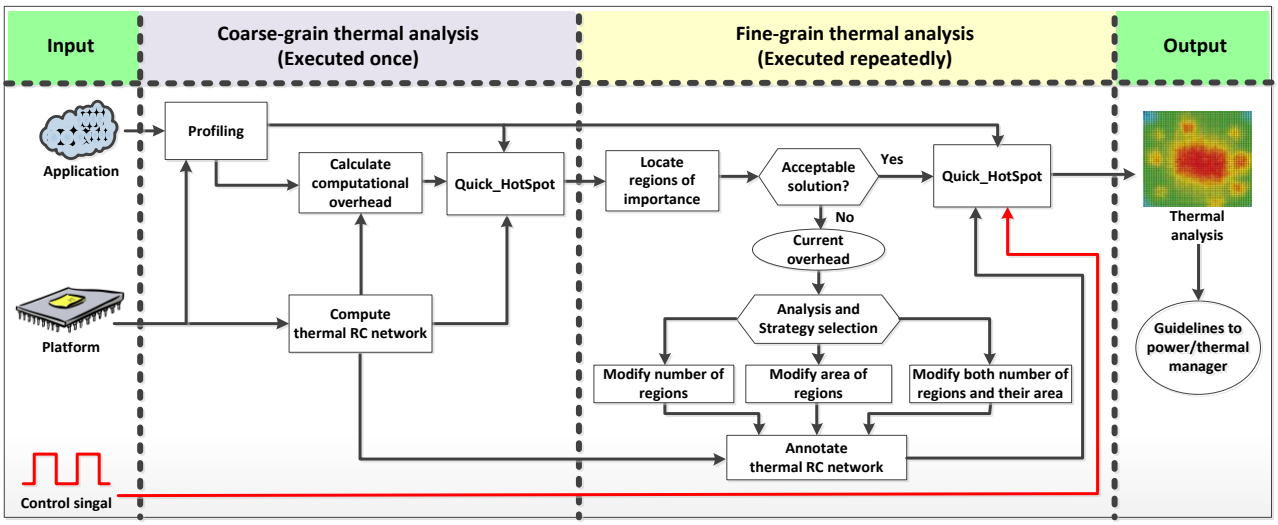

Figure 1 Proposed methodology for thermal analysis

Also, during the coarse grain step, we compute a template of the thermal *RC* network, which describes the way that thermal diffusion is distributed over the chip. The granularity of this network corresponds to the desired detail of the derived thermal analysis. Even though highest detail gives more info about the chip's thermal stress, it also imposes the maximum computational effort, which is not affordable for some application domains.

Existing tools (e.g. [8]) compute the thermal *RC* network by assuming that each distinct core represents a power source. Such an approach leads to a coarse grain thermal analysis, which usually differs a lot, as compared to the actual chip's thermal stress. This problem becomes even more important when critical cores for thermal analysis have increased area and power density values. This occurs mainly due to the fact that existing tools assume that power source is assigned in the middle of the core's area, ignoring about its actual distribution inside core (which might alter considerable the final thermal map).

Moreover, as we will prove later, the calculation of the thermal *RC* network is a computationally intensive task, which needs to be performed only once (in contrast to existing approaches where its calculation is performed any time a thermal analysis is required).

In order to alleviate these consequences, our approach relies on a scalable template of an *RC* network that is easily modifiable and/or expandable. This new network is calculated once at macro-block level, and whenever different accuracy (or detail) of the derived thermal profile is required, it is just annotated. We will prove later that such an approach is much faster than computing the thermal *RC* network from scratch.

Furthermore, this speedup is even more important when the proposed thermal analysis tool is executed at run-time, since it alleviates the requirement for processing cores that perform computations with increased complexity.

Having the spatial distribution of power sources over the target platform, in conjunction to the template of *RC* model, it is possible to estimate the computational overhead (in term of execution time) for performing thermal analysis under the selected granularity level.

Next, we perform a coarse-grain thermal analysis in order to retrieve a rough and global estimation of temperature variation over the chip's area. This task is software supported by a new tool, named *Quick_Hotspot* [2]. Upcoming sections will prove that for the majority of applications, such a coarse grain thermal analysis is enough to determine regions of importance. Furthermore, in case of detail thermal analysis (which is applied uniformly over the entire MPSoC platform) does not provide any further accuracy improvement apart from the mentionable computational overhead.

2.2 Fine-grain thermal analysis

The output of this rough thermal analysis is fed as input to the second step of the proposed methodology that deals with fine-grain thermal analysis. More specifically, based on the already derived rough temperature estimation, our methodology locates regions of importance. These regions cluster hardware resources that operate under temperature values higher than $T_{threshold}$. Note that the absolute value of threshold is defined by designer and depends on: (i) the maximum operating temperature, (ii) the desired level of temperature for applying thermal management, and (iii) the affordable number of hardware resources where we apply thermal analysis (based on the available computation resources).

