NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE

Development of Design Methodologies and CAD tools for System-level evaluation of interconnect reliability issues in SoC designs

DIPLOMA THESIS

Christos A. Papameletis

Supervisor: Dimitrios Soudris,
Assistant Professor

Athens, July 2010

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE

Development of Design Methodologies and CAD tools for System-level evaluation of interconnect reliability issues in SoC designs

DIPLOMA THESIS

Christos A. Papameletis

Supervisor: Dimitrios Soudris,
Assistant Professor

Approved by the tripartite committee on July 13, 2010.

....	....	....
Dimitrios Soudris	Kiamal Pekmestzi	George Economakos
Assistant Professor	Professor	Lecturer

Athens, July 2010

....
Christos A. Papameletis

Graduate of the N.T.U.A. School of Electrical and Computer Engineering

Abstract

The presented diploma thesis deals with interconnect reliability in VLSI systems from a system-level perspective. The dominant phenomena that are examined are Electro-migration (EM) and Time-dependent Dielectric Breakdown (TDDB). The main goal of this work was the creation of a design flow that estimates the system's lifetime (MTTF) because of timing failures caused by the gradual degradation of the electrical characteristics of interconnects. The presented flow is based on a pre-existing work that was developed at IMEC, Belgium. A main feature of the project is the use of actual temperature data for each individual region of the system, which are derived from application-specific simulations. This results in rather accurate lifetime estimations as both reliability-threatening phenomena examined are heavily dependent on temperature. Another improvement that increases the accuracy of the predictions is the estimation of the interconnets' current density through Spice simulations. Other important features are the automation of the design flow as a tool as well as its compatibility with state-of-the-art EDA tools, such as the Cadence SoC Encounter Layout & Timing analysis system and the Synopsys front-end suite.

Keywords: << SoC, Electro-migration, Time-dependent dielectric breakdown, reliability, EM, TDDB, soft failure, timing, HotSpot, SoC Encounter >>

Acknowledgements

I would like to acknowledge Prof. Dimitrios Soudris for assigning me this topic and supervising me in this work. Also, I would like to thank Prof. Kiamal Pekmestzi and Lect. George Economakos for their support in the Microcomputers and Digital Systems Laboratory.
I am particularly grateful to doctorate student Dimitrios Bekiaris for his guidance and the impeccable cooperation we had with each other. The completion of this work would not be possible without his help and technical support. I also am grateful to post-doctorate researcher Antonis Papanikolaou for providing the foundations for this work and for the fruitful discussion I had with him and Dimitrios Bekiaris which included valuable insightful remarks.
Finally, I deeply thank my family and friends for their long-term love and support. They all contributed indirectly to this work, either financially or psychologically.

Table of Contents

List of Figures

List of Tables

1

Introduction

1.1 The importance of reliability

The reliability of CMOS devices and interconnect structures has always been a great concern for their designers, as the desired lifetime of products should be guaranteed at the design and manufacturing phases, before entering the market. Reliable operation becomes even more important when human lives depend on the system.

This is also the case for VLSI systems used in embedded devices, as several of their applications include potentially life-critical situations, such as in cars or airplanes, as well as in medical devices. But even if no lives depend on them, the products' reliable operation for the desired lifetime and under the desired operational specifications is critical for both the producer and the consumer.

However, the trend of CMOS process scaling, which is likely to be continued towards sub-90nm technology nodes, reveals novel reliability phenomena, which affect the system's functionality either abruptly or progressively, leading to functional or parametric failures respectively. In this thesis, we focus on parametric or “soft” failures of two dominant interconnect reliability phenomena, namely of Electro-migration and Time-Dependent Dielectric Breakdown, in a sub-micron technology context. However, the developed design flow presented in this work could estimate the impact of other phenomena with progressive impact on a system's timing behavior, presented in the following section, as well.

1.2 Reliability-threatening phenomena in general

The most common and dominant reliability-threatening phenomena are the following:

- Electro-migration (EM): EM is the transport of material caused by the gradual movement of the ions in a conductor due to the momentum transfer between conducting electrons and diffusing metal atoms. EM may progressively, and depending on the system's operation, lead to the destruction of the wire, as mass transfer caused by electrons eventually causes discontinuities in the wires by transporting also atoms of copper from the anode to the cathode, forming large voids at one end (anode) and extrusions at the other end (cathode). The underlying effect acts like a wear-out mechanism that affects the interconnects of digital and analog ICs and its impact gains in significance with CMOS technology scaling.
- Time-Dependent Dielectric Breakdown (TDDB): TDDB is a progressive failure mechanism in device and interconnect structures where a capacitor is formed, leading to the gradual breakdown of the dielectric material as a result of long-time application of relatively low electric field (in contradiction to immediate breakdown, which is caused by strong electric field). The breakdown is caused by the gradual formation of conducting paths through the dielectric material eventually resulting in a short-circuit.
- Hot Carrier Injection (HCI): HCI is the phenomenon in semiconductor electronic devices, where either an electron or a hole gains sufficient kinetic energy to overcome a potential barrier between different areas of the device and migrates from one area to another. The kinetic energy of microscopic particles is directly related to the temperature of the matter they constitute, so the higher the temperature, the higher the kinetic energy of the particles, hence the word hot. The injection of these high-energy carriers damages the dielectric material, gradually increasing its chance of failure.
- Negative Bias Temperature Instability (NBTI): NBTI is a reliability issue of immediate concern in pMOS devices, stressed with negative gate voltages. NBTI manifests as an increase in the threshold voltage and consequent decrease in the drain current and transconductance, which can render a transistor useless.

From a device-component view, reliability issues in VLSI circuits can be divided in two categories, namely device-related and interconnect-related. As the size and complexity of ICs increase, the number of wires for the interconnection of the transistors in an IC grows very rapidly. So, the high ratio of interconnects in conjunction with newer fabrication technologies, shifts the attention greatly towards interconnect reliability. Newer fabrication technologies shrink the dimensions of ICs in order to make them faster and cheaper, while they also introduce new materials, such as low-κ dielectrics, that make systems faster by reducing parasitic capacitances. However, these porous materials feature poorer electrical characteristics and are more vulnerable, thus making systems less reliable.

1.3 Motivation for studying interconnect reliability

Reliability-aware design enables the location of vulnerable areas and components of the system and their re-design by certain criteria and techniques until the required specifications are met. As a result, ICs can function correctly for longer times and remain as unaffected as possible by random failures or fabrication defects. For most applications, the minimum target lifetime for individual transistors and interconnects is 10 years of continuous activity.

However, standards are not present solely for reliability, but also for yield, fabrication cost, area, power, performance and other parameters. Usually, these factors are connected to each other and influence each other negatively. This makes the tradeoff decisions very crucial as achieving the desired delicate balance is very hard. For instance, a higher performance standard for the system could lead to the up-scaling of supply voltage, which in turn leads to increased currents, temperature and other parameters which shorten the IC's lifetime. Likewise, if a lower performance standard is set, the lifetime of the IC will rise, leading to a more reliable system.

Nevertheless, there is another reason because of which lower performance standards are likely to lead to increased reliability. This reason will be illustrated by an example. Newer fabrication technologies result in a transition from abrupt, hard failure mechanisms to gradual, soft failure mechanisms. Suppose there is an interconnect where EM starts to affect its internal structure by forming voids. At some point, a void in the wire is created because of the metal atom transfer. Electrical current still flows, but there is a significant rise in the wire's electrical resistance, resulting in increased delay for the interconnect.

The same principle can be applied to an interconnect where TDDB occurs. Before the dielectric leakage current flowing through the conductive path becomes large enough to cause a short circuit, the transition time of the interconnect rises because of it. Now, this rise in delay might cause a tightly timed, low slack system to stop functioning correctly because the delay in some register-to-register paths might exceed the clock period. If the system's clock and in general the design's timing constraints allowed for a higher slack, this delay increment might not influence the system, unless a hard failure occurs due to high peak currents leading to hotspots or disrupting the structure of certain, vulnerable wires due to metal atom transfer because of extremely high current density.

1.4 Choice and description of the examined phenomena

In this work and tool flow only two reliability-threatening phenomena are examined, EM and TDDB as they have application in interconnect reliability and according to recent research these phenomena seem to dominate the reliability of newer technologies beyond the 90nm node. Actually, EM starts to gain in significance as manufacturing technology dimensions shrink – especially past the submicron node – as this raises the current density that greatly influences the intensity of the phenomenon. To be more precise, the scaling of a circuit's dimensions by a factor k, increases the power density proportionally to k and the current density increases by k^2. TDDB also becomes increasingly important because of the reduction of the inter-layer distance between adjacent wires, due to CMOS technology scaling, which comes together with the introduction of low-κ dielectric materials, leading to a rise in the inter-metal electric field and accelerating the breakdown process, as the porous nature of low-κ dielectrics leads to the easier establishment of conducting paths, thus making them more susceptible to TDDB.

1.5 The goals and main idea of the presented flow

The main goal of the presented work has been the incorporation of a realistic temperature profile in the calculation of the reliability models' parameters. This is important, because both phenomena presented above exhibit exponential dependence on temperature as the models presented in the next chapter show. Therefore, the use of imprecise temperature data can lead to either too optimistic or too pessimistic predictions regarding the lifetime of a system. In order to overcome this problem, a solution is proposed, involving the estimation of the temperature of each individual hierarchical unit on an architecture level, depending on its application-specific power consumption and its floorplan. This also introduces a further criterion for reliability-critical region location, namely that of high temperature, because of its exponential effect on reliability phenomena.

But still, the accurate estimation of temperature and consequently of the lifetime and degradation parameters of interconnects would not be of much significance, if there was no way to project the impact of these results on the system's timing. A methodology to interpret the quantitative effect of resistance rise in wires due to EM or leakage current rise in interconnects due to TDDB on system timing allows the exploitation of the results of predictive models in a tool, to assist reliability-aware design of systems. In other words, a methodology is presented in this work that allows the estimation of the degradation of a system's performance due to the gradual degradation of the electrical characteristics of its interconnects, caused by various reliability-threatening phenomena. Such a transition is important and has a lot of applications, some of which are mentioned below:

- It enhances the understanding of the designer as far as reliability is concerned, providing him with more information regarding the gradual impact of time on his design, thus enabling him to make a more effective tradeoff between performance and reliability.
- Through a set of incremental simulations over increasing time periods of circuit operation, the gradual reduction of the system's timing slack can be depicted in a diagram, giving a more detailed picture of the evolution of the phenomena, as the operating time progresses.

The concept of moving from the electrical and process characteristics' degradation of wires to the estimation of the system's performance drift over time due to the progressive impact of such degradation on the design's timing is explained graphically in Figure 1.2, which is the timing diagram of the register-to-register paths of Figure 1.1.

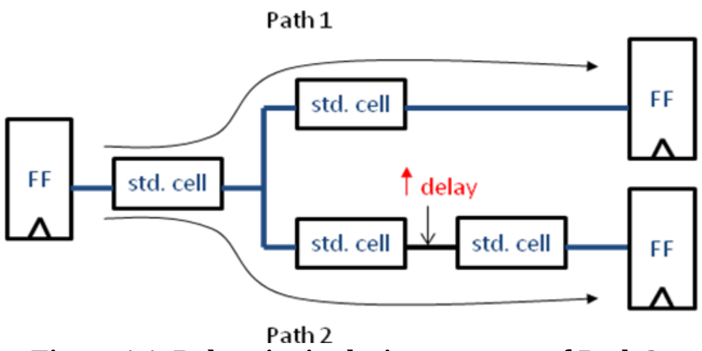


Figure 1.1. Delay rise in the interconnect of Path 2.

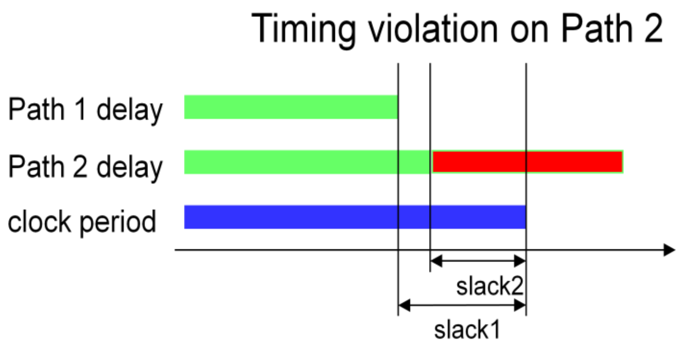


Figure 1.2. Impact of delay rise on system's timing.

If, as stated above, the system was loosely timed and left a larger slack, the delay rise in path 2 in the above example would not lead to the violation of the set timing constraint (clock period), as the output data of Path 2 would be produced before the beginning of the next clock cycle. Hence, the delay tolerance of a design due to the timing impact of such reliability phenomena is strongly dependent on its performance requirements.

The transition described above is realized through an automated tool flow. This is based on state-of-the-art EDA tools such as the Cadence SoC Encounter physical implementation tool suite [22] and the Encounter Static Timing Analysis (Encounter Timing System-ETS) [19] engine. Standardized file formats such as the SPEF and the SDF are used as carriers of the effects of the reliability phenomena examined, as they include information that are important for the estimation of the system's timing. Therefore, either the SPEF or the SDF file of the target design should be read before any timing analysis, through which the design's performance is estimated.

Hence, the computations of the EM model regarding the resistance rise of the affected wires should be incorporated into the design's parasitic information file, so that the timing impact of the underlying wear-out is evaluated. In order to assess the impact of EM, the current density in each wire is first estimated. Next, the wires that are affected by EM are identified and using a model, their expected resistance rise over time is calculated. Finally, this resistance rise is incorporated in the design's SPEF file and the corresponding SDF file is also generated, based on the SPEF's parasitics, which are used for the required delay computations.

On the other hand, in order to assess the impact of TDDB, adjacent wires are located and the leakage current through the dielectric is estimated by performing extrapolation from stress to operating conditions, based on look-up table libraries of inter-metal leakage. The corresponding delay increment because of this leakage is computed by linear interpolation between the wire patterns of a constructed look-up table library and those extracted from the layout. The aforementioned library is constructed in order to estimate the delay increment of wires due to TDDB. Then, the increased delay of charging the wires of the examined interconnects is computed for each interconnect and it is incorporated into the design's SDF file. It should be noted that, the TDDB requires a different approach of annotating the induced delay compared to the EM, as there is no certain electrical characteristic directly affected and. So, the conducting paths through the dielectric are emulated by progressively increasing the delay of charging wires affected by TDDB, after a certain number of operating years. Although the most accurate method for the computation of the inter-metal dielectric leakage is through Spice simulations using distributed RC models, this is a time-consuming solution. In order to overcome this obstacle, a Spice-based wire characterization library including the delay of adjacent wires due to TDDB is constructed instead and look-ups combined with interpolations are used to estimate the delay rise of the layout's real wire patterns. The computed delay increment for each wire of the examined net that belongs to a certain register-to-register path is annotated to the design's SDF file.

The comparison of the timing analysis reports based on the initial SDF file and each one of the other two SDFs, namely after the EM and TDDB models' computation and annotation, will reveal the impact of each phenomenon on the system's timing, considering a certain number of operating years. Also, assuming that the two phenomena are independent, if the delay overhead due to TDDB is annotated on the SDF file produced after the annotation of the EM impact, the resulting SDF file will reflect the combined impact of both phenomena. In order to make the model predictions much more accurate, an application-based power and then temperature estimation is performed, before the core of the interconnect reliability flow is executed.

The encapsulation of the design's temperature distribution across the layout comprises the main contribution of this work, as previous interconnect reliability frameworks [1][2] do not consider realistic temperature traces. However, apart from the temperature profiling, significant additions, corrections and improvements have been made to the initial framework of [1][2] during this thesis. These novel features, introduced by this work, are summarized in Table 1.1, shown below.

Table 1.1. Feature comparison table between past and proposed version of the flow.

Feature	Proposed version	Past version [1][2]
Temperature-aware lifetime estimation	✓	✗
Accurate current density calculation (through Spice)	✓	✗
Accurate transition time computation (through EDA tool)	✓	✗
Accurate look-up of TDDB-induced delay	✓	✗
Simultaneous analysis of multiple paths	✓	✗
Automated tool	✓	✗
Combined EM&TDDB timing impact estimation	✓	✗

1.6 The test platforms

The reliability tool flow has been tested using several designs, which are the testbenches, from which the wires of nets in the examined register-to-register timing paths are extracted. In order to thoroughly test our approach, we used several place&route scenarios, so that their impact on the evolution of the underlying phenomena can be explored. These scenarios were applied to an embedded Multi-processor System-on-Chip platform based on two LEON3 SPARC processors. The aforementioned design, implemented based on an Application-Specific Integrated Circuit (ASIC) flow with a 45nm standard-cell library, is large and complex enough and the implementing technology used is state-of-the-art, so that the impact of the examined interconnect wear-out mechanisms is significant enough. Especially the LEON3 design features buses with long wires, as well as computational-intensive ALUs, which may include timing paths with strict timing constraints and also long nets with wires susceptible to EM.

1.7 Chapter outline

An overview of the chapters that follow is presented in this section, which can act as a guide to the reader. In Chapter 2, the examined reliability-threatening phenomena, as well as their models, are described. Chapter 3 provides a generic, high-level description of the presented reliability analysis flow in the form of steps, so that the reader can develop a general understanding of the origin of the data required at each step in order to calculate the desired parameters. The 4th Chapter elaborates further on the flow by getting a step closer to its implementation, describing the technical details of the individual scripts that comprise it. The physical designs (layouts) that were used as testbenches, together with the tools that generated them and their configuration details, are described in Chapter 5. In the first sections of Chapter 6 the produced experimental results and the conclusions derived from them are presented, whereas the next sections deal with the potential future extensions regarding the reliability analysis framework and attempt to provide solutions that could potentially mitigate the impact of the examined wear-out mechanisms. The 7th Chapter is the Appendix, where the code of the various developed scripts is listed, so that it is possible to delve deeper into the flow’s implementation.

2

Reliability-threatening phenomena & models

2.1 Electro-migration

Electro-migration (EM) is a physical phenomenon that may lead to major reliability problems regarding the structure and the electrical characteristics of copper-based interconnects, which are used in most modern VLSI systems. However, EM in general occurs when electrical current flows through a metal conductor and causes some metal atoms to drift along the wire due to momentum transfer from the electrons (electron wind). The transport of atoms progressively leads to the formation of voids, specifically near one end of the wire (anode). Because of the gradual void formation, the internal structure of the affected wire is not coherent any longer, as areas of void intercept those of copper. As the current cannot pass through voids, it has to come along the copper wire's barrier, which causes a significant increase in the time required for the current to pass through. Consequently, the progressive formation of voids due to EM leads to a gradual resistance rise of the wire, which in turn inserts a delay overhead to the signal's propagation. In recent years, research towards EM was focused mainly on the lines of the power and ground network, which may suffer from high voltage drop when the underlying phenomenon starts to develop.

Regarding data signals, EM has started to gain in significance as the geometric dimensions of interconnects tend to shrink, following the trend of CMOS technology scaling down to the nanometer regime. Technology scaling reduces the width and thickness of metal interconnects and consequently their cross-section area, as well as the wire pitch and spacing, while the operating voltage saturates around 1V in state-of-the-art deep sub-micron CMOS technologies. Therefore, the stressing of wires becomes more intense, as the current density tends to be increasing with scaling. Precisely, the scaling of a circuit's dimensions by a factor *k*, increases the power density proportionally to *k* and the current density increases by k^2. The dominant mechanisms that cause a circuit to malfunction due to the momentum metal atoms drift, caused by EM, are the following:

- The formation of voids along interconnects because of the displacement of material from some spots, leading to a resistance rise.
- The formation of extrusions near the anode because of material accumulation there, leading to higher risk of adjacent wire shorting.

The formation of a void usually starts from a spot of the wire, where some sort of defect in the crystal grid exists. The later results in higher collision rate between electrons and metal atoms and eventually to higher momentum transfer between them. At some point atoms detach from the grid and move. Then, because of the electrical forces that appear along the void in conjunction with the weakening of that spot's structure, the void begins to expand. Gradually, the void tends to occupy larger portions of the wire's width and height, reducing the conductor's cross-section at that spot.

Such a phenomenon is located along a limited length of the wire, so the effect on its resistance is negligible. But at some time, the void fully occupies the cross-section of the wire. At that spot, the current is forced to flow through the diffusion barrier, which has much higher resistance than the metal conductor, due to its low conductivity and cross section. As a result, the wire resistance abruptly rises by several hundred Ohms (resistance step). Since then, strong electrical forces appear between the ends of the void which act as an anode and cathode respectively. Consequently, the void expands further along the length of the wire increasing the wire's resistance at an approximately linear rate (resistance slope).

In the following graph, the stages of the resistance step and slope as EM progresses are demonstrated clearly:

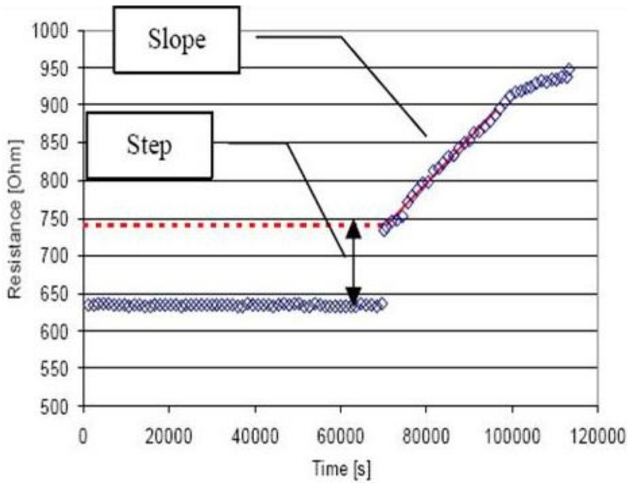


Figure 2.1. Wire resistance change over time due to EM.

While a void forms at some spot in the wire, detached metal atoms are carried away by electrical forces and accumulate near the anode, forming extrusions through their diffusion in the dielectric material that surrounds the wire. If an adjacent wire exists in a close distance to the first one, it is possible that a short circuit between these two wires is created. Even if the material transferred is not adequate to reach the adjacent wire, the decreased distance enhances the electric field locally, thus making the dielectric material breakdown easier. The following electronic microscope photos illustrate EM- induced voids and extrusions, respectively.

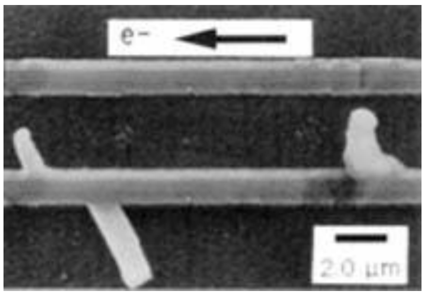


Figure 2.2. EM-induced void above a via.

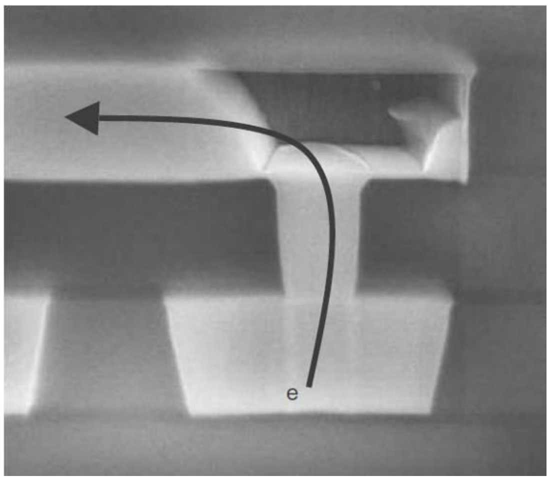


Figure 2.3. EM-induced extrusion near an adjacent wire.

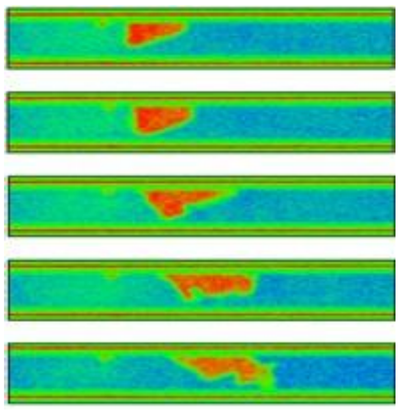


Figure 2.4. Progressive void formation in a wire over time, due to EM.

Because of the thermal nature of the process of EM, temperature is a critical parameter for the underlying wear-out's development, influencing the evolution of the phenomenon exponentially, by both accelerating it and amplifying its results, as it will be shown in the mathematical formulation given below. The positive feedback loop observed in the figure below, demonstrates how the temperature dependence acts on the void's nucleation and expansion, consequently increasing the wire's resistance.

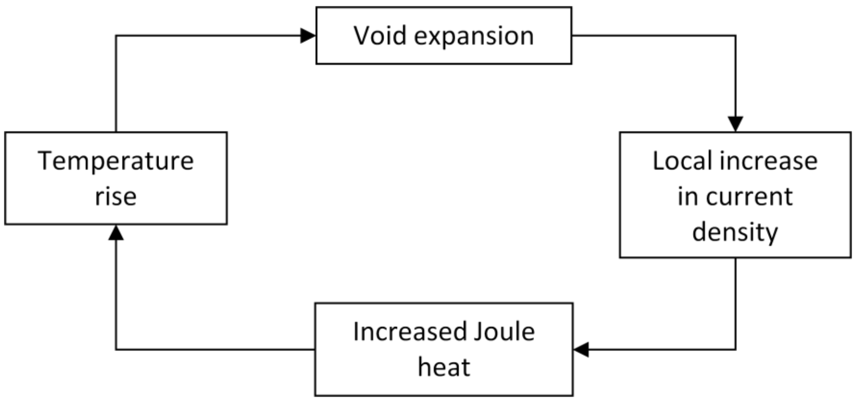


Figure 2.5. Positive-feedback EM acceleration phenomenon.

However, EM only occurs in wires whose length exceeds the Blech length, for a given value of current density. This sets the lower limit of length and density, below which a mechanical stress buildup causes a reversed migration process, which reduces or even compensates the effective material flow towards the anode. The upper limit of the critical product of current density and wire length has been determined to be 3700 A/cm, by conducting extensive experiments. For a given density of current flowing though a wire, the Blech length can then be easily calculated by performing a single division. In conclusion, only wires which are longer than $\frac{jL_{critical}}{j_{normal}}$ are affected by EM.

Additionally, wires conducting DC current are more susceptible to EM than those conducting AC current. In the case of the later, the bidirectional electron flow is considered to have healing effects, counterbalancing the mass transfer caused by EM. Due to the lack of experimental data and therefore of a model for AC current driven

EM, the equivalent DC current is estimated as the absolute difference of rise and fall current and a DC current driven EM model is used to calculate the results of EM.

Due to the relatively high lifetime of interconnects despite the influence of EM, it is impractical to characterize EM under operating conditions as this would require year-long experiments. A semi-empirical mathematical equation, Black's equation, is commonly used instead to predict the lifetime of interconnects in VLSI circuits, based on experiments conducted under stress conditions of high temperature and voltage. The model's results can then be extrapolated to operating conditions to estimate the expected lifetime as common practice suggests. This method is widely accepted as it is known to provide accurate predictions.

Black's equation estimates the mean time to failure (MTTF) of a wire, that is the time until the resistance step rise occurs, using experimental data as:

$$t_{50,stress} = \frac{A}{J^n} e^{\frac{E_a}{kT}} \quad (2.1)$$

, where A is a constant, n is the current density exponent factor (ranging between 1 and 2, depending on the technology of implementation), E_a is the activation energy in Joule, depending on the metal of the conductor, J is the wire's current density, k is the Boltzmann constant and T is the temperature in Kelvin. The first three of the aforementioned parameters are derived from experimental data. The following graph displays Black's equation for a given set of these parameters:

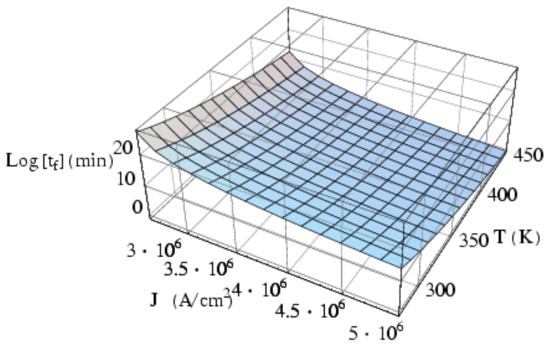


Figure 2.6. Graphical representation of Black's equation.

In order to extrapolate the lifetime to operating conditions the following statistical equation is used:

$$t_{50,normal} = t_{50,stress} \cdot \left(\frac{J_{stress}}{J_{normal}}\right)^n e^{\frac{E_a}{k} \cdot \left(\frac{1}{T_{normal}} - \frac{1}{T_{stress}}\right)} \quad (2.2)$$

, where the indexes stress and normal are used to denote the conditions each variable refers to. Another equation is then used to estimate the wire's resistance rise rate after t_{50}, as:

$$R_{slope,stress} = A \cdot J \cdot e^{-\frac{E_a}{kT}} \quad (2.3)$$

, where all symbols have the same meaning as before. Using the same statistical method to extrapolate to operating conditions, the following equation is obtained:

$$R_{slope,normal} = R_{slope,stress} \cdot \frac{J_{normal}}{J_{stress}} \cdot e^{\frac{E_a}{k}\cdot\left(\frac{1}{T_{stress}}-\frac{1}{T_{normal}}\right)} \quad (2.4)$$

Finally, the resistance step, ΔR, is calculated as follows:

$$\Delta R = \left(\frac{\rho_b}{t_b \cdot (W + \cdot 2H)} - \left(\frac{\frac{\rho_m}{H \cdot W} \cdot \frac{\rho_b}{t_b \cdot (W + \cdot 2H)}}{\frac{\rho_m}{H \cdot W} + \frac{\rho_b}{t_b \cdot (W + \cdot 2H)}} \right) \right) \cdot L_{void} \quad (2.5)$$

, where ρ_b is the resistivity of the diffusion barrier, t_b is its thickness, W is the wire width, H is its thickness, ρ_m is the resistivity of the metal and L_{void} is the void's length, formed in the timing window at which the EM impact is examined.
The above formula essentially calculates the difference of resistance when the current exclusively flows through the metal and when it is forced to flow through the diffusion barrier for the length of the void. The following figure explains how the resistance is obtained in each case by breaking the wire down to its metal, barrier and void and combining their individual resistances which are calculated as the quotient of the product of resistivity and length, divided by the cross-section.

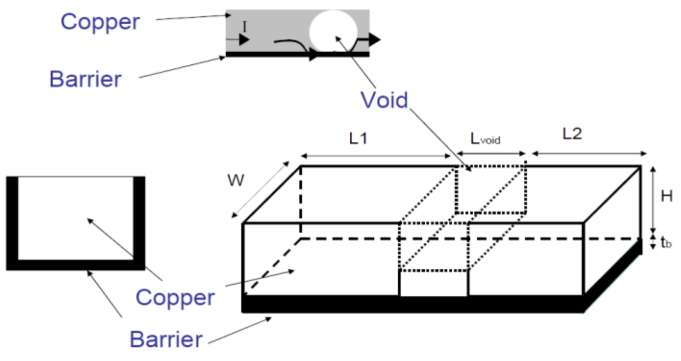


Figure 2.7. Void demonstration in a copper wire.

2.2 *Time-dependent Dielectric Breakdown*

The breakdown of the dielectric material refers to the destruction of the dielectric layer that insulates adjacent conductive areas from each other, preventing an unwanted short circuit. Such insulating layers are found in various parts of an IC, such as the gates of MOSFETs, between the plates of capacitors and between adjacent wires of the same metal layer. This last case, which is highly related to the Time-Dependent Dielectric Breakdown (TDDB), namely the second interconnect wear-out we are focusing on in this thesis, has been gaining in significance lately, due to the scaling of interconnect dimensions, as the CMOS process technologies reach the deep-deep submicron era. This is mainly for two reasons:

- Low-*k* dielectric materials are used in order to decrease the parasitic capacitances of various circuit components and consequently improve both its performance and its power consumption. However, these materials have poorer electrical characteristics and break down sooner and under weaker electric fields.
- The electric fields between adjacent wires are amplified as their distance - and therefore the thickness of the dielectric material - is reduced. As a result, electric fields between adjacent wires are nowadays approaching those encountered in the gates of transistors one or two decades ago.

The Time-dependent Dielectric Breakdown (TDDB) occurs due to the gradual wear-out of all dielectric materials over time. Even those of high quality are threatened by TDDB over time. But this process of wear is accelerated by the presence of defects and imperfections in the dielectric material, as it will be explained below. The wear-out mechanism is divided in two stages in general:

During the first stage of build-up, charges (holes) are trapped in weak spots of the dielectric where defects or imperfections exist, as leakage current flows through it. These rise in number over time, leading to the formation of high electric fields and high leakage current regions along the wire. This process continues for quite a long time (under normal conditions year-long), until a critical concentration of trapped charges is reached, which is when a transition to the next stage of runaway occurs.

At the stage of runaway, the electric field, which is enhanced by the charge injection, exceeds the breakdown threshold in the weak spots of the dielectric material. Strong leakage currents flowing through those spots heat up the dielectric, which in turn leads to further increase of the current flow. This positive feedback loop eventually results in electrical and thermal runaway, eventually destroying the dielectric. The runaway stage happens in a very short period of time. A region where a dielectric breakdown has occurred, resulting in high leakage current and possibly to a hotspot in the specific die location can be seen in the following figure.

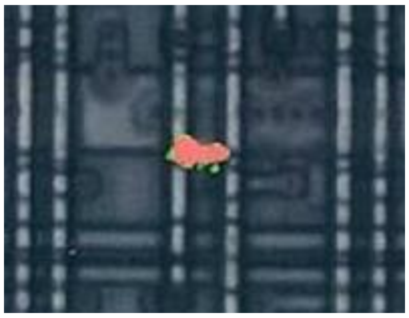


Figure 2.8. TDDB-induced leakage between adjacent wires of the same metal layer.

The presence of defects and imperfections in low-quality porous dielectrics greatly reduces the time needed for transition from the build-up to the runaway stage. These defects actually have the effect of "thinning" down the dielectric where they are located, since they are occupying space that should be occupied by the dielectric. The effective electric field is higher in these thinned-out areas compared to defect-free areas for any given voltage. This is why it takes a lower voltage and shorter time to break down the dielectric at its defect points.

The TDDB leads to gradual increase of the leakage current until the dielectric breaks down as explained above. This leakage current increase can be divided in three stages that are demonstrated in the following diagram.

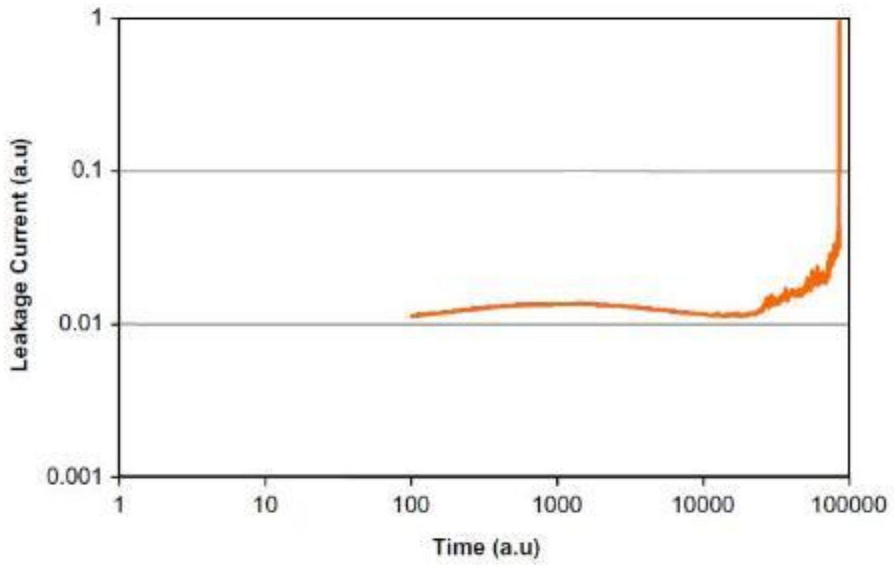


Figure 2.9. Inter-metal leakage current versus time.

During the first stage, the leakage current increases because of the accumulation of trapped charges. During the second stage, it decreases at a rate that depends on the quality of the dielectric material as new electron trapping spots are created. Finally, during the third stage, the leakage current increases at a logarithmic rate before the final breakdown that leads to short circuit. At that point, a conducting path connecting the anode and the cathode has formed. The evolution of such a path is shown in the following figure, which explains the idea behind the statistical percolation model that is used to estimate various parameters of the phenomenon of TDDB:

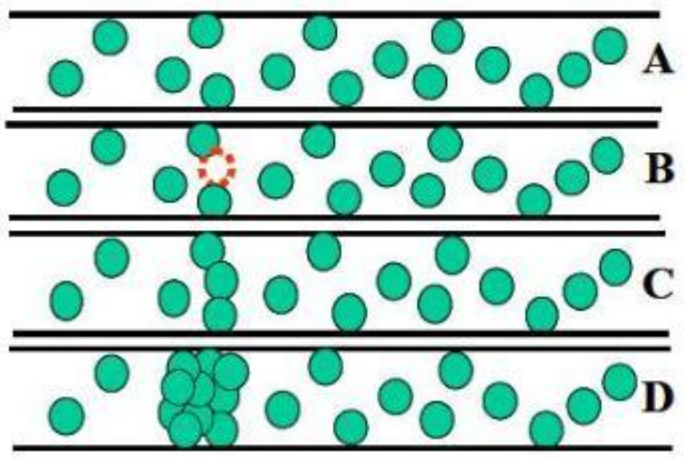


Figure 2.10. TDDB evolution stages.

The circles represent charge-trapping defects. The creation of such defects begins from the sparse trapping of positive charges (holes) that don’t form any conducting path which is a temporary situation (A). At some point in time, a path consisting of both permanent (green circles) and temporary (orange circle) defects may be formed (B). A defect that has trapped a hole can then either return to its initial state by losing the charge (A) or become permanent with the “connection” of the hole with an electron (C). When the leakage current rises significantly, the path (C) can be expanded (D) in combination with the elevated temperature, resulting in the breakdown of the dielectric material.

Under operating conditions encountered in a typical IC, TDDB takes a very long time to fully develop (possibly even decades), so the indicated method of study is through accelerated testing under stress conditions of high temperature and high voltage. The data collected from such experiments can stretch over a period of several days or up to a month. These are then extrapolated using empirical and statistical models to the desired time period and normal conditions.

The most widely used models are two. The first and more well-known is the E-model, which is based on the electric field in order to interpret the phenomenon [11]. The second, namely the 1/E-model, assumes that the dielectric breakdown process is driven by the leakage current [11]. In more detail:

- In the electrochemical E-model, the cause of low-field (<10 MV/cm) high temperature TDDB is due to field-enhanced thermal bond-breakage. In this model, the field serves to stretch molecular bonds thus making them weaker and more susceptible to breakage by standard Boltzmann (thermal) processes. Since the field reduces the activation energy required to break a bond, the degradation rate is expected to increase exponentially with field. Failure occurs when a localized density of broken bonds (or percolation sites) becomes sufficiently high to cause a conductive path to form from anode to cathode.
- The 1/E-model for TDDB (even at low fields) postulates that TDDB is due to current flow through the dielectric due to Fowler-Nordheim (F-N) conduction. Electrons, which are F-N injected from the cathode, may cause damage to the dielectric due to impact ionization as the electrons are accelerated through the dielectric. Also, when these accelerated electrons finally reach the anode, hot holes may be produced which can tunnel back into the dielectric causing damage (hot-hole anode-injection model). Since both the electrons from the cathode and the hot-holes from the anode are the result of F-N conduction, then the MTTF is expected to show an exponential dependence on the reciprocal of the electric field, 1/E.

Despite all the research that has been and is still being conducted on TDDB, there is no definitive consensus on the physical mechanism underlying the phenomenon. So both models are used depending on how well they fit the experimental data obtained using different dielectric materials, electric fields and temperatures.

In the case of interconnects, however, where the dielectric materials are still a few micrometers thick, their dielectric constant is low and the electric field is moderate in intensity, the E-model seems to provide a closer fit to experimental data and therefore it is preferred. The extrapolation proposed in this case is exponentially proportional to the electric field and results in an almost linear rise in the leakage current until the dielectric breakdown. Once again, the temperature plays a very important role in the evolution of TDDB.

3

Design flow methodology

3.1 Basic concepts

The presented design flow extracts the target interconnects from the design in order to estimate the system's lifetime due to timing violations, caused possibly by the gradual degradation of the electrical characteristics of interconnects. As target interconnects, we define those belonging to register-to-register timing paths, the timing of which is evaluated before and after the studied wear-out mechanisms' impact annotation. An important feature is the flow's expandability to other reliability-threatening phenomena besides EM and TDDB, by just incorporating the appropriate model calculations, following the existing generic flow's steps. These steps include, first of all, the retrieval of layout-specific interconnect data, required for the model's computations, and the temperature profile estimation for the given IC. The temperature is calculated for each unit of the design's floorplan, while the wire information for the target interconnects is derived from the Cadence SoC Encounter Database Access (DBAccess) command set [21].

As a result, the presented generic flow can be adopted to estimate the impact of any relevant interconnect wear-out mechanism that may progressively lead to the system's parametric (e.g. timing) failure over time. However, in this thesis, we have focused on EM and TDDB, not only because of the expected impact of the aforementioned phenomena on the interconnect delay (see Section 1.4), but also because of the presence of a pre-existing estimation framework [1][2], which evaluates the system's performance degradation due to these phenomena. Therefore, our intention was to make a step towards the improvement of the initial interconnect reliability toolflow, by considering a more realistic temperature distribution across the target design's layout in order to increase the accuracy of its predictions.

Regarding the flow's steps for EM, the proposed methodology, which has been strongly based on the initial framework, extracts the wires from each interconnect that belongs to the selected timing paths and updates its resistance, only if the examined wire is longer than the Blech length. Then, the updated wire resistances are annotated to the SPEF file of the design, which is input, together with the post-layout netlist, to the static timing analyzer, in order to evaluate the impact of EM on the design's performance.

A similar approach is followed for TDDB, as far as the wire extraction and the final performance evaluation are concerned. Regarding the core of the model, due to the lack of consensus on a formula for Inter-Metal Dielectric leakage current, the proposed flow, as well as its initial version [1][2], rely on the extrapolation of leakage current measurements from stress to operating conditions. The delay overhead introduced to the affected wires is computed in two steps. First of all, a look-up table library, relating the leakage current of several wire patterns to the corresponding delay overhead, is constructed. This library includes wire patterns covering a wide range of length, spacing and leakage current values, interpolation between which, is performed to estimate the actual delay overhead of each wire. The total delay for a specific interconnect is calculated as the weighted mean of its wires' delays and it is annotated to the SDF file, which is used as input to the static timing analyzer in order to evaluate the impact of the studied phenomenon on the design's performance.

The incorporation of an accurate temperature profile comprises the main contribution of the presented thesis, as it improves the accuracy of the timing drift evaluation. However, there are several other features, which differentiate the proposed work from the previous analysis framework [1][2], by contributing to the accurate estimation of current density for EM through Spice simulations, as well as to the wires' delay computation for TDDB through improved interpolations. Moreover, our toolflow is not limited to the analysis of the interconnects of the single most timing-critical path, as experimental results have shown that initially sub-critical paths may become critical after the annotation of the wear-out mechanisms' additional delay [3]. Hence, the new version of the framework is able to analyze as many paths as the designer selects concurrently, and to annotate and evaluate the total impact of the examined phenomena on the interconnects of the examined paths. An overview of the aforementioned novel features is illustrated in Figure 3.1, where we provide a holistic view of the proposed reliability framework, by depicting the individual steps required to evaluate the impact of EM and TDDB on a design's timing.

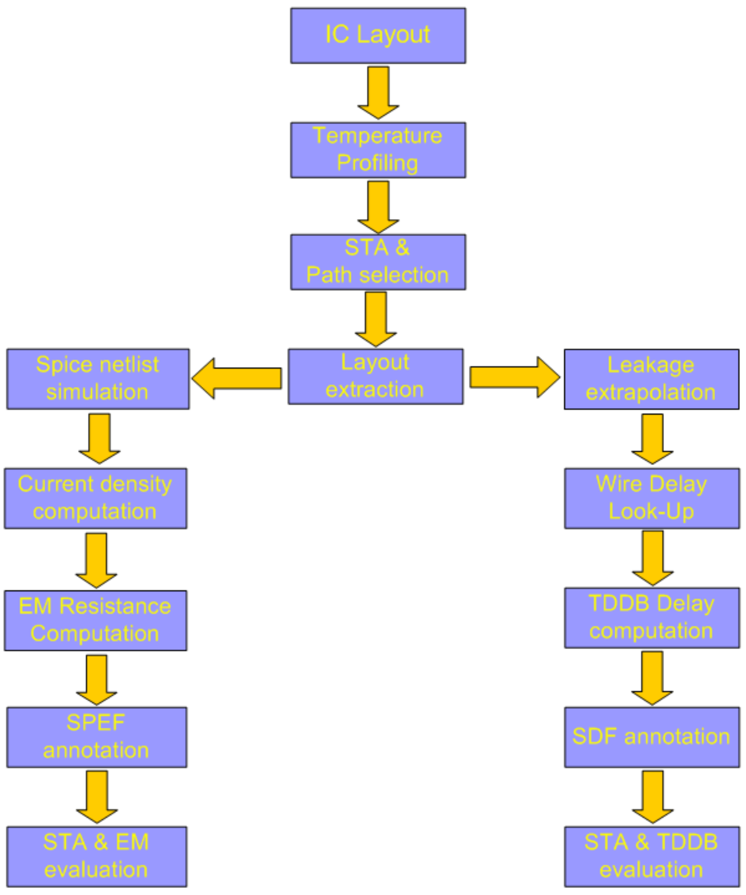


Figure 3.1. The proposed interconnect reliability framework for EM and TDDB.

The comparison of Figures 3.1 and 3.2 clearly reveals several improvements. First of all, the current density estimation performed in the initial flow of Figure 3.2 is quite approximate, as Spice simulations are not involved at all and the tree structure of the interconnect is ignored. Namely, the current of every wire is assumed to be equal to the total current flowing through the net it belongs to. However, this approach, greatly overestimates the current density, which leads to unrealistic results regarding the intensity of EM. Additionally, the boxes of multiple path selection and temperature estimation are missing, whereas in our flow, they comprise two of the main introduced features. Especially the temperature estimation is the most significant feature not included in the early stage of this estimation framework, which has been mainly focused on capturing the system's performance drift, rather than guaranteeing the accuracy of the results in such a detailed granularity. Moreover, the calculation of the TDDB-induced delay overhead used to be based on a rather "poor", layout-unaware, delay look-up library, in contradiction to our approach, which takes the characteristics of the examined layout into account. Nevertheless, the initial framework introduced an innovative approach into the field of reliability-aware design, as it was the first work that attempted to link the degradation of the electrical characteristics of interconnects to the design's performance drift over time.

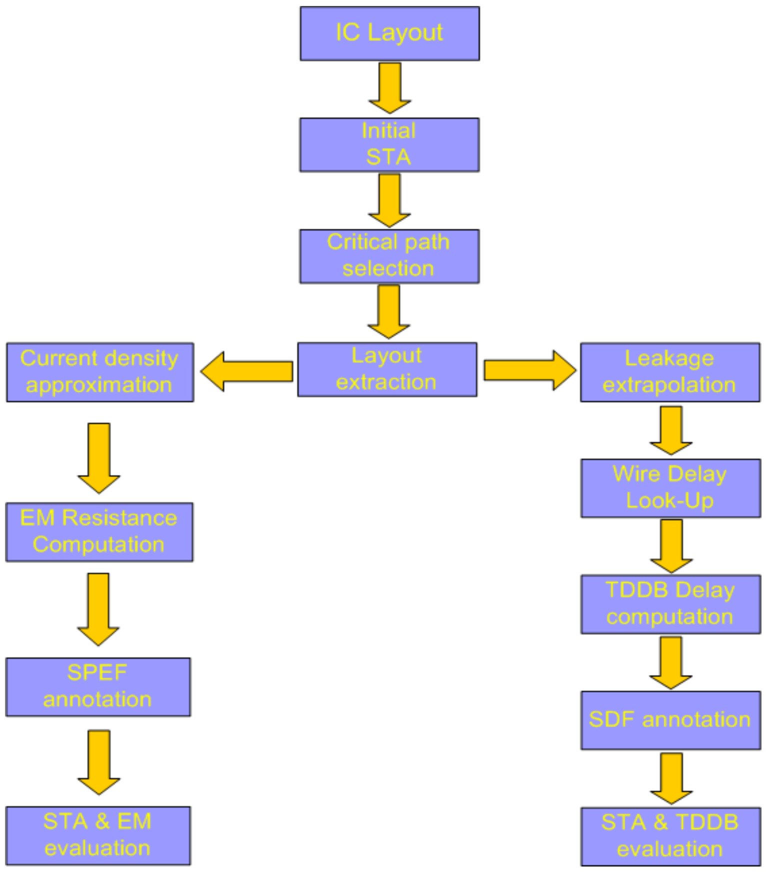


Figure 3.2. The initial reliability analysis framework presented in [1] and [2].

Regarding the required tools for the analysis described above, both flows are based on industrial EDA suites, which aid in the extraction of the layout-specific information and in the static timing analysis of the design. A slight differentiation exists regarding the temperature profiling, which is based on an open-source academic platform, as it is described in the next section.