As we have already mentioned, such a thermal stress is closely related to numerous aging phenomena, while in order to prevent the device from failures,

these resources require better temperature monitoring. We have to mention than it is not possible to employ a unique threshold value for all the target architectures (or for all the operation conditions), since applications usually need to incorporate multiple thermal management policies.

Having as input the number, the area of regions of importance, as well as their spatial distribution over the target platform, the second step of the proposed methodology estimates the new (updated) computational complexity regarding the thermal profiling problem. In case this complexity is affordable for the underline MPSoC platform, thermal analysis is performed periodically (e.g. whenever the power manager of the embedded OS triggers a control signal).

Otherwise, if the MPSoC platform does not have enough free computational resources to support the desired granularity level of thermal analysis, our methodology provides a feedback mechanism that retrieves (at run-time) the level of granularity that better meets available computational resources. In order to accomplish this task, three candidate strategies can be applied: (i) modify the number of regions that are monitored by appropriately increasing $T_{threshold}$, (ii) modify the area that occupy regions of importance, and (iii) apply a combined version of the two previously mentioned strategies.

In case the embedded OS does not support dynamic strategy selection, the order of applied strategies has to be predefined at design time by taking into consideration the computational overhead posed during thermal analysis (assuming average traffic at primary inputs).

Based on the selected strategy, the template of the thermal *RC* network (as it was already computed during coarse-grain thermal analysis), is appropriately annotated. Even though it is possible to compute the *RC* network from scratch for the selected detail of thermal analysis, we prefer to annotate it, since the former task is a computationally intensive process (and probably it is not affordable for re-computing it under run-time constraints).

In addition to that, even though it is not possible to evaluate the on-chip temperature values for different resources of target architecture (due to thermal diffusion effect), it is still possible to locate regions of the device with increased probability of incurring high temperatures (based on the distribution of power sources). Hence, the annotation of *RC* network needs to be applied only into a quite small part of the entire architecture, which leads among others to smaller execution times.

The output of our methodology is a thermal analysis map that plots how temperature values are actually distributed across the target platform. Since we allow dynamically updated full-customized *RC* networks, the derived thermal analysis combines regions with different detail (or granularity).

This analysis gives guidelines to the power/thermal manager about when and where to apply cooling techniques, while the combination of regions with different detail reduces significantly the computational overhead, as compared to similar solutions.

Moreover, the proposed methodology can be integrated as part of a more complex embedded OS for monitoring techniques about controlling (or reducing) heat dissipation when the package's capacity is exceeded. Among others, these techniques can alleviate the requirement for costly thermal packaging.

3 Thermal Modelling and Tool Implementation

In order to compute the heat flow we rely on an equivalent thermal *RC* network, which represents the transient and steady state thermal behavior of hardware blocks [10]. Note that the employed approach is similar to those found in almost the majority of existing algorithms and tools aiming to thermal analysis (e.g. [7], [8], etc).

By taking into consideration the physical dimensions and thermal constants of MPSoC hardware resources, the above duality yields an ordinary differential equation (Equation 1), which describes the way that thermal energy is spread throughout the chip.

$$\frac{d\bar{T}}{dt} + C\bar{T} = A^{-1}\bar{P} \quad (1)$$

where A and C summarize the chip's thermal characteristics [11].

In order to software support the proposed methodology we have released a new tool which is available for download and further improvement through [2]. Rather than proposing a new thermal modeling approach, we extent a well established one (found in Hotspot-5.0 tool [8]), which is also acceptable for commercial designs. The derived tool, named *Quick_Hotspot*, supports accurate, yet fast, thermal analysis of complex MPSoCs.

As we discuss in upcoming sections, the new tool outperforms similar solutions found in literature since it can support custom thermal analysis. The term custom refers to thermal profiles that combine regions with different detail of accuracy, based on the trade-off between actual demand of hardware resources for temperature monitoring and the associated increment in computational complexity. This feature is especially crucial as it results to significantly lower computational complexity, while it takes into account the limited available computational resources of MPSoCs to perform run-time thermal analysis.

In order to meet these requirements, initially we distinguished whether there exists any further room for compromise between the current model's accuracy (found in *Hotspot*-5.0) and its computational complexity. Note that during this study, any processor-specific optimizations (e.g. use of acceleration engines) do not affect the validity of our claims. For this purpose, no accelerating engine is utilized during the profiling procedure. That is because our intention is to make the algorithm faster, while maintaining the same level of accuracy, and not make any processor-specific optimizations.