3.2 Temperature estimation

The temperature for each unit is derived from the HotSpot tool, provided as an open-source temperature estimation framework, developed by the University of Virginia. The input is just the floorplan of all the design's units, the number of which can vary depending on the design's hierarchy levels and the desired granularity, as well as the power consumption for each of them. The result is the individual temperature estimation for each unit, which is strongly dependent on the power consumption's accuracy.

Based on the obtained temperature profile, the proposed flow, which is generic enough, can be used to analyze any path that is given as input, allowing the use of any path selection algorithm or method considered as appropriate, depending on specific parameters and criteria. The selection of specific paths may focus, for instance, on specific regions of the design that are suspected to be susceptible to the studied interconnect reliability problems, or on any set of architectural, temperature-aware or physical implementation criteria.

In perspective, the design flow consists of discrete steps in order to calculate all required data and we will elaborate on these steps further below. First of all, the temperature for each individual floorplan unit needs to be determined, through HotSpot. This requires the presence of:

- The design's floorplan, which designates the exact placing of each unit on the chip and can be directly exported from the design itself through the Cadence SoC Encounter place-and-route tool, using specific commands.
- The estimated power dissipation for each unit, which can be derived either by assuming certain switching activity information for the design's inputs, or by simulating the post-layout netlist of the design, based on a specific application. In this thesis, we performed the experiments by applying both practices, but the latter has been preferred, even though it is more time-consuming, because of its accuracy on the power profile's estimation.

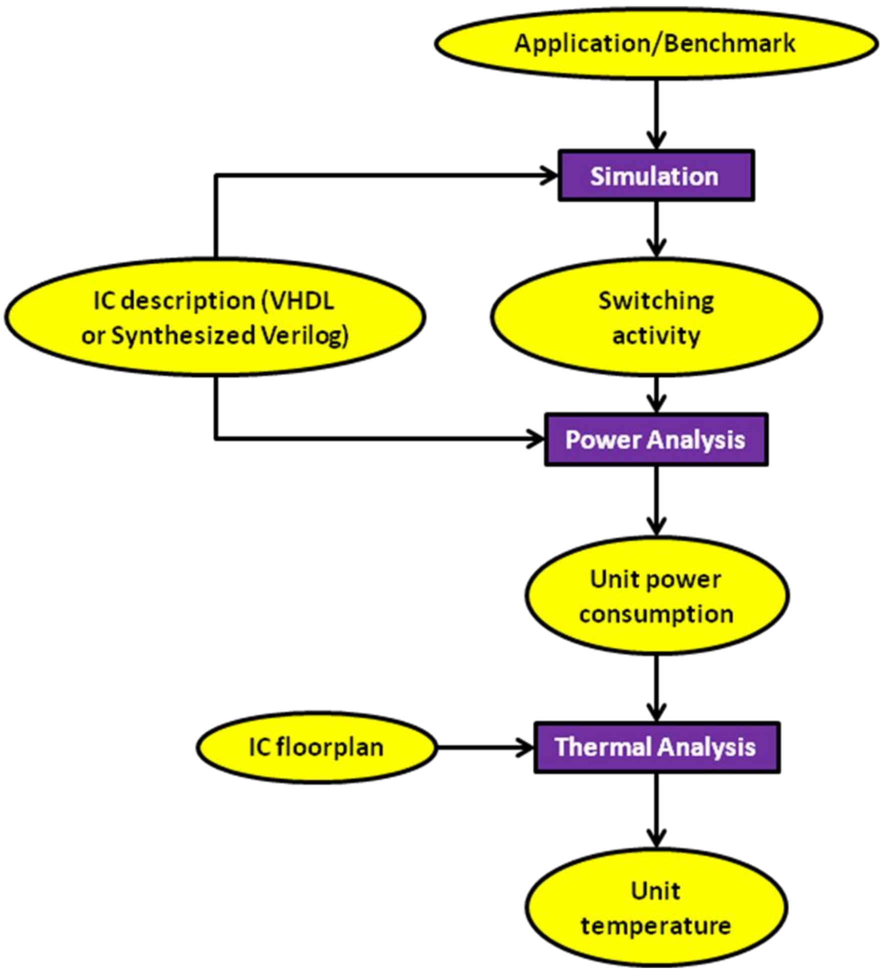


Figure 3.3. Temperature estimation flow.

3.3 EM flow

Once the temperature of each hierarchical unit is extracted, the necessary calculations, in order to obtain the results for the EM impact estimation can be performed. The EM flow consists of the following steps:

- *Cell, pin and net extraction*: The cells, pins and nets involved in the selected register-to-register critical paths are extracted from the timing analysis results.

- *Output transition time calculation*: The output pin rise and fall transition times can be retrieved through static timing analysis from within SoC Encounter using a DBAccess command. Alternatively, and in order to cross-check the command's results, the output transition time can be calculated through a lookup in the timing table for the specific cell in the standard-cell library, because it depends on the input pin transition time, the standard-cell driving strength and the output pin load capacitance. The library's format is assumed to conform to the Synopsys Liberty format's specifications, according to which the output transition time look-up is carried out using the input pin transition time, as a row index, and the output pin load capacitance, as a column index. Each input pin's transition time obviously equals the transition time of the output pin that drives that specific input pin and thus can be recursively calculated using the method described here. The output pin load capacitance is obtained using a relative Encounter DBAccess command.

- *Current density calculation*: The most accurate way to estimate the individual current of each wire of a net is through a Spice simulation. But in order to perform a Spice simulation of a charge and discharge cycle of a net, three components need to be modeled:

  - First of all, the wires of the net themselves. Their Spice netlist is generated from the corresponding net's SPEF distributed *RC* netlist.
  - Second, the input pins that the net is connecting the output pin with. These are modeled as capacitors, whose value is also derived from the SPEF file.
  - Finally, the output pin that drives the net. This has been modeled as a voltage source with specific parameters that closely resemble the behavior of an output pin of the system. Specifically, the voltage low level has been set to ground, the high level to V_{dd} and the rise and fall delays have been set to the rise and fall transition times that have been calculated through the extraction script for the specified output pin respectively.

  The simulation of the above Spice netlist is performed and the average value of the rise and fall current flowing through each wire is estimated. The equivalent current is then calculated as their absolute difference and used in order to estimate the evolution of EM through its model. The above procedure is repeated for every net that is part of the examined critical paths.

  An approximation formula for current density has also been used to cross-check the results of the above procedure. This formula is $\frac{V_{op}C_{net}}{t_{transition}A}$ where V_{op}

is the operating voltage, C_{net} is the total net capacitance, A is the cross section area of the net's wire that is driven by a specific standard-cell's output pin, and $t_{\text{transition}}$ is the transition time of the net. The output pin's load capacitance and the output net dimensions for the determination of its cross-section area are obtained using the respective Encounter DBAccess commands. The output transition times for the calculation of the rise and fall current density are known from the previous step. A great disadvantage of the above formula is its reduced accuracy due to ignoring the tree structure of the interconnect and thus the distribution of the current to each branch.

- *EM parameters estimation*: After having calculated the rise and fall current density as described above, the average current density is calculated as their absolute difference as the EM model is for DC current. The parameters t_{50}, R_{slope}, ΔR and l_{critical}, that quantify the impact of EM, are then calculated using the EM model described in the previous chapter and they are written along with the net's name to an intermediate file, which includes all the information regarding the wires of the examined path's nets which are affected by EM in the timing window of the desired system's lifetime. We will elaborate further on the generation, the structure and the context of this file, namely *deltaR.report*, in the next chapter. The computed EM parameters are needed later in the flow to update the elevated wire resistance values in the SPEF file of the design, in order to take the effect of EM into account while estimating the system's performance drift over time.

- *SPEF annotation*: Depending on the operation time period that is desired to be simulated, the total resistance increase for each net wire is calculated as the sum of the resistance step and the resistance slope multiplied by the number of years since t_{50}. It must be noted that all the aforementioned steps of this flow are performed for nets that belong to certain register-to-register paths of the target design. Then, using the net's name, its parasitic resistances are located in the SPEF file. It must be noted that only those wires longer than the critical length for the specified net must be updated. This is ensured by updating only the wires of each net with resistance values higher than the product of the critical length by the average resistance per length value.

- *Estimation of EM impact on system's timing*: A new timing analysis using the annotated SPEF file will reveal the impact of EM on the timing of the system and will indicate possible timing violations due to the EM-induced interconnect resistance rise.

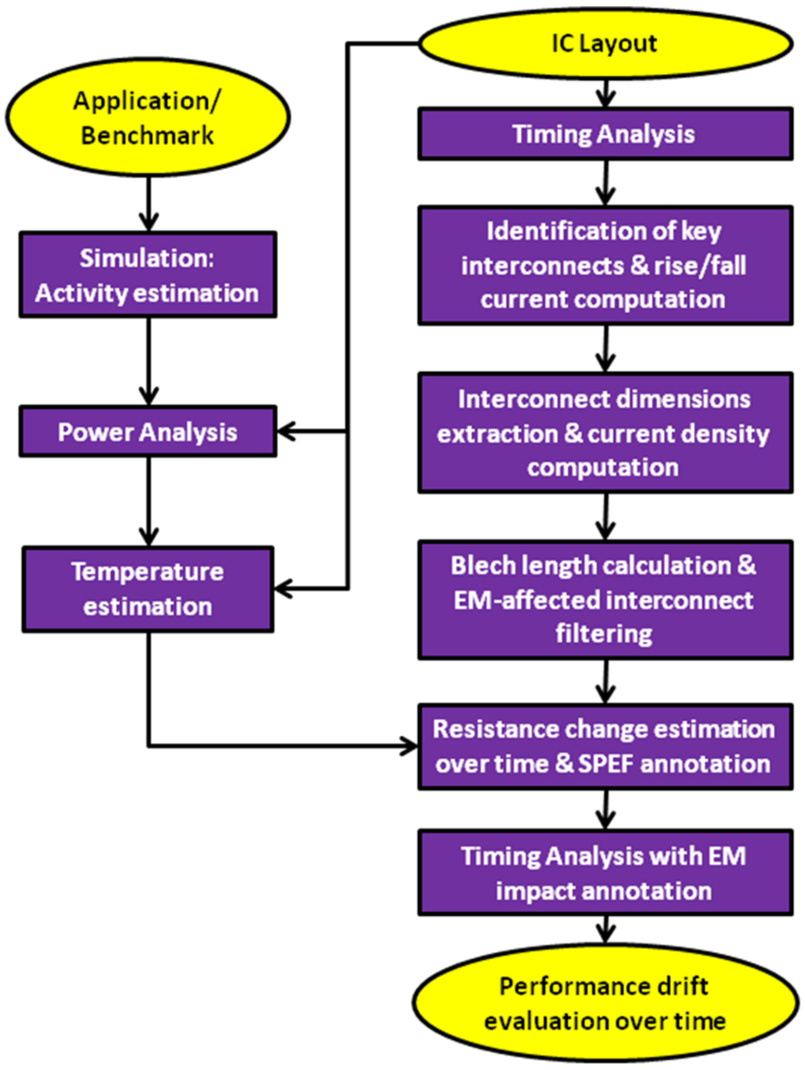

IC Layout
Application/ Benchmark
Timing Analysis
Simulation: Activity estimation
Identification of key interconnects & rise/fall current computation
Power Analysis
Interconnect dimensions extraction & current density computation
Temperature estimation
Blech length calculation & EM-affected interconnect filtering
Resistance change estimation over time & SPEF annotation
Timing Analysis with EM impact annotation
Performance drift evaluation over time

Figure 3.4. The temperature-aware EM flow.

3.4 TDDB flow

In this section of the flow, the interconnects that run parallel to each other, posing increased TDDB induced risk are identified and the impact of the leakage current between them on the system's timing is estimated. The TDDB flow consists of the following steps:

- *Detection of adjacent wires*: TDDB only takes place in wires that are close enough to each other. So, a filtering of all adjacent wires of each wire of a net must be carried out in order to identify them. The maximum distance can be altered, depending on the fabrication technology used. The wires that are discovered are written to a report with their positions and relative distances, which are exploited later in the tool flow. We will elaborate further on the generation, the structure and the context of this file, namely *wire.report*, in the next chapter.

- Estimation of the TDDB-induced leakage current: Using the adjacent wire data gathered before, a calculation of the leakage current by extrapolation of lab measurements under stress conditions to operating conditions is performed as described in the corresponding TDDB model, shown in Chapter 2.

- *Estimation of the delay change due to TDDB-induced leakage current*: The only way to obtain an accurate estimation of the impact of the TDDB-induced leakage current on the system's timing is a set of SPICE simulations to generate a pre-characterized library of representative layout patterns combined with a range of leakage currents. For the rest of the tool flow, on-the-fly simulations using real and accurate wire patterns would be too time consuming and not worth the extra timing overhead for each separate design, so another approach was preferred instead. This involved the generation of a LUT containing permutations of all possible TDDB-influencing values within certain ranges which are expected to be encountered throughout the design, along with their corresponding delay change ratios. The generation is performed based on a distributed *RC* model of the wire, with representative values of resistance *R* and capacitance *C*, as well as leakage current sources, all uniformly scattered across the overlapping section of the wire.
  The constructed LUT library contains key values of the wire's length, the adjacent wire (overlapping) length, the relative position of the wires, the leakage current and finally, the distance from each other. This LUT needs to be constructed only once for each CMOS technology of implementation. Afterwards, the delay change ratio of real wires is estimated using interpolation, so the overhead added is very limited. The characteristics of real wires are read from the report mentioned earlier. Hence, through a look-up table search, followed by an interpolation, if needed, the delay overhead for a specific wire is estimated.
  The following figure provides an insight into how the TDDB delay ratio of a net's wire depends on the wire's length and also on the adjacent wire's distance, when all other parameters (e.g. temperature) are kept constant.

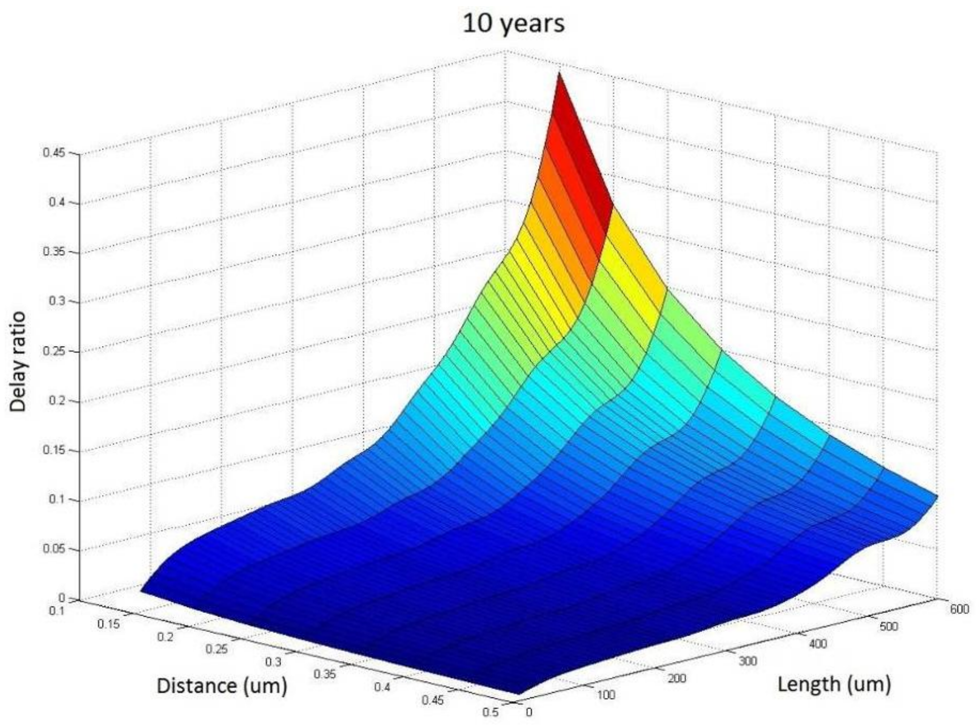


Figure 3.5. Delay impact on a wire due to TDDB, depending on wire length and distance.

- *SDF annotation*: The last step is to update the interconnect delays of the SDF file. Using the interconnect output and input pin, the initial delays are located and read. The total delay change ratio of each interconnect is calculated as the weighted mean (based on lengths) of the delay change ratio of its individual wires. Then the delay of that interconnect in the SDF file is incremented by the total delay change ratio.

- *Estimation of TDDB impact on system's timing*: A new timing analysis using the annotated SDF file will reveal the impact of TDDB on the timing of the system and will indicate possible timing violations due to the TDDB induced leakage current increase which leads to transition delay increase.

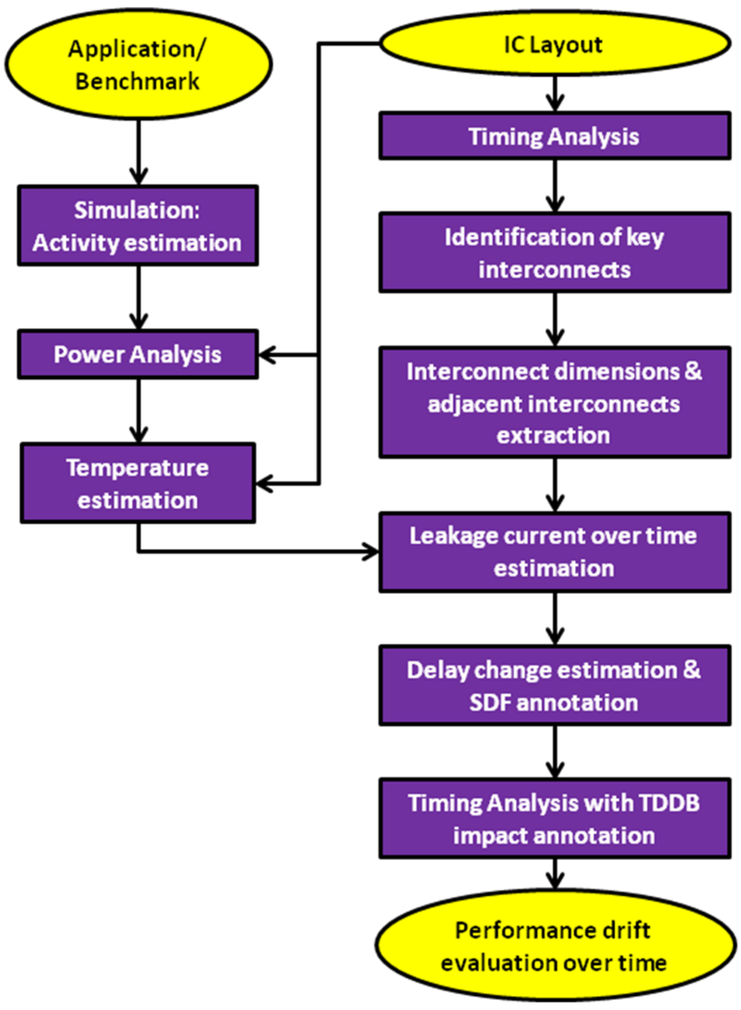

Application/ Benchmark
Simulation: Activity estimation
Power Analysis
Temperature estimation
IC Layout
Timing Analysis
Identification of key interconnects
Interconnect dimensions & adjacent interconnects extraction
Leakage current over time estimation
Delay change estimation & SDF annotation
Timing Analysis with TDDB impact annotation
Performance drift evaluation over time

Figure 3.6. The TDDB flow with the temperature profile's annotation from HotSpot.

3.5 Combined impact methodology

In the presented thesis, the studied reliability wear-outs, namely EM and TDDB, were considered as independent phenomena and consequently, the developed estimation framework captures the timing impact of each one of them without considering any inter-dependence between them. However, EM can affect the inter-metal distance of affected wires, as the void formation leads to extrusions of copper wires' segments at the cathode end. Hence, in such a case, the spacing between the extruded wire and its adjacent one on the same metal layer is locally reduced and the corresponding electric field is enhanced. It is noted that wire extrusions due to EM may occur only when current densities are significantly high and the specific wire is stressed for a long time, which is not a typical case in operating conditions. However, high peak currents in computational-intensive parts of designs such as processors or arithmetic units with high fan-out logic cells may include wires that could be candidate to suffer from extrusions, especially as the technology scaling shrinks the interconnect dimensions. As a result, the development of a methodology that could incrementally capture the combined impact of EM and TDDB gains in significance, as does the need for a model to estimate the location of possible wire extrusions.

From a purely technical point of view, the estimation of the combined impact of both reliability wear-outs on the target design's timing is based on a technique that uses an intermediate step after the annotation of the updated resistances due to EM into the design's SPEF file. This step involves the generation of an updated SDF file of the design through a static timing analysis, based on the annotated SPEF file that includes the impact of EM. In this way, the produced SDF file includes the impact of EM and it can be used as input to the TDDB analysis flow. The execution of the TDDB flow's steps leads to the generation of the final SDF file, including the total delay shifting because of both EM and TDDB. The combined impact is evaluated by the final static timing analysis for the examined design's paths.

In either the separate or the combined version of the proposed framework, the analysis of the steps required to evaluate the timing impact of the studied reliability phenomena is viewed from a rather abstract point of view, as it is presented in this chapter. However, in the next chapter, we elaborate further on the detailed implementation of the aforementioned design flow steps, describing each part of our tool flow in detail, including the DBAccess commands used for the layout extraction and the way each step interacts with the industrial EDA tools. Also, a view on the Encounter DBAccess layout extraction tools is provided, along with the analysis of the scripts used for the proposed flows' implementation.

4

Design flow implementation & Automation

4.1 Flow implementation tools and components

This paragraph covers the implementation details of the design flow including mainly the technical section, such as the tools used and the scripts written to realize each of the EM and TDDB flow steps.

4.1.1 Temperature estimation (HotSpot)

The units' temperature estimation is done with the help of an academic tool named HotSpot that is developed, maintained and distributed by the University of Virginia. HotSpot is an accurate and fast thermal model suitable for use in architectural studies. The actual tool is based on an equivalent circuit of thermal resistances and capacitances that correspond to micro-architectural blocks and essential aspects of the thermal package. The model has been validated using finite element simulation.

HotSpot has a simple set of interfaces and hence can be integrated with most power-performance simulators. The main advantage of HotSpot is that it is compatible with the kinds of power and performance models used in the computer-architecture community, requiring no detailed design or synthesis description. HotSpot makes it possible to study thermal evolution over long periods of real, full-length applications.

HotSpot requires a configuration file which contains various fabrication and packaging dependent parameters such as the chip and spreader thickness. Other than that, being generic, it only requires two input files, which contain the units' placement on the chip and each unit's power consumption respectively. These can be produced by any tool or method the user prefers.

In this case, the floorplan data is exported from the Cadence SoC Encounter tool using the *saveFPlan* command followed by the name of the floorplan (*.fp*) file, which is read by a conversion script and printed to a new *.flp* file format. That filters out the lines starting with "Guide:" which contain each unit's coordinates, namely bottom-left-x, bottom-left-y, upper-right-x, upper-right-y. The new format requires each unit's width and height, as well as the bottom-left-x and bottom-left-y coordinates.

As for the power consumption profile, an application-specific post-layout simulation is performed using the Mentor Graphics ModelSim tool, the activity *.vcd* file is read by Synopsys PrimeTime PX, which reports the power of each unit to a power-report file. An addition of the leakage and dynamic power, which equals the total power, is performed by a script for each individual unit and the result is written to a new file in the HotSpot's power trace format.

HotSpot takes the compatibility-converted versions of the above power trace and floorplan files as inputs and generates the corresponding transient temperatures onto a temperature trace file. There is also an option to output the final steady-state temperatures onto a file. This is useful, as the steady-state temperatures of the first thermal simulation's iteration can be used to perform a second simulation for improved accuracy. After the completion of the initial thermal simulation, the temperature file does contain a thermal trace, but the initial temperatures that were used to generate it were default constant values. These might not be representative if the simulation is not long enough to warm up both the chip and the package. However, the steady state temperatures are a good estimation of what the correct set of initial temperatures are. So, the steady state temperatures produced by the initial run can be used as the set of initial, starting-point temperatures for one final run, a strategy producing more accurate results.

It should be noted that HotSpot has been designed with high performance systems' thermal simulation in mind, so in order to simulate the lack of a heatsink in embedded systems, the thickness of the heat spreader and the heatsink in the configuration file has been reduced to a few tens of micrometers, as the developers of HotSpot advised. These settings made the impact of both the heat spreader and the heatsink on the heat transfer path negligible. Lower values are not used, as they could lead to numerical errors. Also, special care has been taken to ensure that all simulations are long enough, so that the system has reached its steady state temperature, otherwise the extracted temperature traces for each floorplan unit might be underestimated.

4.1.2 Path extraction and formatting

The register-to-register path extraction can be done by any preferred or appropriate method. Special paths can be investigated in some cases, where specific regions of a design are suspected to be particularly weak in terms of reliability and lifetime. Some criteria to locate such regions are proposed in the concluding paragraph of Chapter 6.

In the experiments performed in the context of this work, the most timing-critical paths are considered as the most likely to suffer from a timing violation due to the impact of EM or TDDB on their interconnects, as the least deterioration of their characteristics may be fatal, because of the small slack available. An initial static timing analysis of the design using the Cadence Encounter Timing System produces a report containing these paths. The steps required include reading the design's synthesized Verilog netlist (*read_verilog <verilog filename>*), the loading of the necessary standard-cell technology libraries (*read_lib <library filenames>*), the creation of a clock for the design (*create_clock <clk pin name> -name <clk name> -period <clk period>*) and then, the extraction of the report including the timing of paths with the least slack (*report_timing –machine_readable –nworst <amount of paths>*), which is dumped to a file in a format that is easy to parse.

The file including the report for the most timing-critical paths is then converted using a Tcl script, so that the produced file contains only the needed information of each path. This information includes the driving cell and the corresponding output pin, the net connecting the two cells and the driven cell, along with its input pin driven by the net, given in the following format:

net input-pin(net start) output-pin(net end)
net input-pin(net start) output-pin(net end)
...

Example:

A sample line demonstrating the format described above follows.

core0/leon3core0/mctrl0/n221 core0/leon3core0/mctrl0/U399/A1 core0/leon3core0/
mctrl0/r_reg_BUSW__0_/QN

This format allows the file to be both easily parsed and comprehended by humans, so that the process can be supervised, error-checked and custom paths can be easily added. Based on the parsing of this file, we derive the path's nets and we therefore extract all the wires of each net, by following the steps described in the next section.

4.1.3 Layout geometric data extraction using Encounter DBAccess

Cadence DBAccess is a command set which allows direct access to the internal database of the SoC Encounter hierarchical RTL-to-GDSII physical implementation solution. It is intended for highly experienced designers who are proficient in the use of SoC Encounter. As the whole SoC Encounter user interaction shell, this command set is also based on the Tcl programming language. The commands offered allow the designer to access and set layout specific data. As a result, quite complex scripts can be executed from within SoC Encounter, allowing layout-specific calculations to be carried out. The objects present in DBAccess as well as the commands that help the transition from one object to another (object relations) are displayed in Figure 4.1. Cadence DBAccess has been used to implement the core of the tool flow presented in this work. For instance, all of the layout's data such as nets' wires and adjacent wires, as well as other data such as cell-net timing, have been extracted using DBAccess. In order to create a work that is as complete as possible, the most useful commands are presented below together with their syntax and a short description of their function.

dbInstCellName *<cell instance pointer>*

- Arguments: A pointer to a cell instance in the design.
- Result: The cell (master) name for the specified cell in the standard cell library.

dbHeadMicronPerDBU

- Arguments: -
- Result: The microns that one database unit (DBU) equals to.

dbForEachNetWire *<net pointer> <wire pointer variable> <processing loop body>*

- Arguments: A pointer to a net, a variable that will hold the pointer to the current wire of the net for each iteration, the command block that will be executed for each wire of the net.
- Result: The processing loop is executed for every wire of the specified net.

dbGetLayerByZ <Z layer>

- Arguments: An integer, representing the metal layer index (0, 1, 2, …).
- Result: The address of the specified layer.

dbGetNetByName <net name>

- Arguments: The name of a net.
- Result: The address of the net with the specified name.

dbGetNetTotCap *<net pointer>*

- Arguments: A pointer to a net.
- Result: The total net capacitance of the specified net, including the cell pins that constitute its terminals.

dbHeadOhmPerDBU

- Arguments: -
- Result: The Ohms that one database unit (DBU) equals to.

dbHeadPicoFPerDBU

- Arguments: -
- Result: The pico Farads that one database unit (DBU) equals to.

dbHeadPicoSecPerDBU

- Arguments: -
- Result: The pico seconds that one database unit (DBU) equals to.

dbWireLen <*wire pointer*>

- Arguments: A pointer to a wire in the design.
- Result: The length of the specified wire in DBU.

dbWireZ <*wire pointer*>

- Arguments: A pointer to a wire in the design.
- Result: The metal layer index for the specified wire (0, 1, 2, …).

dbLayerThickness <Z *layer address*>

- Arguments: The address of a Z layer in the design (this can be retrieved through the dbGetLayerByZ command).
- Result: The thickness of the specified Z layer in DBU.

dbWireBox <*wire pointer*>

- Arguments: A pointer to a wire in the design.
- Result: The box of the specified wire. This box is defined by the coordinates of the lower-left and the upper-right corners of the rectangle that encloses the specified wire. The coordinates are returned in this order:
  lower-left-x (llx) lower-left-y (lly) upper-right-x (urx) upper-right-y (ury)
  and are in DBU.

dbWireDir <*wire pointer*>

- Arguments: A pointer to a wire in the design.
- Result: The direction/orientation of the specified wire. There are four possible values, namely *dbcWireN*, *dbcWireS*, *dbcWireE* and *dbcWireW*. Obviously, the first two values indicate a vertical orientation and the remaining two a horizontal orientation.

dbNetLenX <*net address*>

- Arguments: The address of a net in the design.
- Result: The total length of all the wires of the specified net that run horizontally.

Obviously, this command can be used to calculate the total length of a net.

dbNetLenY <*net address*>

- Arguments: The address of a net in the design.
- Result: The total length of all the wires of the specified net that run vertically.

Obviously, this command can be used to calculate the total length of a net.

dbGetTermByName *<cell instance pointer> <cell pin name>*

- Arguments: A pointer to a cell instance in the design and the name of a pin of that cell.
- Result: The address of the terminal specified by the cell instance and pin name.

dbTermTranTime *<terminal address>*

- Arguments: The address of a terminal in the design (this can be retrieved through the dbGetTermByName command).
- Result: The transition times of the specified terminal in this order:
  rise time   fall time
  and in DBU.

A timing analysis through the *delayCal* command in required, prior to the execution of this command.

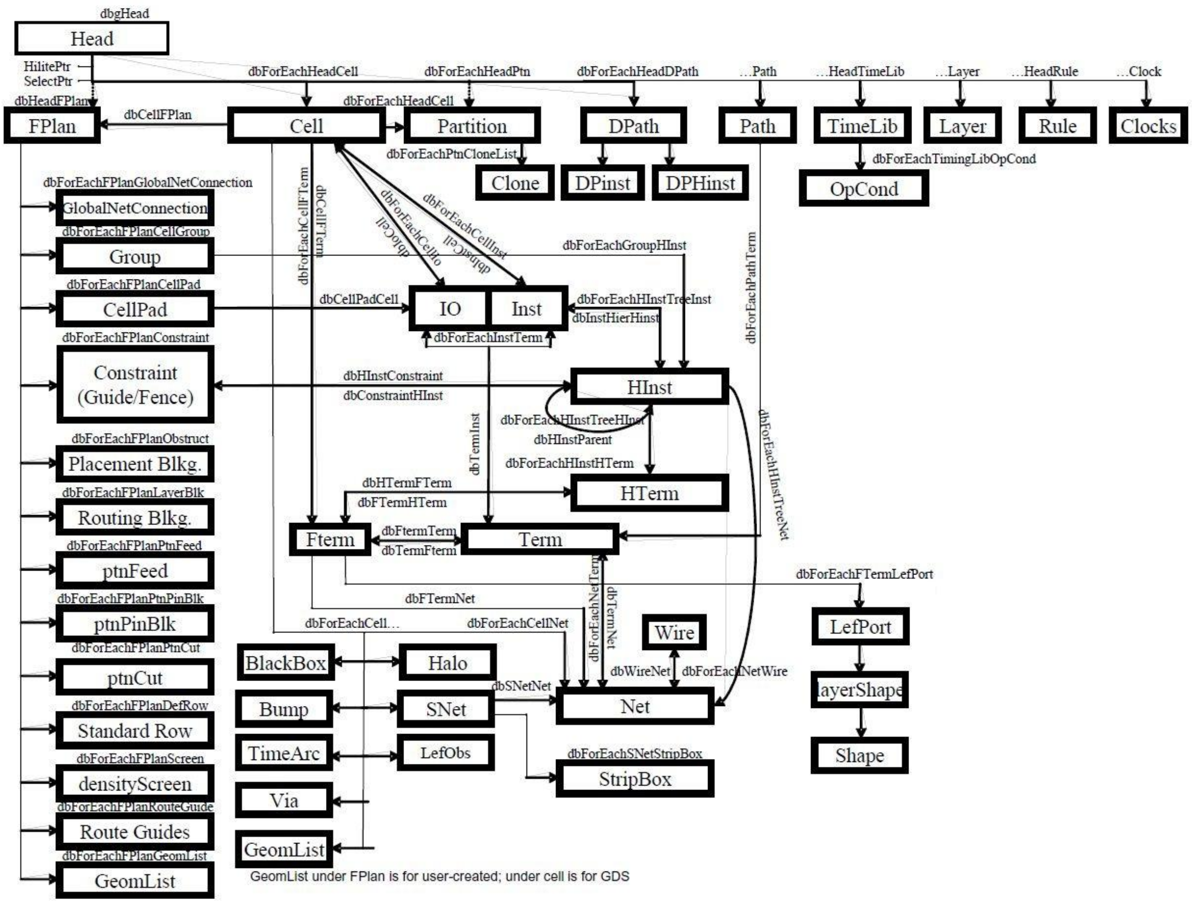


Figure 4.1: DBAccess objects and commands relating them.

Another SoC Encounter command that is very important for the TDDB processing of the flow, will be listed here despite <u>not</u> being part of the DBAccess command set:

findNetsInBox *<llx> <lly> <urx> <ury>*

- Arguments: Four coordinates in microns that define a box as described above, through its lower left and upper right corner coordinates.
- Result: A list containing the addresses of all nets that have wires intersecting within the specified sector of the bounding box.

4.1.4 Rise & fall transition time estimation

The rise and fall transition times of a cell's output pin that drives a net can be computed using two different methods, both of which have been implemented in order to cross-check the validity of their results. A description of each method is provided below.

The following EDA-tool-oriented method is preferred, as it makes use of only a DBAccess command, namely *dbTermTranTime*. Its output is calculated through static timing analysis using detailed parasitic data. As a result, it is rather accurate and quick in terms of execution time compared to the following method.

The alternative method presented below that was initially used in the EM flow, assumes that the designs used as testbenches are composed of standard-cells, whose characteristics are documented in a CMOS standard-cell technology library, provided in a Synopsys Liberty format file (*.lib*). Among their characteristics, which depend on the CMOS standard-cell library used, are their transition times and power consumption, as well as the capacitance of each one of their pins. A detailed description of the specific timing data format is provided below, as it has been used to estimate the rise and fall times of each cell's output.

At the beginning of the library general information such as units and operating conditions, such as voltage and temperature are given. Further below some templates are found. These define the type and values of each parameter in the timing look-up tables. Sometimes these templates are organized in this section in groups and some others, individual templates can be defined inside the section of each standard cell. The most common parameters used are the cell's input transition time (*input_net_transition*) and its output capacitance (*total_output_net_capacitance*). A vector of the values of each parameter for which an output transition time has been recorded is provided in the template. A template sample from the 45nm library used is the following:

```
rise_transition (delay_template_7x7_0) {
  variable_1 : input_net_transition;
  variable_2 : total_output_net_capacitance;
  index_1 ("0.0033, 0.0068, 0.0137, 0.0276, 0.0554, 0.1109, 0.2219");
  index_2 ("0.00045, 0.00183, 0.00459, 0.01012, 0.02117, 0.04326, 0.08746");
}
```

In the above template sample, *delay_template_7x7_0* is a code that identifies the type of the template. In this case, the code indicates the amount of values contained in the template. The first variable is marked as *variable_1* and it represents the input transition time, while the *index_1* vector contains the corresponding values. According to the Liberty specification, as described in [10], the first variable determines the row of the look-up table that contains timing data for the specific parameter value. The second variable, respectively, is marked as *variable_2* and it represents the output capacitance, while the *index_2* vector contains the corresponding values. According to the Liberty specification, the second variable determines the column of the lookup table that contains timing data for the specific parameter value.

So, by knowing the values of both parameters for a specific cell in a circuit, its output transition times can be calculated by reading the corresponding value from the look-up table. These lookup tables are located in the timing section of each cell. As expected, there is a different table depending on the input pin whose state changes and on whether the state changes from high to low (fall) or from low to high (rise). This happens because the transition time of the output pin depends on the cell's implementation both on logic level and on physical level. The timing section of a specific input pin of a specific cell follows:

```
timing () {
  related_pin : "B1";
  timing_sense : negative_unate;
  rise_transition (delay_template_7x7_0) {
  index_1 ("0.0033, 0.0068, 0.0137, 0.0276, 0.0554, 0.1109, 0.2219");
  index_2 ("0.00045, 0.00183, 0.00459, 0.01012, 0.02117, 0.04326, 0.08746");
  values ( \
    "0.02502, 0.03424, 0.05257, 0.08929, 0.1626, 0.3079, 0.5993", \
    "0.02501, 0.03418, 0.05259, 0.08922, 0.1625, 0.3083, 0.5995", \
    "0.02506, 0.03424, 0.05256, 0.08929, 0.1621, 0.3083, 0.5995", \
    "0.02519, 0.03421, 0.05264, 0.08935, 0.1624, 0.3084, 0.5995", \
    "0.02977, 0.0373, 0.05336, 0.08917, 0.1624, 0.3083, 0.5998", \
    "0.04069, 0.04852, 0.06371, 0.09363, 0.1624, 0.3082, 0.5997", \
    "0.06026, 0.06957, 0.0864, 0.1169, 0.1751, 0.309, 0.5998" \
    );
  }
  fall_transition (delay_template_7x7_0) {
  index_1 ("0.0033, 0.0068, 0.0137, 0.0276, 0.0554, 0.1109, 0.2219");
  index_2 ("0.00045, 0.00183, 0.00459, 0.01012, 0.02117, 0.04326, 0.08746");
  values ( \
    "0.01842, 0.02387, 0.03475, 0.05651, 0.09953, 0.1852, 0.3569", \
    "0.01841, 0.02389, 0.03477, 0.05652, 0.09953, 0.1854, 0.3566", \
    "0.0183, 0.02375, 0.03474, 0.05643, 0.0995, 0.1852, 0.3569", \
    "0.0196, 0.02428, 0.03461, 0.05643, 0.09949, 0.1853, 0.3565", \
    "0.0266, 0.03099, 0.03909, 0.05729, 0.09956, 0.1854, 0.3565", \
    "0.03829, 0.04391, 0.05384, 0.07081, 0.1036, 0.1854, 0.3565", \
    "0.05738, 0.06504, 0.07784, 0.09905, 0.1339, 0.1976, 0.3567" \
    );
  }
}
```

The timing header indicates the beginning of the timing data section. Directly below, the input pin whose state changes is noted together with its logic polarity and the two lookup tables, one for output rise transition time and one for output fall transition time. It should be noted, that the transition time is defined as the time interval between the moment a waveform reaches 10% of its total transition and the moment it reaches 90% of it. This time interval is displayed in the next figure.

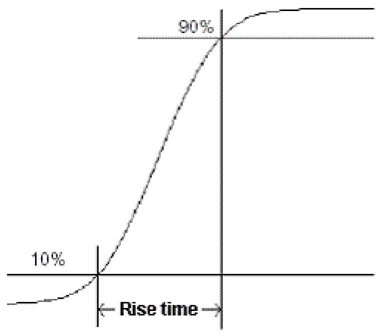


Figure 4.2. Transition (rise) time, counted from the 10% to the 90% of the final value.

The timing section also contains two more lookup tables which are not visible in the above sample which are named *cell_rise* and *cell_fall*. These contain the timing data for the cell's delay, which is the time interval from the moment the input reaches 50% of its total transition until the output reaches 50% of its transition.

The calculation method for the output transition time of the above cell is demonstrated below with a simple example, where the transition time of the input pin B1 is assumed to be 0.0276 ns and the cell's output capacitance is assumed to be equal to 0.00183 pF. These values correspond to the 4th entry of the *index_1* vector, so the *4th* row of the lookup table is selected, and to the *2nd* entry of the *index_2* vector, so the *2nd* column of the lookup table is selected. The value selected in the lookup table equals to the output rise transition time and is 0.03421 ns.

In the case where the exact values of the circuit's input transition time and output capacitance are not present in the template's indexes, linear interpolation between the closest values should be performed, in order to estimate the output transition time.

4.1.5 Prerequisite files generation

In order to proceed with all the necessary flow calculations and steps, certain information distributed in several files is required to be generated before the beginning of the flow's execution, including the computations within the models and the annotation of the model's impact on interconnects, in terms of increased resistance (EM) or delay overhead (TDDB). The tools which generate each of the output files of the proposed flow are displayed in Figure 4.3.

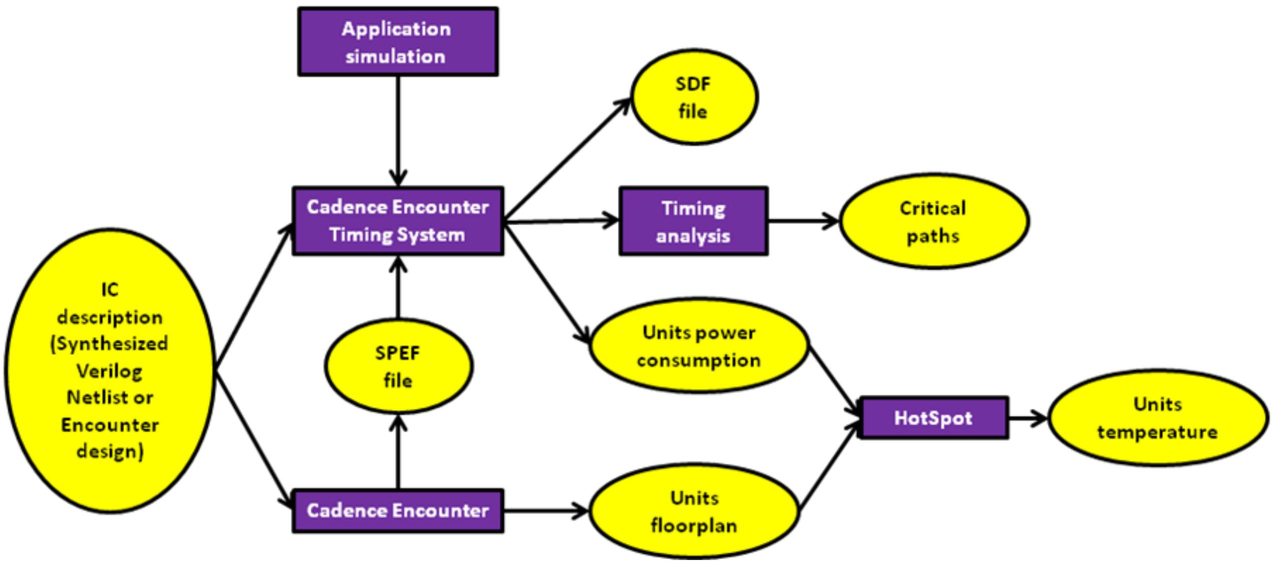


Figure 4.3. Prerequisite files generation.

4.2 EM flow

This paragraph covers the implementation details of the EM flow including the technical details and intermediate computations and steps that help reach to the final results. The EM flow's steps will be analyzed next in order of execution. This flow consists of three discrete scripts, the extraction script, the current computation script and the update script, plus one final step to convert impact on electrical characteristics degradation to system timing.

4.2.1 EM extraction script

This script is written in Tcl and is called from the SoC Encounter's environment, using the *source <script filename>* command after loading the design to be analyzed for reliability issues. This allows the script to retrieve layout-specific data from the Encounter database using a subset of DBAccess commands.

Inputs:

- Standard-cell and possibly memory technology libraries, if applicable.
- Hierarchical units' temperature trace file
- Encounter design file (.enc)
- Critical paths file

Outputs:

The *deltaR.report* file, which contains the data that quantify the impact of EM on the system in terms of interconnect resistance increase over time, namely:

- t_{50}: the time period required for the formation of a void that fully occupies the cross section of an interconnect.
- EM resistance step: the abrupt interconnect resistance increase after the t_{50} time period because of the current flow through the diffusion barrier.
- EM resistance slope: the gradual interconnect resistance increase after the t_{50} time period because of the expansion of the void along the length of the specific interconnect.
- Blech length: the minimal length for which EM occurs in a specific interconnect according to its current density

The above set of parameters is calculated and reported for each net contained in the critical paths along with its name and the number of wires exceeding that net's Blech length.

Critical path cell, pin & net extraction:

Each critical path consists of a set of *output pin, net, input pin* tuples. The script starts from the first output pin which should belong to a register and then processes each tuple after the other extracting the output cell and pin, the net code and the cell input. In each iteration of the script's main loop, the following data are needed:

- The cell and its corresponding output pin that drives the current net.
- The cell's input pin, with which the previous tuple's net is terminated.
- The current net's hierarchical name.

These are extracted by parsing the corresponding tuple data. The pin names and net codes are in the last hierarchy level, whereas the cell instance codes are in the level just before that. Encounter DBAccess is used to identify the cell name from the cell instance code through the *dbInstCellName <cell instance code>* command.

Temperature estimation:

By parsing the net's hierarchical name and cutting off the net's code (e.g. n341), the unit to which the net belongs can be determined. Then, the temperature of that unit can be retrieved from the temperature trace file produced by HotSpot and can be set as the current temperature to increase the accuracy of the calculations to be executed.

Output transition time calculation:

The timing data of the cell and its pins that were determined previously, are located in the standard cell library file with successive searches for the keywords: *cell(<cell name>), timing()* and *related_pin : "<input pin name>";*. Then the required rise and fall transition timing data are read from the *rise_transition* and *fall_transition* sections. Each one contains the template indexes under *index_1(...)* and *index_2(...)* and the output transition time lookup table under *values() {...}*. The command *dbGetNetTotCap <net code>* returns the total output capacitance, whose closest match in the index_2 vector specifies the correct look-up table column. The input transition time equals to the output transition time of the path's previous net and its closest match in the index_1 vector specifies the correct look-up table row. The LUT value designated by the above row and column indexes, or the interpolation of the closest matches, is this output's transition time. This process is repeated twice, in order to calculate both the rise and fall transition time of the output.

Current density computation:

The extraction script that runs from within SoC Encounter creates a file named *nets.txt*. This contains a list of the nets of all the critical paths examined together with each net's rise time, fall time and temperature. This file is used by the *spef2spice.tcl* script that is written in Tcl in order to generate the Spice netlist of each net, based on the information from the SPEF and the *nets.txt* files.

Once a net and its parameters are read from the *nets.txt* file, the generation of its Spice netlist consists of 3 steps:

- Wires' netlist: The generation of the distributed *RC* Spice netlist from the net's netlist found in the SPEF file. The generation of the netlist starts with the retrieval of the net's code from the name map of the SPEF file through its name. Then, the corresponding D_NET section is located and for every capacitance in the CAP section and every resistance in the RES section, a capacitance and a resistance are inserted in the Spice netlist respectively. In the CAP section, the 2nd argument of a line is the SPEF node code between which and the ground the capacitance should be connected, whereas the 3rd argument is the value of the capacitance. In the RES section, the 2nd and 3rd arguments are the SPEF node codes between which the resistance should be connected, whereas the 4th argument is the value of the resistance.
  The concept for the creation of the correct nodes for the Spice netlist is based on a node map. Every time a new SPEF node is encountered, it gets assigned a number, which defines its Spice netlist node's code and it is stored in the node map. If it is encountered again in the SPEF netlist, the node's code is retrieved from the node map and it is used again, thus leading to a consistent netlist. The Spice netlist syntax of either a resistor or a capacitor is the following:

  R/C<resistance/capacitance number> <node1> <node2> <value>

- *Input pins*: The input pins of various cells that are connected to the specified net, are found in the CONN section of the net in the SPEF file. In the CONN section, the 2nd argument is the SPEF node code, the 3rd argument is either I (input pin) or O (output pin) and can be used to distinguish them and the pin's capacitance value is contained in the 8th argument [24]. The node map is used in this step as well in order to end up with a consistent Spice netlist.

- Output pin: The output pin that drives the net is found in the CONN section and can be easily identified from the O (output pin) keyword as described above. The Spice netlist command for a voltage source is the following:
  V<voltage source number> <node1> <node2> PULSE (Vlow Vhigh initial_delay rise_time fall_time pulse_width period)
  , where Vlow is the ground, Vhigh is the V_{dd} voltage, initial_delay is the delay before the first pulse edge occurs, rise_time and fall_time are the pulse rise and fall times respectively, pulse_width is the width of the pulse and period is the pulse period. The rise and fall times are the estimated rise and fall times for the specified output pin and the *node2* always is the ground in this case.