3.1 Existing solution (Hotspot)

Based on our study we found that we need to focus on the transient solver and more specifically in the *rk4_core* function, since it is the computationally dominant function at the *Hotspot* tool. This function performs an iteration of the fourth order *Runge-Kutta* method with adaptive step sizing, which is employed for solving the thermal diffusion problem, as it was described in Equation (1). This approach is applied to steady state solver both for block-based model, as well as the grid model. For a given time step, $t_{step} = h$, *HotSpot* performs a set of calculations until a time interval equals to $t_{interval}$ is covered. This time period (t_{step}) is defined by adaptive step sizing. That way, *HotSpot* can inspect in greater detail parts of the solution that change rapidly, while taking vast strides in smoother parts of the solution.

Being this an iterative process, the thermal analysis algorithm needs to read a power consumption vector every $t_{interval}$ and given the previous temperature and power vectors (mentioned as $\bar{T}_n$ and $\bar{P}_n$, respectively), it seeks to calculate the new temperature vector ($\bar{T}_{n+1}$). Equations (2) and (3) formulate the way that gradients and intermediate components of the *Runge-Kutta* method are calculated.

$$\left.\begin{array}{l} \bar{k}_1 = f(\bar{P}_n, \bar{T}_n) \Rightarrow \ \bar{t}_1 = \bar{T}_n + \dfrac{h}{2} \times \bar{k}_1 \\ \bar{k}_2 = f(\bar{P}_n, \bar{t}_1) \Rightarrow \ \bar{t}_2 = \bar{T}_n + \dfrac{h}{2} \times \bar{k}_2 \\ \bar{k}_3 = f(\bar{P}_n, \bar{t}_2) \ \Rightarrow \ \bar{t}_3 = \bar{T}_n + h \times \bar{k}_3 \\ \bar{k}_4 = f(\bar{P}_n, \bar{t}_3) \end{array}\right\} \quad (2)$$

$$\bar{T}_{n+1} = \bar{T}_n + \frac{h}{6} \times (\bar{k}_1 + 2 \times \bar{k}_2 + 2 \times \bar{k}_3 + \bar{k}_4) \quad (3)$$

where the vectors $\bar{k}_1$, $\bar{k}_2$, $\bar{k}_3$and $\bar{k}_4$ represent the gradients of the temperature graph for each node.

Let's assume that the gradients' calculation is performed by the function *evaluate_slope(Pn,Tn)*, which receives the power and temperature vectors as inputs. We also suppose that the vector $\bar{k}_1$ is available at the beginning of the thermal analysis. Based on the above assumptions, Algorithm 1 shows the pseudo-code for the function *rk4_core(Pn,Tn,k1,h)* which performs an iteration of the fourth order *Runge-Kutta* method, as implemented in *HotSpot-5.0*.

3.2 Intended Approach

The above set of calculations are repeated three times every new $t_{step} = h$, thus being the core of the numerical solution performed by *HotSpot-5.0* [8]. Furthermore, having profiled *HotSpot* for various benchmarks, it appears that the above numerical methodology occupies a significant part of the total execution time. Hence, this set of equations is a candidate for balancing the trade-off between computational effort (or execution time) versus accuracy of the thermal profile.

```
rk4_core(Pn,Tn,k1,h)   {
  t1 = Tn + h/2.0 * k1;
  k2 = evaluate_slope(Pn,t1);
  t2 = Tn + h/2.0 * k2;
  k3 = evaluate_slope(Pn,t2);
  t3 = Tn + h * k3;
  k4 = evaluate_slope(Pn,t3);
  Tnew = Tn + h*(k1 + 2*k2 + 2*k3 + k4)/6.0;
  return Tnew;
  }
```

Algorithm 1 Pseudo-code for computing thermal profile with existing algorithm (based on *Hotspot* tool)

We find that the error of the method employed by the original *Hotspot* implementation is $O(h^5)$ [12]. Given the very small value of the step $t_{step} = h$, even though it is dynamically changing, it is obvious to increase the error in favour of a faster execution. Thus, we conclude to employ the following simpler set of equations for each iteration.

$$\bar{k}_1 = f(\bar{P}_n, T_n) \quad (4)$$

$$\bar{T}_{n+1} = \bar{T}_n + h \times \bar{k}_1 \quad (5)$$

The error of the proposed implementation (namely the Euler method) is only $O(h^2)$ [12]. Given the small values of the time step, we can safely assume that the accuracy of the employed method remains at a similar level in comparison to the original *Hotspot* implementation. Note that this claim will be proven later in the experimental results.