Also, in order to measure and report the current flowing through each wire which is represented by a resistor, the following commands must be inserted for every resistor:

.MEASURE TRAN rise_start<resistor number> WHEN V(<node1>)=0.1<V_{dd}> CROSS=1 PRINT=0*
.MEASURE TRAN rise_stop<resistor number> WHEN V(<node1>)=0.9<V_{dd}> CROSS=1 PRINT=0*
.MEASURE TRAN fall_start<resistor number> WHEN V(<node1>)=0.9<V_{dd}> CROSS=2 PRINT=0*
.MEASURE TRAN fall_stop<resistor number> WHEN V(<node1>)=0.1<V_{dd}> CROSS=2 PRINT=0*
.MEASURE TRAN I_RISE_R<resistor number> AVG I(R<resistor number>) FROM=rise_start<resistor number> TO=rise_stop<resistor number> PRINT=0
.MEASURE TRAN I_FALL_R<resistor number> AVG I(R<resistor number>) FROM= fall_start<resistor number> TO= fall_stop<resistor number> PRINT=0
.MEASURE I_EM_R<resistor number> PARAM=`abs(I_RISE_R<resistor number> +I_FALL_R<resistor number>)`

Only the last measurement, which corresponds to the equivalent EM current, is reported, while the PRINT=0 option prevents the others from being printed in the report. The first four commands are used to find out the time when a terminal node of the examined resistor reaches 10% and 90% of its rise or fall transition voltage respectively. These measurements are used to designate the rise and fall transitions, during which the current must be measured. Figure 4.4 displays a sample waveform that demonstrates the need for individual rise and fall transition time detection for each resistor individually due to propagation delay along the interconnect. The voltage of the output pin is displayed in green color, the voltage of a random node along the interconnect is displayed in blue and the current through the resistor corresponding to the wire of that node is displayed in red.

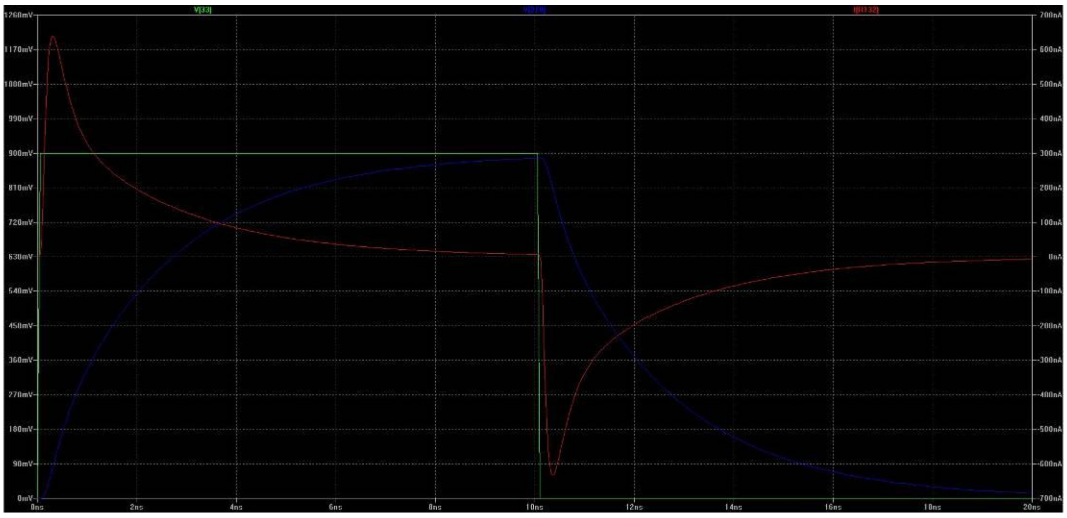


Figure 4.4. Waveform demonstrating the delay shifting of a resistor node's transition, compared to the transition of the voltage source.

It is clearly visible that the transition time (10%-90% charge/discharge) of a random wire along the interconnect is different from that of the output pin, as well as it is also shifted in time, in comparison to the time instant the output pin is completely charged or discharged. As a result, the identification of the exact transition interval for each resistor individually is critical for the accurate measurement of each wire's current. The next two commands are used to measure the average current during each one of these time intervals (rise and fall transition). The last command computes the equivalent EM current of each wire as the absolute difference of the rise and fall currents. This is read from the spice report during the next step of the *spef2spice.tcl* script and it is used to calculate the EM parameters through the respective model's functions.

In order to locate the wires that are longer than the Blech length and consequently suffer from EM, both the current and the length of a wire are required. The current of each wire is calculated through the Spice simulation as described above, so only the length of the wire that corresponds to each resistor in the SPEF file needs to be calculated. This can be done through its resistance value and the assumption of an approximate resistance per length value that is library and technology dependent. The quotient of these two values gives the approximate length of each wire.

Example:

The following example demonstrates the spice netlist complete with its *.MEASURE* commands, as it is generated by the script for a certain net contained in a SPEF file. The rise and fall times of the output pin that drives the net are assumed to be 20ps and 15ps respectively, as calculated from the extraction script running from within SoC Encounter.

Net’s entry in SPEF file:

```
*D_NET *39250 0.00032406

*CONN
*I *97074:ZN O *C 24 413 *L 0 *D AOI32D1BWP
*I *97072:A2 I *C 20 412 *L 0.000523 *D OAI31D0BWP

*CAP
1 *97074:ZN 0.00010262
2 *39250:4 0.00011102
3 *39250:3 4.883e-05
4 *39250:2 4.479e-05
5 *97072:A2 1.68e-05

*RES
1 *97072:A2 *39250:2 6.7
2 *39250:2 *39250:3 2.78
3 *39250:3 *39250:4 6.7
4 *39250:4 *97074:ZN 14.168
*END
```

Generated Spice netlist:

```
.OPTIONS LIST NODE POST
.OP
VIN 1 0 PULSE (0 0.9 0 20e-12 15e-12 10000e-12 20000e-12)
C_i3 2 0 5.23e-16
C1 1 0 1.0262e-16
C2 3 0 1.1102e-16
C3 4 0 4.883e-17
C4 5 0 4.479e-17
C5 2 0 1.68e-17
R1 2 5 6.7
.MEASURE TRAN risestart_r1 WHEN V(2)=0.09 CROSS=1 PRINT=0
.MEASURE TRAN risestop_r1 WHEN V(2)=0.81 CROSS=1 PRINT=0
.MEASURE TRAN fallstart_r1 WHEN V(2)=0.81 CROSS=2 PRINT=0
.MEASURE TRAN fallstop_r1 WHEN V(2)=0.09 CROSS=2 PRINT=0
.MEASURE TRAN I_RISE_R1 AVG I(R1) FROM=risestart_r1 TO=risestop_r1 PRINT=0
.MEASURE TRAN I_FALL_R1 AVG I(R1) FROM=fallstart_r1 TO=fallstop_r1 PRINT=0
.MEASURE I_EM_R1 PARAM=`abs(I_RISE_R1+I_FALL_R1)`
R2 5 4 2.78
.MEASURE TRAN risestart_r2 WHEN V(5)=0.09 CROSS=1 PRINT=0
.MEASURE TRAN risestop_r2 WHEN V(5)=0.81 CROSS=1 PRINT=0
.MEASURE TRAN fallstart_r2 WHEN V(5)=0.81 CROSS=2 PRINT=0
.MEASURE TRAN fallstop_r2 WHEN V(5)=0.09 CROSS=2 PRINT=0
.MEASURE TRAN I_RISE_R2 AVG I(R2) FROM=risestart_r2 TO=risestop_r2 PRINT=0
.MEASURE TRAN I_FALL_R2 AVG I(R2) FROM=fallstart_r2 TO=fallstop_r2 PRINT=0
.MEASURE I_EM_R2 PARAM=`abs(I_RISE_R2+I_FALL_R2)`
R3 4 3 6.7
.MEASURE TRAN risestart_r3 WHEN V(4)=0.09 CROSS=1 PRINT=0
.MEASURE TRAN risestop_r3 WHEN V(4)=0.81 CROSS=1 PRINT=0
.MEASURE TRAN fallstart_r3 WHEN V(4)=0.81 CROSS=2 PRINT=0
.MEASURE TRAN fallstop_r3 WHEN V(4)=0.09 CROSS=2 PRINT=0
.MEASURE TRAN I_RISE_R3 AVG I(R3) FROM=risestart_r3 TO=risestop_r3 PRINT=0
.MEASURE TRAN I_FALL_R3 AVG I(R3) FROM=fallstart_r3 TO=fallstop_r3 PRINT=0
.MEASURE I_EM_R3 PARAM=`abs(I_RISE_R3+I_FALL_R3)`
R4 3 1 14.168
.MEASURE TRAN risestart_r4 WHEN V(3)=0.09 CROSS=1 PRINT=0
.MEASURE TRAN risestop_r4 WHEN V(3)=0.81 CROSS=1 PRINT=0
.MEASURE TRAN fallstart_r4 WHEN V(3)=0.81 CROSS=2 PRINT=0
.MEASURE TRAN fallstop_r4 WHEN V(3)=0.09 CROSS=2 PRINT=0
.MEASURE TRAN I_RISE_R4 AVG I(R4) FROM=risestart_r4 TO=risestop_r4 PRINT=0
.MEASURE TRAN I_FALL_R4 AVG I(R4) FROM=fallstart_r4 TO=fallstop_r4 PRINT=0
.MEASURE I_EM_R4 PARAM=`abs(I_RISE_R4+I_FALL_R4)`
.TRAN 1P 20000e-12
.END
```

The above method is the one that is actually used in the flow as it is much more accurate, but for completeness's sake, the alternative method, according to which the rise and fall current density is calculated using the following formula is documented as well:

$$\frac{V_{op}C_{net}}{t_{transition}A} \quad (4.1)$$

In the above equation, V_{op} is the operating voltage, known from the standard-cell library for one of the characterized design corner of voltage and temperature, C_{net} is the output net's total capacitance that was retrieved at the previous step and $t_{\text{transition}}$ is the output transition time, calculated as it has been previously mentioned. Also, A is the interconnect's cross section, that equals the product of its thickness and its width, both derived from the standard-cell library's geometrical dimensions files (Library Exchange Format – LEF).

The thickness depends on the metal layer of the wire, so the result is calculated through a set of DBAccess calls. At first, the metal layer's number is retrieved using the *dbWireZ <wire pointer>* command. Then, this number is passed as an argument to the *dbGetLayerByZ* command which returns the layer's address, which in turn is used as an argument to the *dbGetLayerThickness* command. This command gives the layer's thickness in database units that are converted to micrometers by a multiplication with the result of the *dbHeadMicronPerDBU* command.

The width is calculated from the wire's wirebox data, which consists of the upper-right and bottom-left corners' coordinates on the chip and is retrievable from the Encounter database with the *dbWireBox <wire pointer>* command. The result of the *dbWireDir <wire pointer>* command is used to distinguish the length from the width. If its result is *dbcWireE* or *dbcWireW*, the wire's direction is horizontal and the difference between the y-coordinates equals the width, whereas if its result is *dbcWireN* or *dbcWireS*, the wire's direction is vertical and the difference between the x-coordinates equals the width. Again, the result is in DBU and must be converted to um as above.

Because the available EM model only supports DC current, the equivalent DC current density is estimated as the absolute difference of the rise and fall current densities that is $J_{equiv.} = |J_{rise} - J_{fall}|$.

<u>*EM parameters calculation*</u>:

The first parameter that needs to be estimated is the Blech length, whose formula is $\frac{jL_{critical}}{j_{normal}}$. If the length of the current wire is less than the Blech length, it is ignored because no EM occurs there. The rest of the EM model's parameters are calculated and reported to *deltaR.report* only for wires with lengths greater than that critical value. A counter is used to keep track of the number of each net's wires that are longer than the Blech length.

Example:

An example of the *deltaR.report* file's structure for one reported net is shown below:

```
####################################################################################
Net:             core0/leon3core0/ahbctrl0/n395          Temperature(C): 98.39
R   Blech_length(um)  t50(years)  R_slope(Ohm/year)  R_jump(Ohm)  J(A/um2)
56  16.4968152866     6.1991291   44.7638580112      491.507710   0.02242
98  9.98898071625     3.3952828   73.9275726026      491.507710   0.03704
####################################################################################
```

All the above steps, performed within the EM extraction script, are summarized in Figure 4.4, where their sequence towards the end results can be seen more clearly.

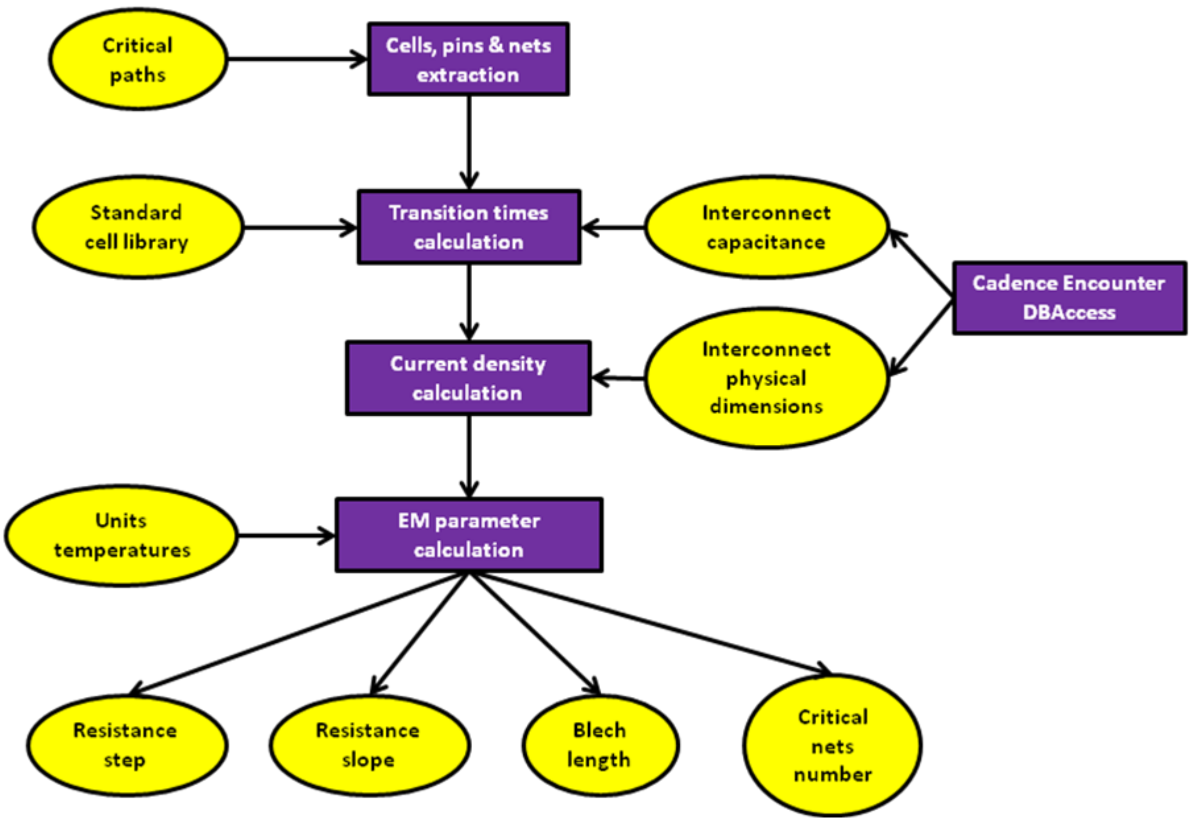


Figure 4.5. EM extraction script steps and results.

4.2.2 *SPEF update script*

This script is written in Perl and is responsible for the incorporation of the impact of EM to the design, so that it can be quantified by a timing analysis. This is achieved through the annotation of the parasitic resistances of interconnects in the SPEF file, according to predictions based on the EM model.

<u>*Inputs*</u>:

- The initial SPEF file, which does not include any wear-out impact yet.
- The *deltaR.report*

<u>*Outputs*</u>:

An updated SPEF file which includes the impact of EM on the resistance of the critical paths' interconnects after the desired amount of years of operation. After reading the parameters of a net from *deltaR.report*, the first action is to check if its t_{50} is exceeds the simulated time period. If it is, its wires must be updated through the following procedure:

<u>*Net lookup*</u>:

The nets that contain wires exceeding the Blech length and thus needing resistance update are read from *deltaR.report*. Their names are then looked up in the SPEF's name table which maps each name to a short code, so that the SPEF file is both easier readable and smaller in size.

<u>*Location of the net's wires parasitic resistances*</u>:

The parasitic resistances of each net's wires are located in a specific section of the SPEF file. In order to locate it, first *D_NET <net code> is searched for and then *RES. What follows are connections between *RC* distributed model nodes, together with each one's resistance value.

<u>*Detection and update of critical wires*</u>:

Wires that are longer than the Blech length can be detected from their resistance, because it is greater than the product of the Blech length and the average resistance per length value. This value is derived from the technology library as a mean value of all metal layers. When a wire with a resistance value that is greater than critical is encountered, its resistance is increased by the sum of:

- The wire resistance's step, ΔR.
- The resistance's slope, multiplied by the time difference between t_{50} and the end of the simulated time period.

The above steps must be repeated for every wire of the net until the *END keyword, that designates the end of the net's parasitic data, is read.

Once the process is completed for every net in *deltaR.report*, the impact of EM on the interconnect resistances of the critical paths' wires are imprinted in the new SPEF file.

4.2.3 EM impact on system's timing

The impact of EM can be estimated by a new timing analysis based on the annotated SPEF file using the Cadence Encounter Timing System (ETS). The steps in order to do this include reading the design's synthesized verilog netlist (*read_verilog <verilog filename>*), loading the necessary libraries (*read_lib <library filenames>*), creating a clock for the design (*create_clock <clk pin name> -name <clk name> -period <clk period>*). Then, in order to have timing reports that are uniformly produced and their comparison can give more accurate relative results on timing difference, a SDF file is generated based on the SPEF file. So, the SPEF file is read (*read_spef <spef filename>*), all delays are calculated through the *delayCal* method of Encounter's timing analysis engine and the SDF file is generated (*write_sdf –precision <significant digits> <sdf filename>*). Then, a report of the paths with the least timing slack (*report_timing –machine_readable –nworst <amount of paths>*) is produced, in order to compare it with the initial one and determine the delay shifting of the design's performance due to EM, after a certain amount of operating years.

4.3 TDDB flow

This paragraph covers the implementation details of the TDDB flow including the technical details and intermediate computations and steps that help reach to the final results. The TDDB flow's steps will be analyzed next, in order of execution. This flow consists of two discrete scripts, the extraction script and the update script as well as a one-time step to generate a lookup table, matching the leakage current to interconnect delay change.

4.3.1 TDDB extraction script

This script is written in Tcl and is called from within the SoC Encounter's environment, using the *source <script filename>* command, after loading the design to be analyzed. This allows the script to retrieve the layout-specific interconnect information from the Encounter database using DBAccess commands. The inputs and the outputs of the script performing the wire stack's extraction from the layout are summarized below:

<u>*Inputs*</u>:

- Standard-cell library
- SoC Encounter's design file (.enc)
- Critical paths file

<u>*Outputs*</u>:

The *wire.report*, which is the file containing all nets that are part of the critical paths examined, together with each net's wires and all adjacent wires of each of them. Each wire's coordinates, metal layer and geometrical characteristics as well as each adjacent wire's relative location (start and end coordinates and distance from the main wire) are reported.

Detection of potentially adjacent wires:

The goal is an initial retrieval of all wires that are near a wire that belongs to a net of one of the critical paths that are examined. The nets are available after parsing each line of the critical paths file, whereas the iteration over each wire is achieved with the *dbForEachNetWire <net pointer>* DBAccess command. For each one of these main wires, another DBAccess command is then used to get an initial list of all wires with x and y coordinates that place them near the main wire. The coordinates of a box that defines the maximum distance for a wire to be considered to be nearby must be determined first. Of course, the distance is technology-dependent. The command mentioned earlier is *findNetsInBox <box llx> <box lly> <box urx> <box ury>* and it returns the required list.

Filtering of really adjacent wires and classification:

The problem is that the nets returned by the above command are not necessarily adjacent, in the way that is needed for TDDB. There are several reasons for this:

- Only specific wires of these nets are actually within the given box.
- The two adjacent wires must run in parallel to each other with the same orientation and they must also overlap.
- The wires need to be on the same metal layer.

The following figure shows the main wire, the box that defines the maximum distance of its neighbors and some of the nets' wires that would be returned by the *findNetsInBox* command.

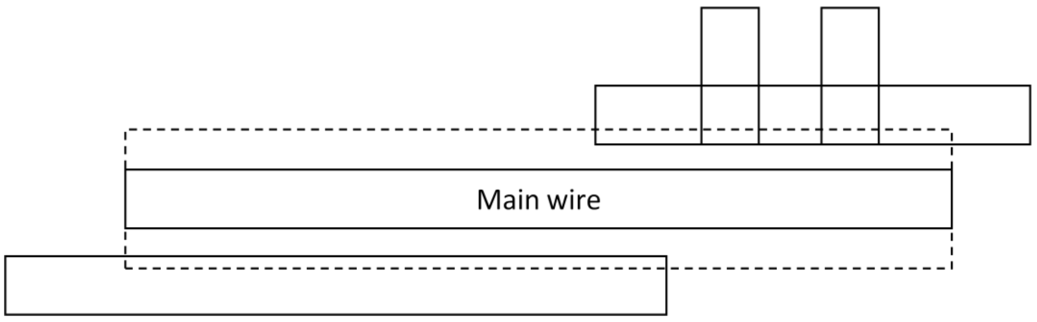


Figure 4.6. Adjacent wires - possible locations and filtering.

For informational purposes it is mentioned here, that a wire's metal layer can be determined through the *dbWireZ <wire pointer>* command and its orientation can be determined with the help of the *dbWireDir <wire pointer>* command.
After the filtering of all wires of all nets that are candidates using the above criteria, the wires need to be classified according to their relative position to the main wire in order to be reported. This is done by comparing the main wire's coordinates with the

adjacent wire’s coordinates. The four cases of adjacent wire overlapping are shown below:

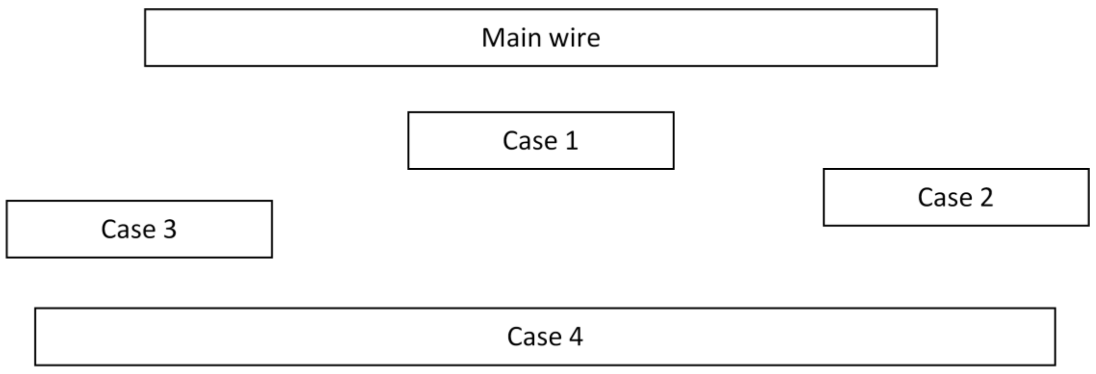


Figure 4.7. Possible locations of inter-metal adjacent wires.

Also, by using the orientation and the coordinates, we can find out if a wire is above, below, left or right of the main wire, as well as its distance from the main wire and the overlap length. All of these parameters, together with some other, describe the wires that are susceptible to TDDB and thus are reported to the *wire.report*, in order to be used later in the flow to estimate the evolution of TDDB over time.

Example:

An example of the structure of *wire.report* for one reported net is shown below:

```
############################## Wire detailed report ##############################
##################################################################################
Net:            core0/leon3core0/leon3s0_1/p0/iu0/n5750
Input pin:      core0/leon3core0/leon3s0_1/p0/iu0/r_reg_X__DATA__0__5_/D
Output pin:     core0/leon3core0/leon3s0_1/p0/iu0/U450/ZN
Total Capacitance(pf): 0.00263318
Total Length(um):      9.545
-------------------------------------------------------------------
Wire:   0x22dcf6f0     Layer   metal2
Direction: dbcWireN
Length: 1.4     um      Thickness      0.14    um      Width   0.0700000000002        um
Location:      (1516.935, 1228.745)   (1517.005, 1230.215)   (Unit:  um)
Via:    VIA12_1cut_V
****** left ******
Unit:   um
Start   End     Distance       Wire
1197.665       1267.875       0.0699999999999       0x230b0834
-------------------------------------------------------------------
Wire:   0x22dcf738     Layer   metal3
Direction: dbcWireE
Length: 7.42    um      Thickness      0.14    um      Width   0.0699999999999        um
Location:      (1516.935, 1230.145)   (1524.425, 1230.215)   (Unit:  um)
Via:    VIA23_1cut
****** above ******
Unit:   um
Start   End     Distance       Wire
1524.215       1525.685       0.21    0x22e19530
1478.855       1766.765       0.35    0x229d1dbc
1473.115       1546.825       0.0700000000002       0x229c95cc
****** below ******
Unit:   um
Start   End     Distance       Wire
1508.255       1519.945       0.0699999999999       0x22bc6b88
1523.515       1531.145       0.0699999999999       0x22ae4bb4
1523.375       1526.525       0.21    0x22eaa270
```

4.3.2 *TDDB delay library*

This step is not executed every time in the flow, but only once for each technology node. The delay library that it generates is used instead in every run to accelerate the execution of the flow.