Furthermore, given that each iteration has to be computed several times and for several cases in order to achieve adaptive step sizing, it is crucial that the corresponding part of the source code remains fast and yet accurate. Algorithm 2 gives the corresponding pseudo-code regarding our adopted thermal analysis algorithm, which replaces the *rk4_core* function found in the original *Hotspot*.

The computational complexity of the proposed solution is seriously decreased since each iteration requires 4× less the original number of operations. More specifically, in the original version, the complexity of the *rk4_core* function is dominated by repeated (3×) computations of gradients regarding

arrays with dimension *n*. The complexity of this task is $O(n^2)$. On the other hand, by replacing these computations with an addition, the new complexity is reduced to $O(n)$.

```
intended_approach(Pn,Tn,k1,h)   {
  Tnew = Tn + (h * k1);
  return Tnew;
  }
```

Algorithm 2 Pseudo-code for the proposed thermal profiling and analysis algorithm

Apart from this significant improvement in execution time, as we will depict in the next section, this performance speedup does not impose any accuracy degradation on the results about thermal analysis. Moreover, given that the proposed code alterations are purely algorithmic, we can safely claim that any additional acceleration methods (e.g. processor-specific acceleration engines) will only further improve the execution performance.

4 Experimental Results

This section provides numerous experimental results that evaluate the efficiency of the proposed thermal analysis methodology. Unfortunately, there is only one public available tool for thermal analysis, and hence we make comparisons solely against *Hotspot* [8].

Since it is more complex to perform physical synthesis for a representative number of IC designs (in order to extract the appropriate input files for thermal analysis), in this section we provide results about applications mapped onto FPGA platforms. More specifically, the target applications belong to benchmark suite [3], whereas the target device is an Altera Stratix-based FPGA. For each of these benchmarks we extract the corresponding floor-plan of the underline architecture based on datasheets provided from [9]. More info regarding the complexity of employed benchmarks, as well as the target platform can be found in Table 1.

Table 1 Complexity of benchmarks and specifications about the target FPGA

Benchmark	Slices	Power sources	Array of slices in target FPGA
alu4	1,600	1,600	40×40
apex2	2,025	2,025	45×45
apex4	1,369	1,369	37×37
diffeq	1,600	1,600	40×40
ex5p	3,025	3,025	55×55
misex3	3,721	3,721	61×61
s298	2,025	2,025	45×45
seq	1,849	1,849	43×43
tseng	1,156	1,156	34×34
Average:	**2,041**	**2,041**	**44×44**

For the scopes of this paper, we assume that each of the fabricated slices corresponds to a power source, while in order to compute the power consumption at a given slice, we summarize all the partial power sources inside it (e.g. LUTs, F/Fs, wires, etc) with models provided by [13].

4.1 Evaluation of Computational Complexity

Since the primary goal of our methodology is to support run-time thermal analysis, first of all, we evaluate the proposed solution, as compared to *Hotspot* [8], in terms of computational complexity. For this purpose, Table 2 summarizes the number of execution cycles required to perform thermal analysis with the two alternative approaches.

Table 2 Performance improvement for our proposed thermal analysis approach.

Benchmark	Number of cycles for performing thermal analysis ($\times 10^9$)		
	Hotspot	***Quick_ Hotspot***	**Gain (%)**
alu4	24,466	7,759	68.29%
apex2	23,215	15,876	31.62%
apex4	8,121	4,934	39.25%
diffeq	12,028	7,688	36.08%
ex5p	5,331	3,059	42.61%
misex3	10,644	6,655	37.47%
s298	22,660	15,701	30.71%
seq	17,541	11,726	33.15%
tseng	5,328	3,060	42.56%
Average:	**14,371**	**8,495**	**40.88%**

Based on these results we can conclude that, up to now, the detailed thermal analysis task is an extremely computation intensive procedure, since *Hotspot* requires about $14{,}371\times10^9$ execution cycles. On the other hand, *Quick_Hotspot* achieves an average reduction on the number of execution cycles about 41%. Such a significant reduction in computational complexity is especially crucial for supporting run-time thermal management, as embedded devices usually are limited in term of available computational resources.

In order to compute these numbers, for each benchmark we summarize the execution time plus any potential overhead of trifling procedures (like freeing memory blocks etc). Regarding our monitoring tool for extracting number of cycles, we can safely assume that the additional overhead is very small to be taken into consideration.

Note that both of these tools are fed with identical input files (placements and power traces), while, as we will depict later, our proposed solution leads to negligible error as compared to *Hotspot*. Moreover, the excessive high numbers of execution cycles are based on the requirement for computing detail thermal

analysis in slice level. This can also be shown by the average number of power sources (about 2,041) used as input for thermal analysis.