Supposing that it is known, that TDDB occurs between two adjacent wires, a way must be found to estimate the delay that the TDDB-induced leakage current inserts to the system. An accurate prediction can only be done by a Spice simulation using a RC model. The required simulations could be executed on-the-fly using real wire patterns that are extracted from the design. This would provide maximum accuracy but the execution-delay overhead it would introduce in the flow, presents a serious problem. So, another approach was chosen instead. This included the generation of a delay library only once for each technology, that would estimate the delay based on characteristic cases that cover all possible wire patterns.

Simulation script generation:

The simulations whose results are needed to be included in the library are automatically generated through a Perl script. The script needs some technology parameters as inputs. These are the thickness of the metal layers, the dielectric constant of the dielectric material and the average resistance and capacitance per unit length. A RC-equivalent of each wire pattern case to be simulated is generated as follows. Ten stages are used for the RC model. The wire pattern parameters are:

- The wire's length.
- The adjacent wire's overlapping length.
- The offset that defines the relative position of the two wires. Only cases where the lengths and offset result in overlap are examined.
- The distance between the two wires.
- The leakage current between the two wires.

The resistance and capacitance are calculated as wire length multiplied by the average resistance or capacitance per unit length respectively and are distributed evenly across the *RC* model. An extra capacitance, existing because of the capacitor formed by the surfaces of the two wires, is distributed evenly across the part of the wire that overlaps with the adjacent one according to the offset and the overlap length values. This extra capacitance is calculated as follows:

$$C_{extra} = \frac{\varepsilon * \varepsilon_0 * thickness * length}{distance} \quad (4.2)$$

Along the overlapping region, current sources are also evenly distributed and their total value equals the leakage current between the two wires. At the end of the wire, a capacitance which represents the load is added. Each simulation's transient report is written to a separate file, while a shell script executes the simulations in HSPICE [20].

Example:

An example of a *RC* model for a specific wire pattern is shown in the following figure

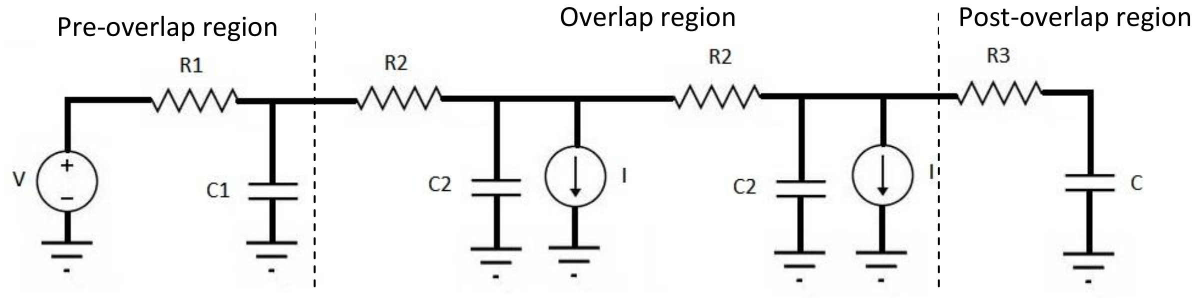


Figure 4.8. Inter-metal leakage current RC model.

In the above figure, *V* is the voltage applied to the start of the wire, *R*1, *R*2 and R3 are the distributed resistance values of the wire sections in each region, *C*1, *C*2 and *C*3 are the distributed capacitance values of the wire sections in each region (*C*2 also contains the inter-metal adjacent wire capacitance), *I* is the distributed leakage current value and *C* is the driven load capacitance.

Library generation script:

After the completion of the simulations, the results are read and the delay change library that matches each set of (wire length, adjacent wire overlap length, offset, leakage current, distance) values to the TDDB-induced delay change ratio is generated. First, the delay when there is no leakage current is read for each set of values to be used later as reference. Next, for the rest of the sets that differ only in the leakage current value, the delay change ratio is calculated and is written to the library along with the set's values. The above steps are repeated in a *Perl* script for all simulations, until all value sets are covered in the library.

Example:

An example of a part of the TDDB delay library showing the increase of the delay change ratio with the increase of the leakage current in a specific wire pattern, follows:

```
Length(um) Length(neighbor um) start_point(um) Leakage(uA) distance(um):
delay_change_ratio
200 30 0 0 0.06 : 0
200 30 0 0.25 0.06 : 0.000819672131147554
200 30 0 0.5 0.06 : 0.000819672131147554
200 30 0 0.75 0.06 : 0.00163934426229511
200 30 0 1 0.06 : 0.00163934426229511
200 30 0 2.5 0.06 : 0.00409836065573777
200 30 0 5 0.06 : 0.00737704918032785
200 30 0 10 0.06 : 0.0147540983606558
200 30 0 15 0.06 : 0.0221311475409837
200 30 0 20 0.06 : 0.0295081967213116
200 30 0 25 0.06 : 0.0368852459016394
200 30 0 30 0.06 : 0.0442622950819673
200 30 0 40 0.06 : 0.059016393442623
200 30 0 50 0.06 : 0.0737704918032787
```

Using the TDDB delay library, the delay change ratio of a wire pair with given geometrical characteristics and leakage current value, can be looked up both easily and quickly. Interpolation is used to estimate the delay change ratio for intermediate wire patterns.

4.3.3 SDF update script

This script is written in Matlab mainly because of the integrated look-up function in multidimensional lookup tables that it offers. It is responsible for the estimation and incorporation of the impact of TDDB to the design, so that it can be quantified by a timing analysis. This is achieved through the annotation of the delays of interconnects in the SDF file, according to predictions based on the TDDB model.

Inputs:

- The initial SDF file of the target design.
- The temperature trace file of the hierarchical floorplan units.
- The *wire.report* file
- The TDDB delay library
- TDDB leakage measurement files under stress conditions

Outputs:

An updated SDF file, which includes the impact of TDDB on the delay of the examined paths' interconnects due to inter-metal dielectric leakage current, after the desired amount of years of operation.

Leakage current estimation:

The leakage current is estimated as it was described in the TDDB model section, by performing extrapolation of the experimental data, gathered under stress conditions, to operating conditions, while considering a much longer operation time. First, the experimental data of leakage current increase over time is read and entered in a single look-up table in the corresponding positions, according to the temperature and voltage under which the respective experiment was conducted. Next, these are used to estimate the actual leakage current after the given time period by performing a linear extrapolation as described previously. For added accuracy, the temperature that is used, is the one of the specific unit, read from the temperature trace file after parsing the net's hierarchical name in order to determine the corresponding unit. Finally, as the leakage current is directly proportional to the surface of the wires that are adjacent, the result of the extrapolation is divided by the cross section of the experimental wire and multiplied by the cross section of the actual wire.

Wire delay change ratio estimation:

Now, that all required values for a lookup in the TDDB delay library are known and especially the leakage current that is derived from extrapolation and is not included in *wire.report*, as all of the other wire pattern parameters, a look-up can be performed. The result is of course the delay change ratio of a single wire, which belongs to the specific interconnect.

Interconnect delay change estimation:

The delay of a wire can be assumed - without significant error - to be proportional to its length, considering a wire delay model similar to the Elmore delay approach [33]. Thus, in order to estimate the delay change ratio of an interconnect from the delay change ratios of its wires, a weighted average value is calculated. The weight factor for each wire is the fraction of its length to the total interconnect's length. Once the delay change ratio for an interconnect is known, the input and output pin that define it are read from the *wire.report* and the interconnect is looked up in the SDF file. When found, the rise and fall transition times are read, increased by the delay change ratio that was calculated above and then written back to the SDF file.

Once the above steps have been executed for every interconnect included in the *wire.report*, the SDF file contains the delays that the path's interconnects will have, after sustaining TDDB for the specified period of time. If the SDF file, that has already been updated, contained also the impact of EM on the system's timing, then the total timing impact of both EM and TDDB on the examined paths of the design would have been incorporated into the final SDF file.

The steps leading from the wires' geometrical characteristics contained in the *wire.report*, towards the estimation and annotation of the TDDB impact to the SDF file are displayed graphically in Figure 4.8, to provide an overview of the steps' sequence followed.

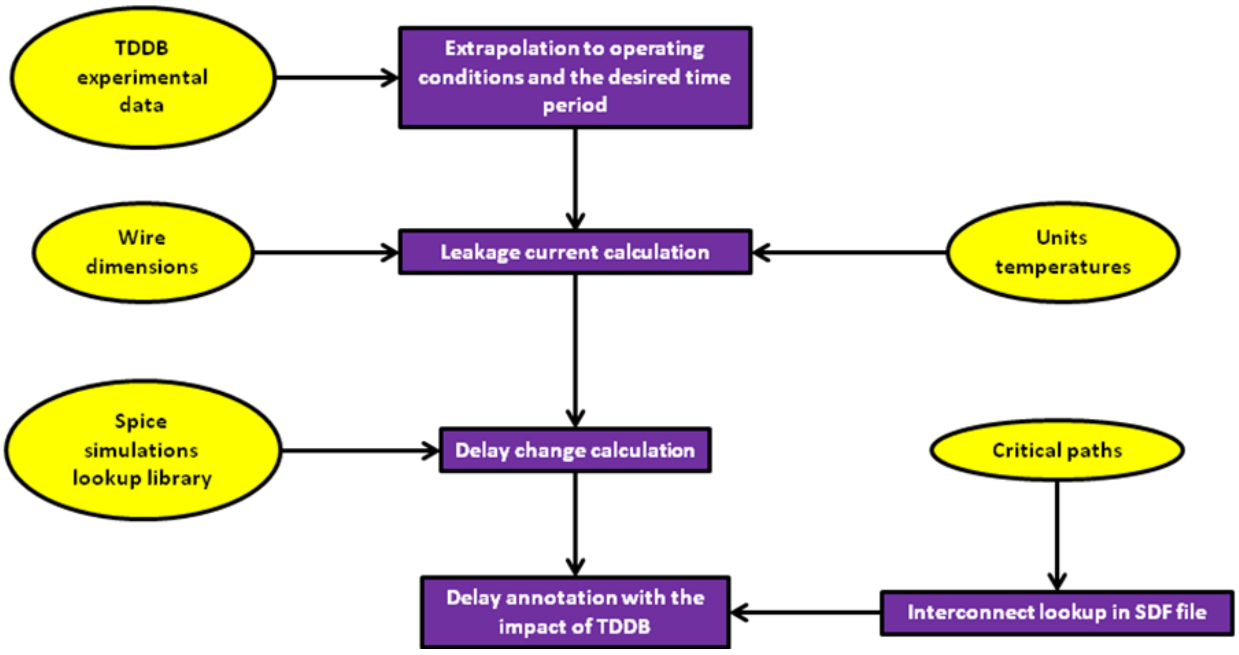

TDDB experimental data
Extrapolation to operating conditions and the desired time period
Wire dimensions
Leakage current calculation
Units temperatures
Spice simulations lookup library
Delay change calculation
Critical paths
Delay annotation with the impact of TDDB
Interconnect lookup in SDF file

Figure 4.9. Inter-metal leakage current extrapolation, delay estimation and impact's annotation.

4.3.4 TDDB impact on system's timing

The impact of TDDB can be estimated by a new static timing analysis, based on the annotated SDF file and through the Cadence ETS timing engine. The steps required for the final timing impact's evaluation in order to do this include the reading the design's synthesized Verilog netlist (*read_verilog* <*verilog filename*>), the loading of the necessary standard-cell libraries (*read_lib* <*library filenames*>) and the definition of/creating a clock for the design (*create_clock* <*clk pin name*> *-name* <*clk name*> *-period* <*clk period*>). Then the updated SDF file with the new point-to-point delays in the examined path is read (*read_sdf* <*sdf filename*>) and a report of the paths with the least slack (*report_timing –machine_readable –nworst* <*amount of paths*>) is produced, in order to compare it with the initial and determine the timing impact of TDDB on the design.

It should be noted that if the post-EM SDF file is used as the initial SDF file in the TDDB impact estimation process and it is already annotated before the TDDB delay overhead's computation, the updated final SDF file would contain the combined impact of both reliability phenomena.

4.4 Automation of the flow

4.4.1 Summary of flow steps and of implementing scripts

To summarize, the reliability analysis flow, excluding the TDDB delay library generation process which is simple and only needs to be done once for every new fabrication technology used, is comprised of the following scripts:

- **format_paths.tcl**
  It converts the format of the paths reported by the timing analysis or any other path selection method or tool to a format that the scripts that implement the flow can process.
- **extraction_temp_multipath.tcl**
  It implements the core of the flow which extracts all useful information regarding EM and TDDB from the layout and stores them to *deltaR.report* and *wire.report* respectively.
- **spef_update.pl:**
  It incorporates the impact of EM which is found in *deltaR.report* to the SPEF file of the design, so that a new timing analysis can reveal the impact on the design's timing.
- **sdf_update.m:**
  It estimates the impact of TDDB on system's timing through the *wire.report* and the TDDB delay library and updates the SDF file with the new interconnect delays, so that a new timing analysis can evaluate the impact on the system's performance over time.

In order to execute the whole flow *automatically*, the user should first produce the TDDB delay library. Because it is a one-time process and only utilizes two short scripts and a simple configuration file for the technology parameters, no automation script has been created for that. But apart from that, because of the complexity of the flow, its execution has been automated for any design that uses compatible EDA tools and requires only the presence of the corresponding TDDB delay library, an activity file of the design (*.vcd* or *.saif*) and the SoC Encounter (*.enc*) file that contains the design with the placement and routing information. Any other file required for the execution of the proposed reliability flow is automatically generated from the two Unix shell scripts, which are the basis of the automated tool flow.

4.4.2 Temperature estimation script

The first script is called *temp_flow.sh* and it only needs to be executed every time the user wants to estimate the temperature of the hierarchical units of a new design or update the temperatures of the current design based on power results produced by a different application.
In order to make the full automation of this process possible, two additional scripts needed to be written, in order to convert floorplan and power data in formats compatible with the tools which will take this information as inputs. The functionality of these scripts is analyzed below:

- **floorplan_converter.tcl**
  This script is used to convert the floorplan derived from SoC Encounter to the format that is required by HotSpot. This involves the parsing of the floorplan file (*.fp*) to detect all lines beginning with "*Guide*:", which contain the hierarchical units of the design. Then the llx, lly and urx, ury coordinates that are read must be converted from micrometers to meters and to a llx, lly and width, height format and finally written to a *.flp* file.

- **power_analysis.tcl**
  This is used to convert the power data derived from SoC Encounter with the help of an application generated activity file to the format that is required by HotSpot. This involves parsing the power report to detect all lines beginning with "average power", which contain the power consumption for each hierarchical unit in the design. Then the power needs to be converted from whatever units it is to Watts and written together with the unit name to a power trace file compatible with HotSpot.

After the above clarifications, the steps followed in the automated temperature estimation script can be explained:

<u>*Floorplan file generation*</u>:
After restoring the design in the SoC Encounter environment, the *saveFPlan <floorplan file name>* command is executed and then the *floorplan_converter.tcl* script is called to convert it to the format required by HotSpot.

<u>*Power trace file generation*</u>:
After restoring the design in the SoC Encounter environment, the extraction of the parasitic resistances and capacitances of the design is performed using the *extractRC* command. Then, the *probePower <hierarchical units>* command is used to define which hierarchical units' power needs to be calculated. The space separated

hierarchical units' names are retrieved from the .fp file generated by SoC Encounter earlier through the following *awk* program:

```
`awk '{if ($1=="Guide:") printf("%s ",$2)}' <floorplan file name>`
```

Then, a power analysis is performed using the command *updatePower -vcd <vcd filename> -vcdTop <vcd top design name> -noRailAnalysis -report <power report filename> VDD*. Finally, the power_analysis.tcl script is called to convert the power trace file to the format required by HotSpot.

Temperature trace file generation:

The temperature of each hierarchical unit is estimated by providing HotSpot with the correct floorplan and power trace files. A second execution with the steady-state temperatures of the first execution is required, as a second iteration ensures a more accurate temperature estimation for each unit.

The shell commands contained in the script to call HotSpot are the following:

```
./hotspot -c hotspot.config -f <floorplan filename> -p <power trace
filename> -o unit_temps.ttrace -steady_file <steady temperatures
filename>

cp <steady temperatures filename> <initial temperatures filename>

./hotspot -c hotspot.config -init <initial temperatures filename> -f
<floorplan filename> -p <power trace filename> -o unit_temps.ttrace
```

4.4.3 Flow configuration file

At this point and before continuing with the second Unix shell script that executes the core of the flow and needs to be executed every time a new path needs to be analyzed or a change in the design has been made, in order to estimate the impact of EM and TDDB on system timing, a configuration file that contains all the environment variables that are used by the Unix shell scripts throughout the tool flow needs to be presented. This configuration file contains both filenames and required information that are design-specific and need to be set for every new design. The name of the configuration file is *flow_conf.sh* and it is imported using the *source* command at the beginning of the two Unix shell scripts of the flow, in order to set the environment variables to the values corresponding to the current design. The variables that the configuration shell script contains are listed and explained below.

- **design_name**
  This is the name of the design to be analyzed. The user can provide any name, as this name is only used as a prefix to many intermediate files that are invisible to the user.

- **wire_report**
  This is the name of the *wire.report* file that is generated by the extraction script and contains all the adjacent wires for TDDB. The filename should not

be changed by inexperienced users of the tool flow, as if it is, it also needs to be changed in the extraction and SDF update scripts.

- **deltaR_report**
  This is the name of the *deltaR.report* file that is generated by the extraction script and contains all the data for the wires in which EM occurs. The filename should not be changed by inexperienced users of the tool flow, as if it is, it also needs to be changed in the extraction and SPEF update scripts.

- **initial_timing**
  The name of the file containing the initial timing analysis in order to compare it with subsequent timing analyses, in order to observe how the degradation of the system over time has affected its timing and how timing criticality is shifted from some paths to others, because of uneven wear.

- **em_timing**
  The name of the file that includes the timing analysis report, extracted after the annotation of the updated resistances due to EM to the design.

- **tddb_timing**
  The name of the file in which the timing reports after the annotation of the TDDB delay's impact to the design.

- **critical_path**
  This is the name of the *critical_paths.txt* file that contains the critical paths that will be analyzed by the flow for reliability issues. The filename should not be changed by inexperienced users of the tool flow, as if it is, it also needs to be changed in the extraction and format paths scripts.

- **initial_spef**
  The filename of the initial SPEF file of the design.

- **em_spef**
  The filename of the annotated SPEF file that contains the updated resistances, which represent the impact of EM on the examined path's interconnects.

- **initial_sdf**
  The filename of the initial SDF file of the design.

- **em_sdf**
  The filename of the SDF file that contains the impact of EM on the delay of the interconnects that belong to the investigated critical paths.

- **tddb_sdf**
  The filename of the SDF file that contains the combined impact of EM and TDDB on the delay of the interconnects that are part of the examined critical paths.

- **libs**
  This variable holds the space separated filenames of all libraries that are used within the design. These include the standard-cell library used, as well as any additional libraries, such as these of memories or of other components.

- **verilog**
  The filename of the design's post-layout Verilog netlist.

- **top_module**
  The name of the top module of the design, as it is used in SoC Encounter.

- **sdc**
  The *.sdc* file, containing any design constraints that are needed during the post-layout static timing analysis.

- **enc_dat**
  The filename of the SoC Encounter design file, including all the information about the design's placement, clock tree and routing (*.enc.dat*).

- **library**
  The name of the standard-cell library.

- **vcd**
  The filename of the *.vcd* file containing the application-specific switching activity of the design.

- **vcdtop**
  The name of the top module of the design, as it is written in the *.vcd* file.

- **clk_name**
  The name of the clock's input pin in the design.

- **clk_period**
  The desired clock period at which the design should operate, thereby comprising its main performance constraint.

- **paths**
  The number of the most timing-critical paths to be generated by the timing analysis engine of the Encounter's timing system. The interconnects of these paths will be later extracted, in order to estimate the impact of EM and TDDB on their delay and consequently on the timing of the specific paths, in which the underlying interconnects belong. The user can experiment with the number of paths to be examined, depending on the tradeoff of searching for the paths with the maximum possible timing impact due to EM or TDDB (or both), versus the execution time, which increases with the number of paths tested.

4.4.4 Flow core script

The description of the second Unix shell script that executes the core of the flow and needs to be executed every time a new path needs to be analyzed or a change in the design has been made, in order to estimate the impact of EM and TDDB on the system's timing, is demonstrated in this section. The steps followed will be described in a high level for the most part, as the majority of the command sequences required for the generation of a SPEF or an SDF have already been covered previously.

First of all, a cleanup of all files from the previous execution of the flow is performed, as some information are appended to files for code simplicity's sake instead of creating a new file and that would mix up data from different iterations of the flow.

The existence of each of the remaining files, such as the initial SPEF file, is checked through the *–s <filename>* Unix command, so that certain files are only created during the first run, thus saving time from any subsequent executions. The same command is used after each step, to ensure that every necessary file has been created and it is not empty. If a file has not been created at the point it should, the execution of the tool flow is interrupted and a diagnostic error message is displayed. Thus, the detailed steps included in the Unix shell scripts are described as follows:

- Generation of the initial SPEF file through SoC Encounter, if it does not exist.
- Generation of the synthesized Verilog netlist file through SoC Encounter, if it does not exist.
- Generation of the initial SDF file through the Encounter Timing System, if it does not exist.
- Generation of the initial timing report containing the specified number of most timing critical paths through the Encounter Timing System.
- Format conversion of the critical paths using the format_path.tcl script.
- Extraction of the EM and TDDB related data and storage in *deltaR.report* and *wire.report* respectively, through the *extraction_temp_multipath.tcl* script, running from within SoC Encounter's environment.
- Annotation of the SPEF file with the elevated wire resistance caused by EM, using the *spef_update.tcl* script.
- Generation of the SDF file with the EM impact through the Encounter Timing System, after loading the updated SPEF file.
- Generation of the EM timing report containing the specified number of most timing-critical paths through the Encounter Timing System, after loading the SDF file including the EM impact's annotation.
- Calculation of the delay overhead for each interconnect in the examined timing path, due to TDDB, and annotation of the EM-aware SDF file with the new, increased delays, through a Matlab script.
- Generation of the timing report after having annotated the delay impact of TDDB, containing the specified number of most timing critical paths through the Encounter Timing System after loading the SDF file with the delay overhead due to TDDB, in order to estimate the combined effect of both reliability wear-out on the system's performance over time and, of course, for the desired lifetime.

Consequently, given the necessary prerequisite files for the execution of the flow, the user can then use a variety of numbers for the critical paths to be examined, as well as different switching activity profiles derived from the post-layout netlist's simulation, thereby giving an alternative *.vcd* file. The next section provides an insight into the approximate execution time of the tool flow.

4.5 Execution time of the flow

A tool is only useful, as long as it can deliver its results in a reasonable amount of time. Therefore, attempts have been made to keep the execution time of our tool flow fast. These attempts include, but are not limited to the use of pre-built look-up table libraries, instead of performing on-the-fly simulations and calculations, wherever possible. The following table lists some sample times required for the execution of each one of the core components of the framework for the *50 most timing-critical paths* of the target design, some of which share a number of interconnects. It must be noted that the non-dedicated, multi-user host machine was operating under heavy load at the time these measurements were taken.

Table 4.1. Approximate execution time of the flow's core components for 50 paths.

Framework component	Approximate execution time (in minutes)
Extraction of net transition times for the EM (Encounter DBAccess, Tcl script)	< 1
Extraction of adjacent wires for the TDDB (Encounter DBAccess, Tcl script)	17
Current density and EM parameters calculation (including Spice simulations in HSPICE)	11
SPEF annotation (Perl script)	< 1
TDDB model calculations and SDF annotation (Matlab)	14

The time required for the static timing analyses before and after the examined phenomenon's impact annotation is not listed here, as these tasks are "outside" of the framework's core and also depend on the specific static timing analysis engine used. The total measured time for the execution of the framework considering the *50 most timing-critical paths*, using the Leon3-based MP-SoC layout with timing-driven placement, was approximately *50 minutes*, including the initial parasitic extraction, the static timing analyses and the generation of the prerequisite SPEF, SDF and Verilog files of the design. Consequently, the execution time of the framework can be roughly estimated at about 1 *minute per path*, which is a reasonable amount of time, considering the overhead of the on-the-fly Spice simulations and of the prerequisite files generation.