Even though such a detail analysis usually is beyond the requirements of embedded systems, there are many problems during platform design (e.g. reliability analysis, packaging selection, etc.) where such information is crucial. Furthermore, it is possible to reduce considerable the computational complexity, and hence the number of execution cycles, if we choose to apply a coarser thermal analysis (consisted of fewer power sources). Later, we will show how our methodology further reduces this number of execution cycles by applying selectively detail thermal analysis only at hardware resources belonging to regions of importance (with increased temperature values).

Next, by using some simple timing functions, we evaluate the time spent for the execution of various parts of the thermal profiling algorithm. For this purpose, we cluster the tool's functionalities into four parts:

- Parsing and Initial Configuration: Processing of the input commands and combination of the latter with the configuration file.
- *R* Model Population: Construction of an array with the vertical and lateral thermal resistances between the functional blocks. As we will depict, this step is very important and time-consuming.
- Names, Temperature and Power Arrays Initialization: Memory allocation and preparation of power and temperature arrays.
- Numerical Part: Numerical iterations.

The results of this analysis are summarized in Tables 3, 4, 5 and 6, respectively. The total number of cycles for each benchmark (as it was already shown in Table 2) is the summary of the partial number of execution cycles regarding each of the above steps.

Table 3 Number of execution cycles ($\times 10^9$) for parsing and initial configuration

Benchmark	***Hotspot***	***Quick_ Hotspot***	**Gain (%)**
alu4	1.02	0.54	47.2%
apex2	1.43	1.09	23.6%
apex4	0.44	0.41	7.3%
diffeq	0.51	0.52	-3.4%
ex5p	0.33	0.348	4.8%
misex3	0.48	0.46	3.9%
s298	1.24	1.56	25.5%
seq	1.12	1.04	6.8%
tseng	0.26	0.34	29.9%
Average:	**0.76**	**0.70**	**7.6%**

Based on the values provided in Tables 3-6, *R* model population and the numerical part are the steps with increased number of execution cycles. Since *R* model population is closely related to *RC* thermal modeling, this explains our choice to selectively annotate the *RC* model rather than computing it from scratch.

Our proposed *Quick_Hotspot* tools leads to reduction in term of complexity of the numerical part (as it is shown in Table 6) about 82%, as compared to conventional *Hotspot* tool. This occurs due to a different numerical approach used for solving the thermal diffusion problem.

Note that no accelerating engine is utilized throughout this paper. That is because our intention is to make the algorithm faster, while maintaining the same level of accuracy and not make processor-specific optimizations.

Table 4 Number of execution cycles ($\times 10^9$) for *R* model population.

Benchmark	***Hotspot***	***Quick_ Hotspot***	**Gain (%)**
alu4	6,751.3	6,704.7	0.69%
apex2	13,471.2	13,472.8	-0.01%
apex4	4,180.9	4,166.2	0.35%
diffeq	6,651.1	6,643.8	0.11%
ex5p	2,510.7	2,511.7	-0.04%
misex3	5,771.3	5,711.2	1.04%
s298	13,493.8	13,462.6	0.23%
seq	10,257.8	10,256.4	0.01%
tseng	2,508.9	2,512.2	-0.13%
Average:	**7,288.6**	**7,271.3**	**0.24%**

Table 5 Number of execution cycles ($\times 10^9$) for names, temperature and power arrays initialization.

Benchmark	***Hotspot***	***Quick_ Hotspot***	**Gain (%)**
alu4	5,471.4	6.2	99%
apex2	1,061.6	672.0	37%
apex4	4.21	4.2	0.11%
diffeq	5.7	5.7	0.0%
ex5p	3.06	3.1	-0.1%
misex3	5.2	5.2	0.0%
s298	509.1	388.4	23.7%
seq	57.2	53.8	5.88%
tseng	3.0	3.0	0.0 %
Average:	**791.2**	**126.8**	**84%**

4.2 Locate Regions of Importance

Up to now we have explored the complexity of the proposed methodology, as compared to a state of the art solution. Next, we evaluate the efficiency of our approach to locate regions of importance over the target platform. Figure 2 plots the thermal profile regarding *des* benchmark, as it is retrieved with *Quick_Hotspot*. The underline Altera Stratix-based FPGA platform consists of 65×65 slices, while thermal analysis was performed by computing power sources at slice level.

Different colors in this figure denote regions of the device that operate under different values of temperature. As closer to red color a region is, the corresponding hardware resources (slices) operate under higher temperatures. Regarding the *des* benchmark, thermal analysis reports that on-chip temperatures range from 75ºC up to 104ºC. For demonstration purposes, these temperatures are plotted in a normalized manner over the maximum temperature.

Table 6 Number of execution cycles ($\times 10^9$) for numerical part.

Benchmark	***Hotspot***	***Quick_ Hotspot***	**Gain (%)**
alu4	12,242.9	1,047.5	91%
apex2	8,681.5	1,730.1	80%
apex4	3,935.9	763.3	80%
diffeq	5,371.1	1,038.2	80%
ex5p	2,817.3	544.3	80%
misex3	4,867.9	938.8	80%
s298	8,656.8	1,849.0	78%
seq	7,225.3	1,415.6	80%
tseng	2,816.3	544.9	80%
Average:	**6,290.6**	**1,096.8**	**82%**

A number of conclusions might be derived based on this thermal map. Among others, temperature values are not constant over the target platform since they vary considerably between any two arbitrary points (x_1, y_1) and (x_2, y_2) of the device. Apart from this non-uniformity, it is still possible to determine regions with similar (high or low) temperatures, and hence increased (or not) probabilities of failure. Note that different applications exhibit different regions of importance, while even the same application might result to different thermal distributions (e.g. under power-aware, timing-driven, area-optimized P&R).

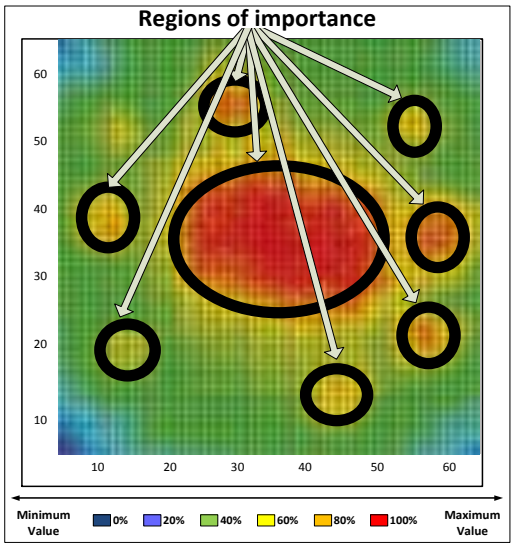

Figure 2 Hardware clustering based on temperature values

In this figure we have also highlighted (with a circle) regions of the device with increased temperatures values (greater than $\frac{T_{max}}{2}$). We have to mention that the three strategies discussed in Figure 1, which alleviate the computational complexity of thermal analysis (namely, modify number of regions, modify area of regions, and modify both number of regions and their area), are applicable to such a design by performing thermal analysis: in fewer circles, in circles that cluster fewer resources, and a combination of them, respectively.

Based on such a classification of hardware resources, another conclusion is drawn from Figure 2: although the majority of existing thermal management techniques is applied uniformly over the device, the actually critical for failure resources provide a non-homogeneous and irregular picture. Consequently, careful analysis of the points of failure must be performed, while target architecture needs to combine regions with different thermal/power management strategies.

Hence, the challenge with which a designer is faced up is to apply thermal analysis at run-time only to the actually needed parts of architecture, considering the associated spatial information from the distribution graph shown in Figure 2, as well as the consequence overhead due to additional computational complexity.

Figure 3 evaluates the improvement in computational complexity offered by the three alternative scenarios discussed in our methodology (reduction in number of monitored regions, reduction of area coverage for these regions and a combination among them) regarding the *des* benchmark. As we have already mentioned, this task is employed (usually at run-time) in order to decrease the computational complexity for detail thermal analysis, with a penalty in accuracy. The horizontal axis in Figure 3 plots the desired reduction in number of execution cycles for thermal analysis, whereas the vertical one gives the maximum temperature error, as compared to a detailed thermal analysis with *Quick_Hotspot*.

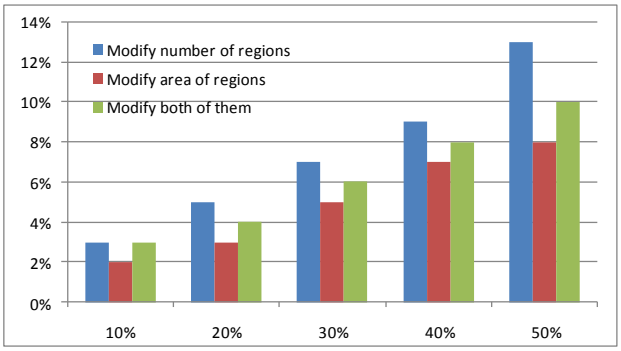

Figure 3 Maximum temperature error for different optimizations

Based on Figure 3, all the three scenarios result to similar error, while the optimal among them is the modification of area coverage (scenario 2). Note that, even for the case where error in temperature values is about 8%, this leads to a significant speedup execution since it results to reduction execution cycles about 50%.