5

Design platforms and implementation flows

5.1 Digital implementation flow of design platforms

In the two previous chapters, we elaborated in detail on the proposed interconnect reliability flow, by presenting the motivation for the deployment of this work, as well as the discrete steps of the timing estimation framework regarding the two studied reliability wear-out mechanisms. We also analyzed the technical aspects of the design flow's implementation, considering the existence of a standard-cell based design, which is used as the platform for the extraction of the target interconnects and the annotation of the EM and TDDB impact on the affected wires.

In this chapter, we will elaborate further on the implementation of the designs used as testbenches for the conduction of the experiments during the development of the presented thesis. These designs are generated based on an RTL-to-GDSII flow, which is a top-down methodology for the implementation of physical designs, starting from behavioral or structural descriptions of digital systems at the register-transfer level (RTL), written into a certain HDL language, either Verilog or VHDL. The underlying design flow is a standard in the Application-Specific Integrated Circuit

(ASIC) Electronic Design Automation (EDA) industry, as it enables the ASIC designers to work at any level of abstraction to implement their designs at a certain design layer, while allowing them to export the produced design of the current layer to other EDA tools, and work at lower or higher abstraction layers. Hence, apart from a top-down execution of the flow, which starts from an RTL description and finishes with a physical design, exported as a GDSII layout database, there are several other formats available, so that the designers can work either in a top-down or in a bottom-up manner.

Later in this chapter, we will present the design flow followed for the generation of the layouts used in the conducted experiments. During this thesis, our main design platform was a Multi-Processor design, based on the Leon3 SPARC [32] core, which was included in the package of the Gaisler Research toolsuite [23]. The motivation for the selection of the Leon3-based platform was focused on several advantages, among which we can distinguish the configurability and the flexibility of the design's description, based on parameters and packages that permitted the usage of different kinds of components and peripherals, thereby giving the designer adequate control over the complexity of the system.

Moreover, Gaisler's toolsuite also provided support regarding the programmability of the system, by including the required software for the development of applications and the proper interfaces for their simulation in either the RTL or in the post-layout design layer. Additionally, the underlying platform was free-of-charge and there was significant experience regarding the VHDL code and the software tools, as well as background from another thesis working on this platform. However, in the conducted experiments, we have also included smaller, but of also significant complexity ASIC designs, which have been used as alternative platforms in order either to enrich the current thesis from the aspect of quantity of results, or to conduct experiments on smaller designs in order to evaluate the proposed flow in different design classes and therefore to acquire a more holistic view regarding the flow and the assumptions that came along with its implementation.

In the following section, we will present the RTL-to-GDSII ASIC design flow, used for the generation of the Leon3-based physical designs, by describing the most important steps at the synthesis and place-and-route design layer. Apart from the implementation steps themselves, the next section also sheds light on the different timing, power and area formats of the ASIC libraries used at the synthesis and place-and-route stages, as the standard-cell and memory library formats are coming along with the layer of design abstraction, concerning the detail and the accuracy of the information. Moreover, in the same section, we will present the static timing analysis and power estimation tools mentioned in the two previous chapters, as well as the various options that the used EDA tools provide to the designer, during the implementation of such complex designs. Finally, in the remaining sections, we will present the flow's steps for the development of the applications used for power analysis, as well as a way of performing functional simulation of either the RTL or of the post-layout netlists of Leon3-based MP-SoCs.

5.2 Synthesis and Physical Implementation flow

In this section, we will elaborate on the analysis and the implementation strategy followed for the transformation of an RTL description to a physical design, comprised by cells of a characterized standard-cell library, in three different design corners, namely typical, fast and slow. The standard-cell libraries form the basis of the ASIC design flow, as they provide the primitive components a synthesis tool is based on, in order to transform a design written in VHDL or Verilog, to generate an intermediate gate-level circuit or netlist, which is the equivalent term in the ASIC industry, and to map this netlist, which is initially composed of technology-independent gates, to the technology-aware cells of the incorporated library.

Usually, these libraries comprise of common, typical cells, like inverters, NAND and NOR gates, multiplexers and tri-state buffers, which are the necessary combinational cells, while they should also include at least one D flip-flop cell and a latch, for synchronous and asynchronous design styles. It must be noted that in our designs, as well as in most common RTL-to-GDSII flows, the logic cells are implemented based on the CMOS design style and they follow certain rules regarding their boundaries and their internal structure, as well as their geometry.

In perspective, most of the state-of-the-art standard-cell libraries do not include logic cells of bi-directional inputs such as transmission gates or separate PMOS or NMOS transistors, as long as the characterization of these cells requires analog rather than digital simulations, while the verification of the design’s operation cannot be fulfilled in the context of a totally digital standard-cell toolflow. Moreover, the geometric dimensions of the cells and especially their height should be the same, so that the place-and-route process is facilitated, as the core area is divided into rows where the cells are to be placed.

Based on the technological constraints imposed by the available standard-cell library, the designers are likely to select the proper implementation strategy, depending on the design’s complexity and the performance constraints that their design should achieve, after the physical implementation’s completion. As optimizations at higher design layers like gate-level synthesis are likely to show greater impact that their post-layout counterparts, the synthesis process is a critical design stage, at which the designer should select both libraries and strategy of implementation, in order to achieve the desirable performance in terms of delay, area and power. In the following sub-section, we present the synthesis steps followed for the implementation of the Leon3-based MP-SoC design, while an overview of both Gaisler’s RTL design platform and the Synopsys Design Compiler [35] synthesis tool is presented.

5.2.1 The Leon3 design platform

In general, the synthesis of such a complex design like the MP-SoC Leon3-based platform used in the presented thesis requires decent knowledge of the target design’s components and consequently, of the possibilities offered by its configurability and flexibility. These are the two main advantages of Gaisler’s Leon3 distribution, along with the fact that the VHDL code is available through the Internet and can be downloaded from Gaisler’s website [23]. Another significant advantage of the specific RTL code is that the designer can configure the number and also the sort of the components included into the system as masters or slaves (peripherals), as well as their functionality by either setting or unsetting specific parameter values included in a certain configuration package, or even through a graphical user interface. For example, the internal functionality of the Leon3 processor’s pipeline can be configured depending on selections that deal with the implementation of the Arithmetic and Logic Unit, by selecting to add or not, multipliers, dividers or floating-point operation modules. Apart from the Leon3 processor itself and the number of cores that may be used in the design, we can also configure the functionality of the AMBA bus controllers, which are also provided by Gaisler's suite, as well as the peripherals of the target system and the interface of the design to the external memory. A schematic diagram that illustrates all of the aforementioned components and their connectivity with each other is provided in the next figure, which demonstrates the way the Leon3 processor core of a single-core system is attached to the AMBA system bus, along with the other components that may operate either as masters (e.g. the Memory and the Interrupt request controller), or as slaves (e.g. peripherals like Universal Asynchronous Receiver Transmitter or General-Purpose I/O modules). The processor cores, along with the Memory and the interrupt request controllers, are connected to the high – performance bus, namely the AHB, while the peripherals communicate with the masters as slaves through the APB bus and the AHB-to-APB bridge. For an MP-SoC system with two or more cores attached to the AHB, the presented diagram changes slightly, by adding one or more additional boxes labeled as "Leon3 Processor" on the AHB Controller. We will elaborate further in the next paragraph on the configuration of such a Leon3-based system, as the way of selecting important design parameters like the number of cores used, the size and the sets of the internal data and instruction caches or the peripherals to be used, comprise the basis for the development and the implementation of the target system, as the parameters of the *config.vhd* package control both the usage and the functionality of the processor and also of the supplementary modules of the design.

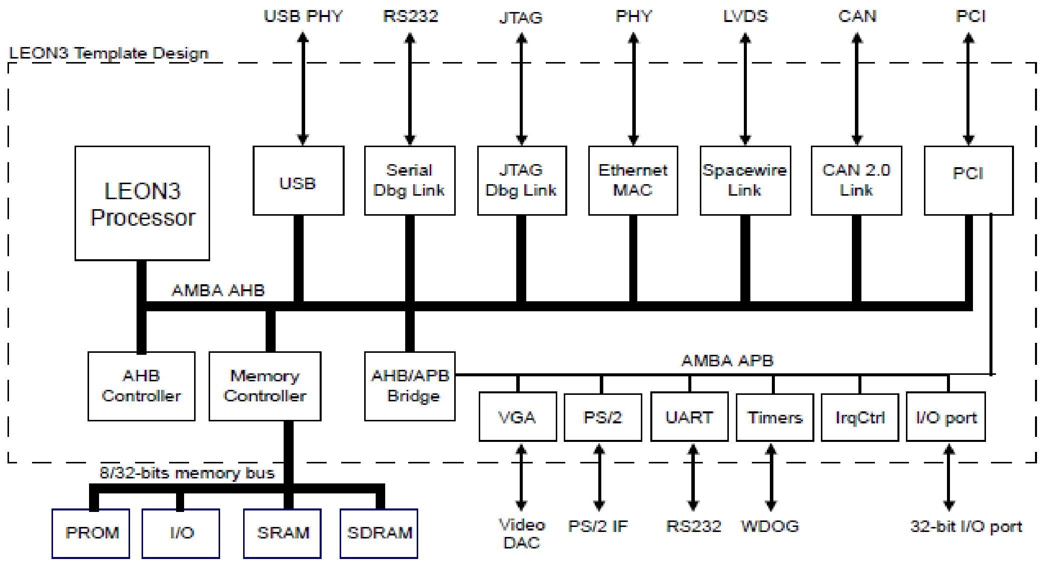


Figure 5.1.A Leon3-based SoC with the core and peripherals on the AMBA bus.

In perspective, the aforementioned package, through which the designer can configure the components included into the target RTL design, contains all the critical parameters regarding the processor's configuration, the internal data and instruction caches, the memory controllers available and also the functionality of the AMBA bus, either for the high performance modules (Advanced High-Performance Bus – AHB) or for the attached peripherals of the system. The configuration of the system can be performed through the use of a graphical user interface executed through automated Make scripts, provided in Gaisler's toolsuite, along with the RTL code.

The makefiles along with the VHDL files of the top hierarchical modules, the testbenches and other scripts, which correspond to either the synthesis or the simulation stages, and on which we will elaborate further below, are all together included into the specific folder that corresponds to the implementation platform of the Leon3-based MP-SoC design. This platform may be either a FPGA architecture of various vendors, like *Xilinx* or *Alterra*, or a standard-cell library, which is suitable for an ASIC RTL-to-GDSII flow. Hence, depending on the platform and based on the provided makefiles, the configuration of the design's parameters, considering either a FPGA architecture or an ASIC standard-cell library as the implementation's platform, is performed through the following command:

```
$ make xconfig
```

The execution of the specific command requires the installation of the GNU Make program first. Hence, after the 'xconfig' part of the makefile has started to be executed, an option pane with various tabs and corresponding design options is prompted, as it can be seen in the following screenshot.

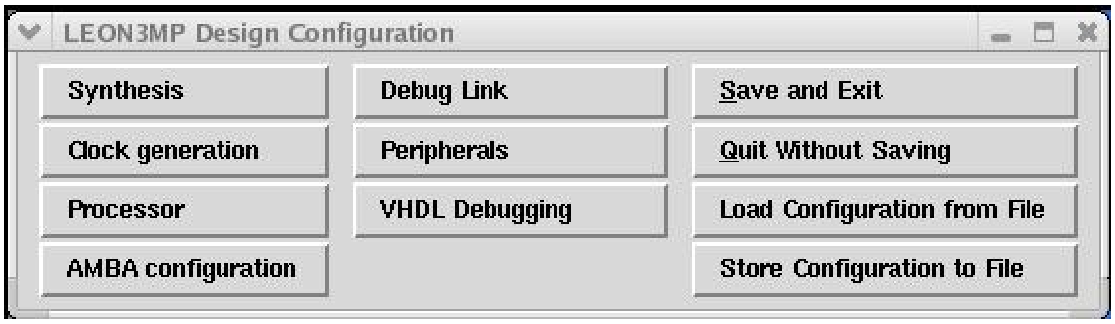
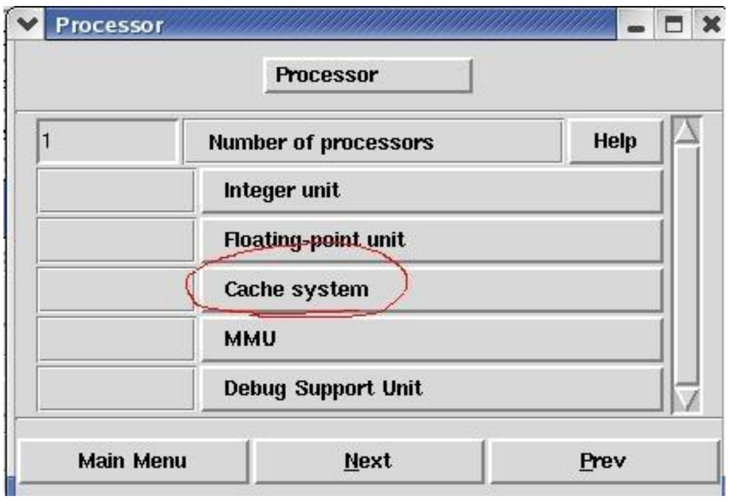


Figure 5.2.The GUI for the configuration of the Leon3 MP-SoC platform.

In Figure 5.2, each of the illustrated buttons correspond to options that the designer should take into consideration, in order to configure the design with the desired parameters and characteristics. It can be seen that there are options regarding the Leon3 processor core's configuration, as it is shown in Figure 5.3, where also the size of the internal data and instruction caches can be selected, according to Figure 5.4. Moreover, the designer can control the functionality of the AMBA bus, while also choosing to add or not specific peripheral components. Diverse selections are also offered regarding the synthesis strategy, as well as the phase-locked loops that may be possibly used for the generation of clocks with higher or lower frequencies. Finally, after having selected the desired options, depending on the application, the designer can store this configuration to *config.vhd*, which will be used for the elaboration of the design during the RTL simulation and the synthesis stage. The values of the system's options set through this GUI are also dumped to *config.vhd* along with their selected values and they are used in most of the design's VHDL files as generic parameters, making the RTL code configurable and thereby offering great flexibility to the designer.

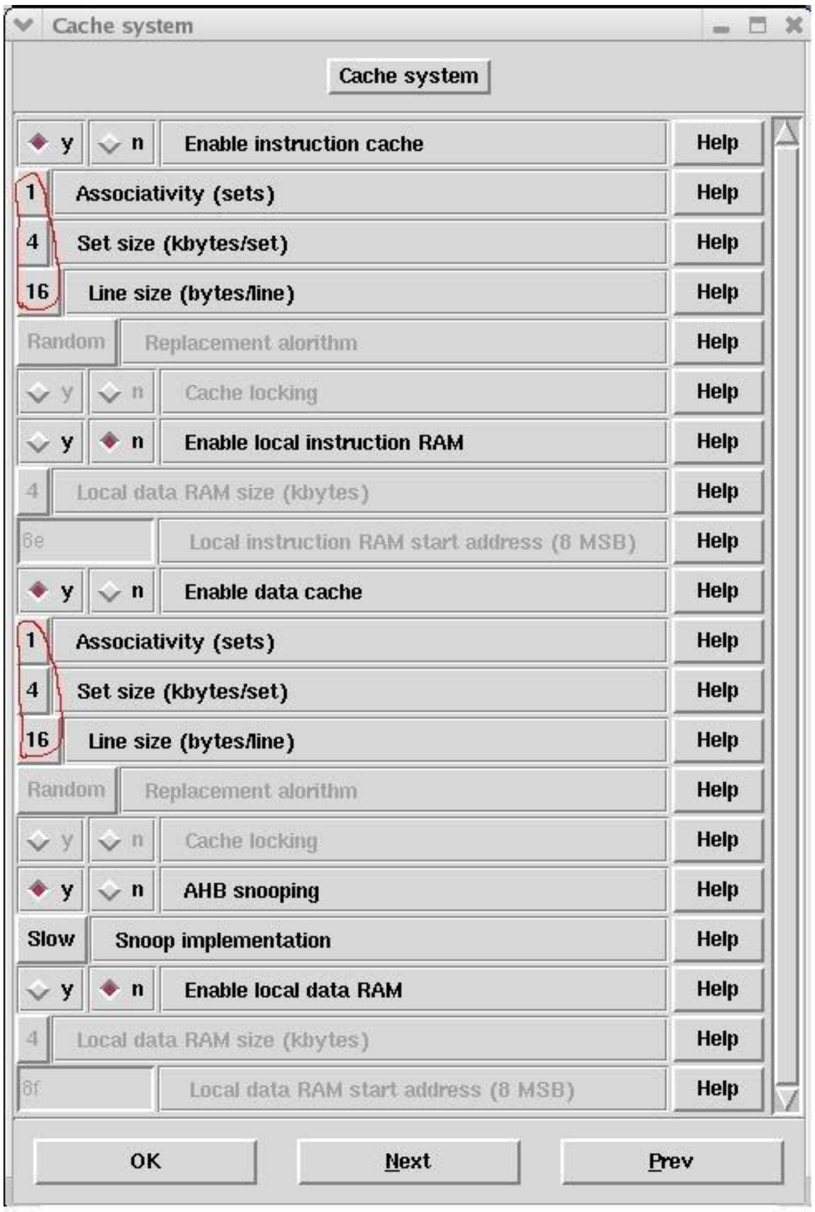

Processor

Processor

1	Number of processors	Help
	Integer unit	
	Floating-point unit	
	Cache system	
	MMU	
	Debug Support Unit	

Main Menu | Next | Prev

Figure 5.3.The option pane for the configuration of Leon3 processor.


Cache system

Cache system

[✓ y] n	Enable instruction cache	Help
1	Associativity (sets)	Help
4	Set size (kbytes/set)	Help
16	Line size (bytes/line)	Help
Random	Replacement alorithm	Help
y n	Cache locking	Help
y [✓ n]	Enable local instruction RAM	Help
4	Local data RAM size (kbytes)	Help
8e	Local instruction RAM start address (8 MSB)	Help
[✓ y] n	Enable data cache	Help
1	Associativity (sets)	Help
4	Set size (kbytes/set)	Help
16	Line size (bytes/line)	Help
Random	Replacement alorithm	Help
y n	Cache locking	Help
[✓ y] n	AHB snooping	Help
Slow	Snoop implementation	Help
y [✓ n]	Enable local data RAM	Help
4	Local data RAM size (kbytes)	Help
8f	Local data RAM start address (8 MSB)	Help

OK | Next | Prev

Figure 5.4.The Leon3 Data and Instruction cache configuration screenshot.

The detailed analysis of all the available Makefiles included into Gaisler's suit and of their functionality, as well as the analytic description of the dominant Gaisler's components, provided in parameterized VHDL code and used in our designs, can be found in the respective documents of Gaisler's Intellectual Property (IP) design library, namely *GRLIB* [32]. However, a decent presentation and analysis of Gaisler's bundle, from both the aspects of designs and of software support tools, can also be found in a relevant diploma thesis [27], dedicated to the deployment and the exploration of the underlying suite, as well as to the implementation of Leon3 MP-SoC designs into both FPGA and ASIC platforms.

5.2.2 Synthesis of the Leon3 MP-SoC platform

Based on Gaisler's toolflow presented in the previous sub-section, the scripts and supplementary files dedicated to the ASIC synthesis and simulation of the Leon3-based MP-SoC design's version are included into the corresponding design platform's folder, named as *leon3-asic* by Gaisler. Hence, in the specified folder, Gaisler provides, apart from the VHDL files of the Leon3 MP-SoC design and of the I/O pads, another Makefile, through which the designer can generate the scripts required for the synthesis. The command that produces the synthesis scripts is, similarly to the previous one for the system's configuration, *make scripts* and generates the required gate-level synthesis scripts, depending on the target ASIC implementation technology included into Gaisler's bundle.

The makefile's execution produces scripts for a wide variety of synthesis tools, in order to cover the wide variety of EDA toolchains used by designers working with Gaisler's Leon3 distribution. Also, the range of the included technology libraries is quite extensive, starting from libraries of 180nm up to 65nm, which was the state-of-the-art technology node two years ago. In this thesis, we have used the TSMC 45nm CMOS standard-cell library, as it is based on a technology node of the deep-deep submicron regime and therefore, it is suitable for the exploration of the impact that EM and TDDB may have on the interconnects of such complex designs as our MP-SoC platform.

Among the synthesis scripts generated by the *make scripts* command, we decided to use those corresponding to the UMC 180nm technology library and we adapted them to our technology libraries. It must be noted that the selection of the specific category of synthesis scripts was not dependent on any specific criteria and therefore, we could have also used the ones generated for the 90nm or the 65nm technology libraries, without loss of generality.

The generated synthesis scripts were divided into several parts, depending on the consecutive steps that should be followed, in order to transform the RTL code into a technology-dependent gate-level netlist. Hence, there is a main script, namely *rhumc.tcl*, which includes the scripts that perform the technology library definition (*setup_rhumc.tcl*), the VHDL compilation of the design's components (*compile.dc* and *leon3mp_dc.tcl*), the declaration of the timing constraints (*timing.tcl*) and the final commands that perform the synthesis of the design, extract the report and generate the netlist in design formats compatible to the Synopsys Design Compiler and the SoC Encounter (*rhumc.tcl*).

The first one of the aforementioned scripts that is executed is the *setup_rhumc.tcl*, which includes the commands for the definition of the standard-cell libraries, as well as of the macro cells, which correspond to the SRAM memory libraries we used to implement the data and instruction caches of Leon3. The Design Compiler requires these libraries to be provided in the Synopsys database (*.db*) format and the corresponding standard-cell and SRAM $*library.db* files have been generated from the Synopsys Library Compiler [29] tool, based on the following commands.

```
$ lc_shell
$ read_lib library_name.lib
$ write_lib library_name –format db –output library_name.db
```

The first command invokes the Library Compiler tool and the user enters its shell. With the *read_lib* command, we input the technology library's information written in the Synopsys Liberty format and finally, we dump it into the corresponding *.db* file, while keeping the library's name, defined on the top of the *.lib* file, unchanged. After the generation of the required *.db* files, the libraries used in the synthesis process are declared, based on the *setup_rhumc.tcl* script, as follows:

```
set search_path [concat $search_path [list $designs_path $libraries_path]]
set target_library {$standard_cell_library_name.db $memory_library.db}
set synthetic_library {dw01.sldb dw02.sldb dw03.sldb dw04.sldb dw_foundation.sldb}
set link_library [list $standard_cell_library_name.db $memory_library.db *]
```

In these commands, given to the Design Compiler, we define the paths where the VHDL files of the design are located ($*designs_path*), as well as the directory to look for the standard-cell library and the memory libraries ($*libraries_path*). The *set target_library* command defines the libraries that should be used in the synthesis, while the *set link_library* is necessary to be defined while linking the design with the target libraries, during the synthesis process.

Also, the linking libraries are necessary to be defined when reading an already synthesized design, which includes logic cells and macros of a certain technology library, into Design Compiler's environment. In perspective, when a design is synthesized and we want to modify its netlist or to extract the timing, area and power reports as a standalone process, we should read it through the available formats, namely Verilog, VHDL or DDC, (Database Design Compiler format) and then link it through the *link* command, so that the timing, area and power characteristics of the involved technology libraries are incorporated.

Finally, we also define the usage of Synopsys DesignWare IP libraries, which include components of parameterized size and complexity, in order to efficiently implement operations like additions (*dw01.sldb*), multiplications (*dw02.sldb*) and other Digital Signal Processing operations (*dw_foundation.sldb*). These components have been used for the high-performance and low-area implementation of arithmetic operations located into the Leon3 processor's integer unit, as well as in other components of the MP-SoC designs.

Regarding the generation of the SRAM memory *.db* libraries, we have been based on the *.lib* files of an educational technology library of CMOS 90nm, provided by Synopsys Armenia (SAED 90nm) [30], and on a TSMC design kit, as this was the only available SRAM cell library close to the 45nm technology node. Hence, we transformed the technological characteristics regarding the process corner names, the operating voltage and the timing characteristics of the SAED *.lib* SRAM cell files and adapted them to the ones defined into the corresponding TSMC 45nm *.lib*. This conversion was required in order to avoid problems in static timing analysis at both the synthesis and the physical implementation stages, as the operating voltage and the rise and fall time thresholds were different in the 90nm SAED standard-cell library (e.g. different "trip point" warning messages in Design Compiler occur when cells are inconsistent in terms of timing parameters or operating voltage). However, significant problems also arised in the usage and the instantiation of the specific memory library's SRAM cells, as the address and data vectors of the input and output ports were not properly defined. Hence, we modified the declaration of input and output ports in the *.lib* files of the target SRAM cells, based on the Synopsys Liberty format's specifications [31]. A more detailed view of the SRAM cell *.lib* files is available in the Appendix.