4.3 Accuracy of Thermal Profile

The accuracy of the derived thermal analysis is another crucial issue in order to confirm the mathematical validity of the proposed methodology. For this purpose, we compute average and maximum temperature errors between the two software approaches.

More specifically, during our exploration we found that the maximum error regarding the temperature of a single block is up to 0.055 ºC, whereas the average error over the entire platform is about 3×10^{-5} ºC. Consequently, we can safely claim that the acceleration of the thermal analysis tool has been successfully accomplished with almost no effect on the tool's accuracy.

By maintaining the same level of accuracy with *Hotspot*, while the execution time is much less, our solution becomes a very powerful tool with a wider variety of applications. Typical examples, among others, are MPSoCs that utilize ad hoc power/thermal modeling and management tools.

Apart from this, we also study the efficiency of applying localized thermal analysis. This task involves finding out and clustering slices that operate under temperatures higher than a threshold value, similar to Figure 2. Since these slices exhibit higher probability of failure (due to increased temperature values), it is safe to claim that exclusive analysis of these slices will improve execution time without error in thermal stress.

The results of this study are summarized in Table 7, where threshold value was set equals to $T_{threshold} = \frac{T_{max}}{2}$. More specifically, the second column gives the total number of fabricated slices (hence the candidate number of power sources), whereas the values at third and fourth columns denote the number of slices that belong to critical for failure regions (without operating temperature higher than threshold value), as they derived with *Hotspot* and the proposed tool (*Quick_Hotspot*), respectively. Finally, the last column gives the error between these two values. Based on these results, we can claim that our methodology does not impose any accuracy degradation, apart from the significant elimination of computation complexity, since the average error in number of slices that clustered as critical with only one of the two algorithmic approaches is about 10^{-5}.

Even though the results provided in Table 7 gives estimation about the accuracy of proposed solution, we also evaluate how the temperature values varies among slices. More specifically, up to now we compare the number of slices where temperature exceeds a certain threshold between the two alternative ways (conventional and quick hotspot). However, there is a decent chance that two slices both have higher temperature than threshold but still differs to each other by a non-trivial percentage.

Table 7 Slices with temperatures higher than $T_{max}/2$.

Bench-mark	Number of critical for failure slices			Error (%)
	Total	***Hotspot***	***Quick_ Hotspot***	
alu4	1,600	510	512	-0.06%
apex2	2,025	1,276	1,275	0.05%
apex4	1,369	684	684	0.00%
diffeq	1,600	831	832	-0.06%
ex5p	3,025	1,241	1,240	0.00%
misex3	3,721	1,340	1,339	0.03%
s298	2,025	1,177	1,175	0.05%
seq	1,849	1,035	1,035	0.00%
tseng	1,156	437	427	0.00%
Average:	**2,041**	**948**	**948**	**0.00%**

Figure 4 gives the total number of slices that exhibit different temperature values between the two alternative thermal analysis approaches. Based on this analysis, the average error in terms of slices that operated under high temperature values with only one of the two algorithmic approaches is only 7. Note that this number corresponds to an average error (as compared to the total number of slices) about 0.004%. Furthermore, if we take into account the considerable reduction in computational complexity provided by our proposed solution, we can almost safely claim that this error is negligible.

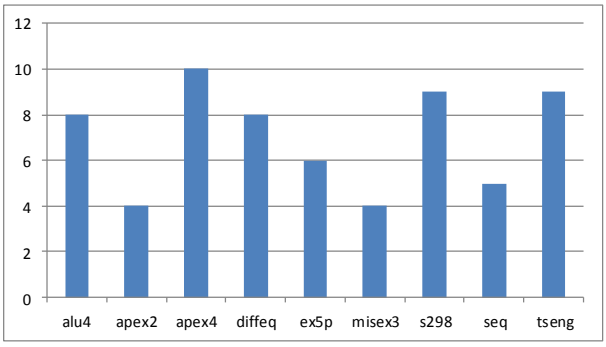

Figure 4 Error in number of slices classified as critical with only one of the two alternative algorithms.

In addition to that, we also evaluate the maximum temperature variation for the same slice between the two alternative thermal analysis tools. The results of this study are summarized in Figure 5. Based on this analysis we can conclude that the accuracy derived by our proposed solution is almost identical to the one

achieved by *HotSpot* 5.0, since the average maximum error in slices among the benchmarks is only 0.003ºC.

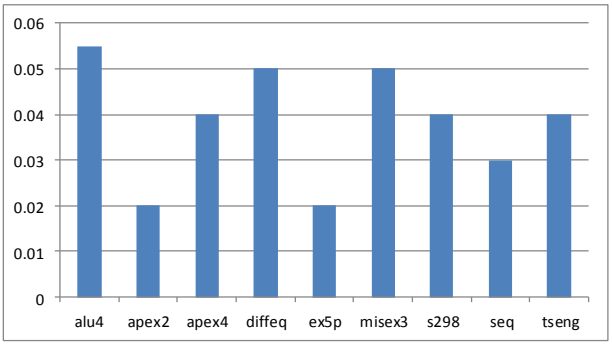

Figure 5 Maximum temperature difference for the same slices with the two alternative thermal analysis tools.

Finally, in order to show the additional improvement in execution time achieved by using a localized *Quick_Hotspot* execution, rather than conventional *Hotspot* (which performs thermal analysis in the whole chip), Figure 6 evaluates this gain in number of execution cycles. Note that this feature is especially crucial for thermal analysis at run-time, where computational resources are limited.

More specifically, this graph plots the improvement in number of execution cycles achieved by the localized thermal analysis. The values at this figure are normalized over the corresponding number of cycles required from original *Hotspot* [8].

Based on this analysis, we can conclude that if selective thermal analysis is applied both on *Hotspot* and *Quick_Hotspot* tools, it achieves an additional improvement in the number of cycles at our proposed solution about 43%, on average. Note that, this additional reduction is translated as 75% reduction in comparison to the original *Hotspot* tool (that models the entire architecture).

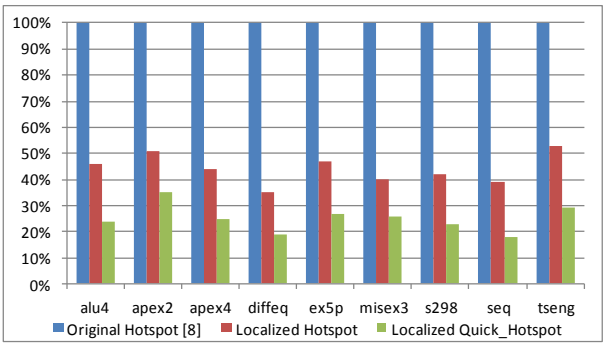

Figure 6 Speedup in execution time for the localized thermal analysis as compared to *Hotspot* 5.0.

5 Conclusions

A systematic methodology for performing run-time thermal analysis for MPSoC designs was presented. The methodology is supported by a new software tool, named *Quick_Hotspot*. The proposed methodology was applied to evaluate thermal analysis of alternative floor-plans. Experimental results show that our solution achieves significant improvement in computational complexity ranging from 41% (when applied as a stand-alone tool), up to 75% (when applied only to critical for failure resources), as compared to an existing state-of-the-art solution, without any accuracy degradation (average error about 3×10^{-5} ºC).

6 Literature

[1] T. Sakurai and A. Newton, "Alpha-power law mosfet model and its applications to cmos inverter delay and other formulas", IEEE JSSC, April 1990

[2] Quick_Hotspot tool, available at http://proteas.microlab.ntua.gr/ksiop/software.html

[3] S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0", Technical Report, Microelectronics Centre of North Carolina, 1991.

[4] International Technology Roadmap for Semiconductors, Assembly and Packaging, 2007 Edition

[5] K. Sankaranarayanan, et.al., "A Case for Thermal-Aware Floorplanning at the Microarchitectural Level", The Journal of Instruction-Level Parallelism, Sept. 2005

[6] A. Chandrakasan , R. Brodersen, "Low Power Digital CMOS Design", Kluwer Academic Publishers, Norwell, MA, 1995

[7] David Atienza, et.al., "HW-SW Emulation Framework for Temperature-Aware Design in MPSoCs", ACM Transactions on Design Automation for Embedded Systems, Vol.12, N.3, pp. 1 – 26, August 2007.

[8] K. Skadron, et.al., "Temperature-Aware Microarchitecture: Modeling and Implementation", ACM Trans. on Architecture and Code Optimization, 1(1):94-125, Mar. 2004

[9] Altera Stratix FPGA devices (available at http://www.altera.com)

[10] K. Skadron, et. al., "Temperature-aware microarchitecture: Extended discussion and results", Technical Report CS-2003-08, University of Virginia, Computer Science Department, 2003.

[11] Standard Cell Benchmark Circuits from the Microelectronics Center of North Calorina, http://vlsicad.cs.binghamton.edu/gz/PDWorkshop91.tgz

[12] W. Press, et.al., Numerical Recipes in C; The Art of Scientific Computing, 2nd ed. Cambridge: Cambridge University Press, 1997.

[13] K. Poon, S. Wilton, A. Yan, "A Detailed Power Model for Field-Programmable Gate Arrays", in ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 10, Issue 2, April 2005, pp. 279-302.