The proper library setup and definition is perhaps the most critical issue in ASIC synthesis and in general, in any design abstraction layer of the RTL-to-GDSII flow and this is the reason of emphasizing on this part of the text in the previous two paragraphs. After the completion of this stage, the synthesis flow, according to the structure of Gaisler's scripts and also of any other design's case study, continues with the VHDL compilation of the designs in the Leon3's hierarchy. The corresponding scripts are the *compile.dc*, in which the Design Compiler's *PRESTO* tool checks the correctness of the VHDL components included in the design's hierarchy, and the *leon3mp_dc.tcl*, which performs the same task for the top hierarchical components, namely the Leon3 MP-SoC design and the I/O pads used.

In *compile.dc*, each one of the IP components that correspond to a specific IP library, the name of which is determined by Gaisler, are compiled, by defining the corresponding library of the same name in Design Compiler. The top hierarchical modules are compiled through the execution of the *leon3mp_dc.tcl* script, as mentioned above, under the default design library of Design Compiler (*work*). The compilation of the VHDL modules is performed by using the *analyze* command, as follows.

```
$ define_design_lib $current_design_lib
$ analyze –format VHDL –library $current_design_lib $module.vhd
```

With the *define_design_lib* command, we define a working library in Design Compiler, under which all the corresponding modules should be compiled. Therefore, the produced synthesis files for these modules will be placed into the folder named $*current_design_lib* in the working directory from which the scripts are executed. The *analyze* command compiles the $*module.vhd* file under the *$current_design_lib* design library, defined previously. This process has been followed for all the designs included in the *GRLIB* IP library, as they could potentially be used in the target design. The components of GRLIB included in the synthesis process are determined by the system's parameters configured via the GUI of the *make xconfig* command. The values of these parameters are also included in the *config.vhd* package, compiled under the library work in the *leon3mp_dc.tcl* script.

The final step in *leon3mp_dc.tcl* is the transformation of the RTL netlist of the top design, namely of *leon3mp*, from the RTL description to logic gates of an abstract technology library, named as *GTECH* in Design Compiler. At this step, the checking of the VHDL modules is completed successfully and we have stepped into the main process of the top-down synthesis flow, which is the generation of a circuit composed of logic gates of a specific technology, which is defined in the *setup_rhumc.tcl* script. This process is performed by the Design Compiler in two stages. First, the RTL representation of the target module, namely *leon3mp*, is transformed to logic gates which do not correspond to a specific CMOS technology, so as to produce an intermediate representation of gates, which would be later mapped to those of the target standard-cell and SRAM technology libraries. These gates do not include any timing, area or power information and this is the reason that Synopsys uses the name GTECH for this library. This step is performed by the *elaborate* command, as follows:

$ *elaborate $top_design*

In our case, the top_design was the *leon3mp* module, namely the top hierarchical component of our design.

After the completion of the design's elaborate process, the next stage deals with the setting of constraints imposed by the user and bowl under the timing, area and power bounds, in between which the design should operate. In the Design Compiler, the setting of timing constraints is the most critical task, as this is the constraint of the highest importance for the synthesis. Consequently, the designer should define the clocks of the design and their periods, so that the register-to-register paths are constrained by the corresponding clock. Also, the in-to-register and the register-to-out paths are affected by such constraints, not only in terms of setup time boundary, which is considered during the synthesis, but also of hold time, which can be monitored at the post-layout timing analysis. Also, in-to-out paths can be constrained by the *set_max_delay* command, which is provided below, along with a clock definition. These two are the core timing constraints used in the corresponding Gaisler's script, namely *timing.tcl*, according to the following syntax.

$ *create_clock "$clock_name" –name $clock_name –period $clock_period*
$ *set_max_delay $max_output_delay –from $inputs –to $outputs*

Regarding the operating condition constraints, the design was synthesized at the typical process "corner" of the TSMC 45nm library, where the operating voltage was set to 0.9V and the temperature to 25^{o}C. Also, we let the tool decide about the proper wire load model of the standard-cell library that should be used in the internal parts of the modules and across the design boundaries, by setting the *set_auto_wire_load_selection* variable to *true*. The rest of the constraints included in the *timing.tcl* and concerning the ungrouping of specific modules in the design's hierarchy were set as inactive in our synthesis approach, as the original Gaisler's script followed a strategy of ungrouping small hierarchies in the design, thereby corrupting some of the produced design's hierarchical levels. Provided that the generated post-synthesis netlist will be used as input to Cadence SoC Encounter to construct the design's floorplan, which will be used by HotSpot for the thermal simulations, we should keep all the designs' hierarchical levels, as these existed before the gate-level synthesis stage. The main reason for the maintenance of the design's hierarchy is that the generated floorplan guide affects the thermal profiling in HotSpot. Consequently, hierarchical units of the lowest possible hierarchical level and therefore of more detailed granularity could possibly aid in the extraction of more accurate thermal analysis results by the thermal simulation tool. Hence, we excluded all the *ungroup* commands from the *timing.tcl* script, as it can be seen in the corresponding section of the Appendix.

The setting of the design's constraints is the last step before the *compile* command, which performs the synthesis of the design using the standard-cell and SRAM components included into the libraries defined in *setup_rhumc.tcl*. Therefore, the design's netlist, which was, until now, kept in an intermediate form, in logic gates of an abstract technology, is now transformed into a new, technology-aware netlist, which, at the end, will comprise only of components that correspond to a CMOS technology library. Regarding the synthesis strategy, we followed a top-down approach, where the target design (leon3mp) and its subdesigns are read from the Design Compiler in one step and any optimizations across the design's hierarchy are performed on-the-fly. This implies that sub-designs are not synthesized in a standalone manner and read afterwards, but the whole design is loaded at once and any optimizations of the sub-designs are performed during the synthesis's iterations, depending also on the degree of optimization that the designer selects. Alternative, bottom-up synthesis approaches could be also applied in such a complex design, but this exploration is out of the scope of the presented thesis.

In general, the Design Compiler provides two options for compiling a design into cells of a certain technology. The first one is the default *compile* command, in which extra optimizations regarding the efficiency of technology mapping or the cross-hierarchy optimizations are by default set at a medium level or disabled respectively. In the *compile_ultra* command, these flags are by default enabled by the tool, so that the maximum complexity reduction and timing improvements are achieved.

Each one of these options has its pros and cons, depending on the design. However, in datapath optimization and especially in non-programmable ASIC designs, it is more appropriate to use the *compile_ultra* strategy, in order to achieve the maximum performance in terms of timing and area. On the other hand, a more conservative approach should be followed when synthesizing designs including processors, as the merging of registers and signals *compile_ultra* possibly performs, could lead to errors that may affect the proper post-synthesis functional simulation of the design.

In our approach, we synthesized the design using the *compile* command and under a clock period constraint of 2ns, while the in-to-out paths where constrained at 1.9ns, so as to be compatible with the delay of register-to-register paths, considering the setup time's impact. After the successful completion of the synthesis, we exported the design into the Synopsys Database format (*.ddc*) and also into a Verilog netlist (*.sv*), which was then provided to the Cadence SoC Encounter for the place-and-route stage, along with the set of the technology library file required and the timing constraints, which were dumped into an *.sdc* file, also needed in the physical implementation process. The corresponding commands used for the design's exportation and also for the constraint file's generation can be found near the bottom of the *rhumc.tcl* script, placed in the Appendix.

5.2.3 Physical Implementation of the Leon3 MP-SoC design

5.2.3.1 Design import and Floorplanning

In a typical RTL-to-GDSII flow, the physical implementation stage includes the steps of floorplanning, placement, clock-tree synthesis and routing of the imported gate-level netlist, resulting into the design's layout, the database of which is then imported in our framework and used for the extraction of the target interconnects. Each of the aforementioned separate physical implementation steps impacts the following ones in the design flow and holistically, the generation of the final physical design (layout). Therefore, each of these steps is of high importance, especially when the imported design is complex and contains SRAM cells and other macros in addition to the standard-cells.

In this section, we will elaborate on the flow of the placement and routing of a standard-cell based gate-level netlist of high complexity, like the Leon3 MP-SoC design produced from the synthesis process described previously. Our design, which includes two Leon3 processor cores and whose gate complexity is about 30K gates, contains one instruction and one data cache of 8K per processor, while it can also contain I/O pads, depending on the technology library of the implementation. It is noted that in the *config.vhd* package, through which the designer can determine the critical system's parameters as it has already been mentioned, we did not select any technology of implementation for the I/O pads, as there were no timing, power and physical information files available in the TSMC 45nm library distribution. Hence, in our design, the I/O pads were implemented by tri-state and typical buffers of our library, by selecting cells of high fanout to drive the input and output pins.

However, the complexity of the design was not reduced by absorbing the existence of I/O pads, as the remaining parts of the design required a careful floorplan of the hierarchical modules, so that the design's performance is kept at a reasonable level, while the congestion of cells during placement and routing is maintained low, in order to avoid problems during both the global and detail routing of cells. Moreover, we also came against the challenge of adjusting the pin shapes and dimensions in the *.lef* files of the SAED 90nm SRAM cell library, so that no design rule violations occur during the routing. This problem was solved by reducing the width of input and output pins in the .lef files of the SRAM cells, considering the minimum width of metal layers in the corresponding .lef of the TSMC 45nm library. By such an approach, we wanted to guarantee that no design rule violations could occur, leading to open nets of the memory cells and of the standard-cells that are connected to the caches. In the presented design, we used five SRAM macro cells, namely the SRAM32x128, SRAM32x1024, SRAM32x64 and SRAM8x128, all migrated from the SAED 90nm library to the 45nm used in the context of this work.

Among those, the SRAM32x1024 memory cells were used to implement the 8K instruction and data caches of each processor core, while the SRAM32x128 and SRAM8x128 cells were used for the instruction trace buffer and for the processor's register file. Also, the SRAM32x64 blocks were used to map the memory blocks of the Debug Support Unit and the SRAM32x16 for the Memory Management Unit's internal memory. It must be noted that we did not perform any scaling on the overall size of the memory blocks or on any other geometric dimension internally, apart from the input and output pins, which were responsible for the connection of the memories with the standard-cells.

Regarding the placement of memories onto the core area, this was a difficult task, as we had to experiment with the trade-off of performance and congestion. In perspective, the data and instruction caches were interfacing with the Leon3's integer unit, which comprises the processor's pipeline. Also, the Memory Management Unit and the data and instruction cache controllers interface directly with the caches as well.

Consequently, the distance of the hierarchical units of the Leon3 core in the floorplan had to be adjusted, so that there was enough space left for the violation-free placement and routing of the logic cells surrounding the SRAMs that implement the caches. Moreover, the integer unit, which comprises the Leon3 pipeline, is placed near the memory management unit and to the register file, so that the interconnect delay is minimized in between these components. However, as the technology of implementation is at 45nm, such an approach has been followed in every part of the design's floorplan as a general strategy, as the interconnect delay tends to increase dramatically with technology scaling and overwhelm the delay of logic gates in technology nodes below 45nm, depending on the design. The aforementioned hierarchical placement strategy is depicted in the floorplan of the following figure.

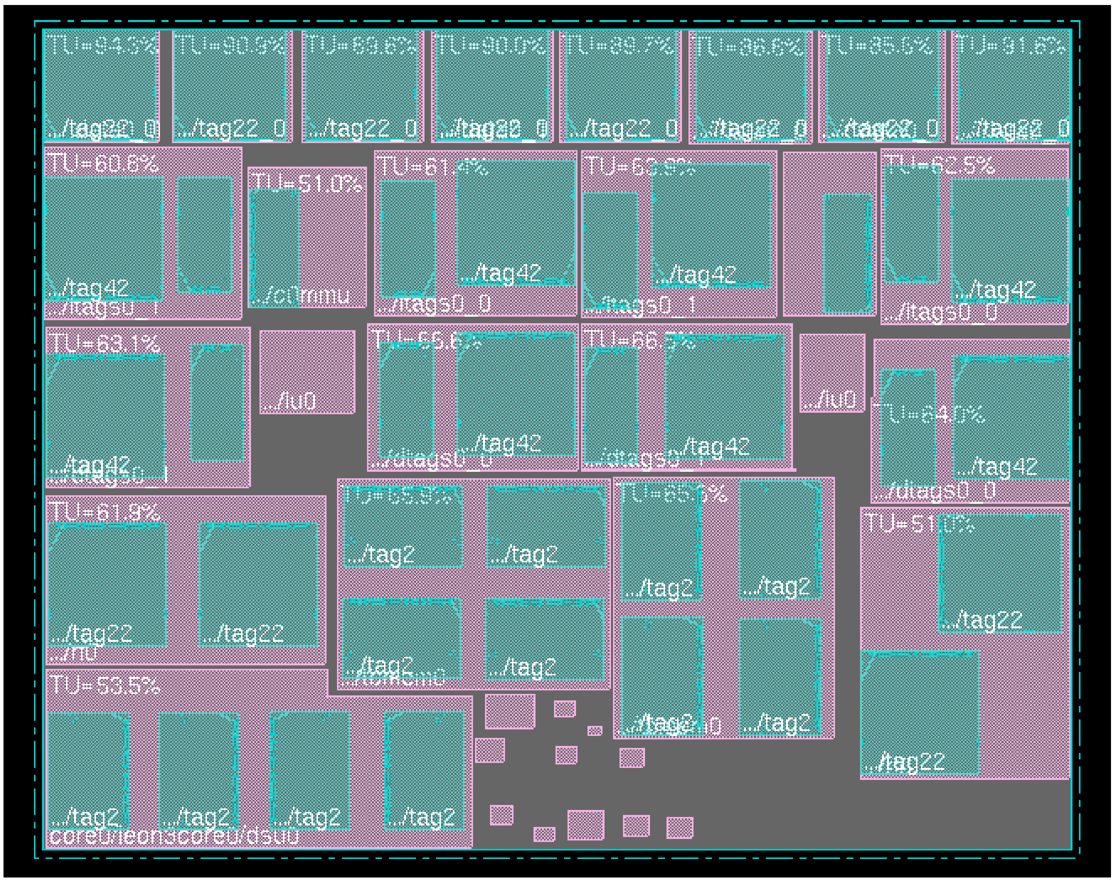


Figure 5.5. The 2.32x1.85 mm2 Leon3 MP-SoC design's floorplan with two cores.

The screenshot of Figure 5.5 demonstrates the floorplan of the design with the two Leon3 processors' integer and memory management units (iu0 and c0mmu respectively), which are surrounded by the data and instruction caches on the top of the core area and also by the caches storing the data and instruction tag bits. The register file and the instruction trace buffer memories are placed below the integer unit. It can be seen that the hierarchical units of the two processors are placed by following the same floorplanning strategy, illustrated above. The Debug Support unit (core0/leon3core0/dsu0) of the system is placed in the left bottom corner of the core area and is comprised by four SRAM32x64 caches, while the AMBA AHB and APB controllers and the design's peripherals are placed near the bottom right corner, next to the DSU unit.

In the picture of Figure 5.5, the green-colored blocks represent the SRAM cells, while the purple boxes enclosing them are the hierarchical units of the design, which also contain standard-cells, apart from the memory macros. The floorplan's implementation has been performed manually from within the SoC Encounter's suite, by using its graphical interface and moving or expanding the blocks, so that they are placed in the desired location. For such purposes, we have used the Encounter's moving and expanding buttons of the graphical user interface, as the location of units was not decided a priori. However, we could have realized the floorplan by using the respective commands of DBAccess to automate the placement of macro blocks and hierarchical units, once we had decided on the left bottom and right top coordinates of the blocks.

Regarding the floorplan's details affecting the next tasks, namely placement and routing, we have selected the hierarchical units to be of type "*Guide*", instead of "*Fence*", so that the standard-cells may not remain enclosed between at the boundaries of the hierarchical unit during the detailed placement. The type of each hierarchical unit in the floorplan can be set by double-clicking on a specific unit, so that the window with the properties of that unit appears and the user can select among the different floorplan types, in the "Constraint Type" option list. In this window, the designer can also set other relevant floorplan parameters, like the unit's width and height, its orientation and location on the core area, by adjusting the (x, y) coordinates of the left bottom and the right top points of the unit's rectangular box. In our implementation approach, the "*Guide*" option is selected so that Encounter's placer is not limited by such a constraint, when the designer decides to perform a timing-driven or a congestion-driven placement, which are scenarios explored in this thesis. The details on the available standard-cell placement strategies of SoC Encounter are analytically discussed in the following sub-section, along with our physical design approaches at this stage.

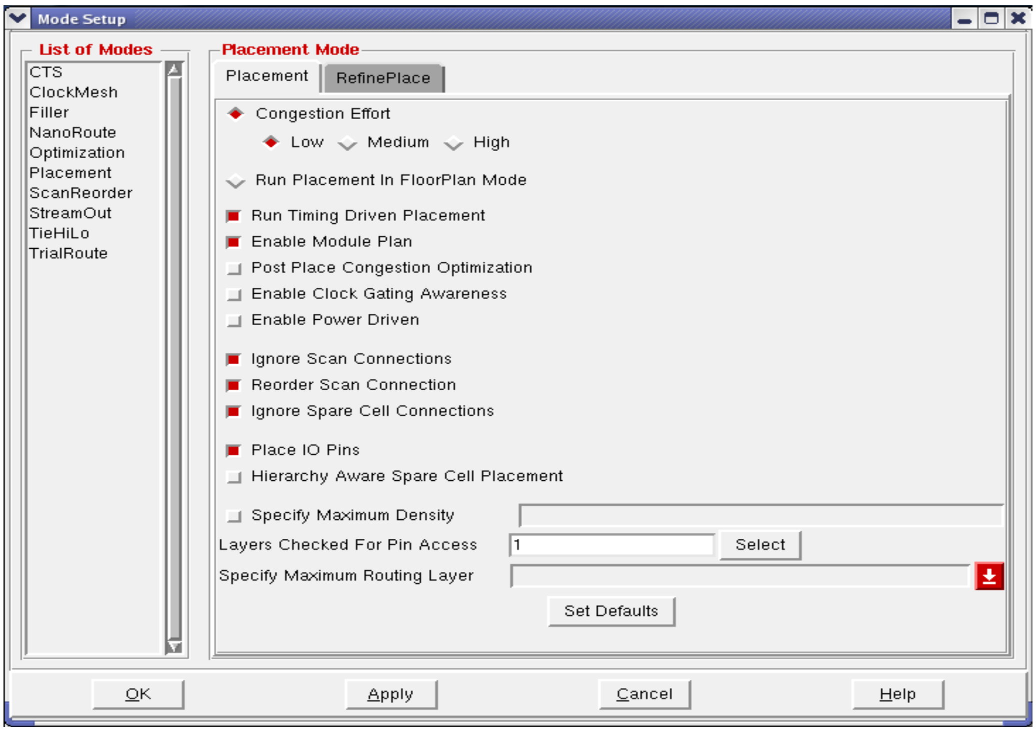
5.2.3.2 Standard-cell placement

The standard-cell placement is the most significant design stage of the physical implementation flow, as it takes the global placement of hierarchical modules and macro blocks as input and places the logic cells in the core area, so that each component has a legal location on the die. By the term "legal", we refer to the coordinates of a standard-cell or macro block after performing a detailed placement on the design. It must be noted that in RTL-to-GDSII flows, the core area on which the design must be placed is divided into rows of standard height, the value of which is determined by the technology library's LEF file, which contains the width and the height of standard-cells, as well as its physical dimensions, regarding the existing metal layers and internal pins.

All the standard-cells should have the same height, so that they can fit into the core area's rows. The row-based organization of the layout's area bowls under the issue of routing the power and ground signals (Special routing) of standard-cells and their connection to the core's V_{dd} and *Gnd* rings. Hence, as the cells internally have their power and ground metal lines on the top and bottom respectively, they should be placed within a specific row, so that the Special router of Encounter connects their V_{dd}/*Gnd* pins to the core ring, by using a single long metal line for each power signal across the entire layout, from one end to the other. In order to achieve a legal placement, all cells should be placed on a specific layout row without overlapping with each other or with macro blocks, while their locations should be on the manufacturing grid. The manufacturing grid is a virtual mesh of horizontal and vertical lines that Encounter uses in order to properly place and route the design, following the design rules of the specific technology used. The horizontal and vertical spacing of these lines is defined in the technology library's LEF file and determines the spacing of the manufacturing grid, the resolution of which depends on the physical dimensions of the library. Therefore, for a typical 180nm CMOS technology, this value is 0.05um, while for the 45nm TSMC, it is reduced to 0.005um. Consequently, the designers should be careful when manually placing the macro cells or performing standard-cell placement, as non-legal module locations could lead to increased runtime in routing or to areas that cannot be routed, due to off-grid placement.

However, these are not the only constraints a designer should worry about when attempting to place complex designs with numerous macro blocks, such as our Leon3-based case study. Placement can impact routing and consequently the design's closure in numerous ways, as the connections of pins are bounded by technological constraints listed in the LEF file of the technology library and resulting into over-congested areas, while routing the design. Hence, congestion-aware placement should be employed, while not underestimating the design specs, which depend on the timing constraints. Therefore, the designer has to decide from a variety of placement scenarios, depending on the core area's size, the designs' complexity and the performance constraints. In this section, we will present how we performed both the timing-driven and the congestion-driven placement approach on the Leon3 MP-SoC design using the Encounter's QPlace placer [34], while taking the placement legalization and the timing constraints into account.

From the graphical user interface of Encounter, placement is performed by selecting from the menu Place->Standard-cells. Then, the option pane that appears provides choice of either a detailed placement (default) or a low-quality one (Run placement in Floorplan mode). Details on the placement's process can be fine-tuned by clicking on the button "Mode", which opens a window with more granular placement options, shown below.

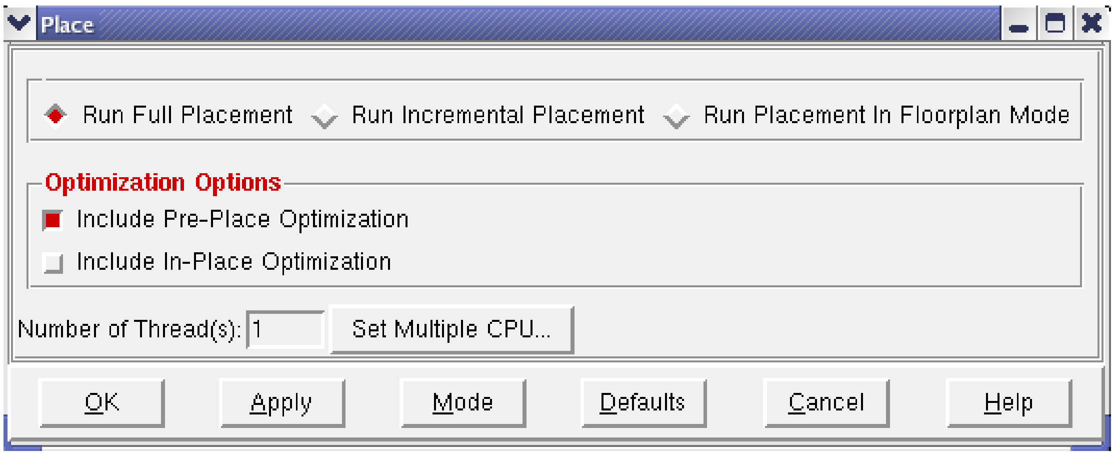


[✓] Congestion Effort
[✓ Low] Medium High
[ ] Run Placement In FloorPlan Mode
[✓] Run Timing Driven Placement
[✓] Enable Module Plan
[ ] Post Place Congestion Optimization
[ ] Enable Clock Gating Awareness
[ ] Enable Power Driven
[✓] Ignore Scan Connections
[✓] Reorder Scan Connection
[✓] Ignore Spare Cell Connections
[✓] Place IO Pins
[ ] Hierarchy Aware Spare Cell Placement
[ ] Specify Maximum Density
Layers Checked For Pin Access 1
Specify Maximum Routing Layer

Figure 5.6. Placement options of Encounter's Qplace placement tool.

In Figure 5.6, the placement tool of Encounter provides options regarding several issues affecting detailed placement of the design. In our approach, we have focused on performance and congestion as the two parameters that may guide placement either in conjunction with each other, or separately. For the purposes of the interconnect reliability framework's evaluation under different placement scenarios, we decided to implement the placement either in timing-driven or in a congestion-driven mode.

For the timing-driven approach, we selected the corresponding checkbox as shown in Figure 5.6, where it is already selected. At the same time, the designer should load the timing constraints, by selecting from the Encounter's menu the tab "Timing->Load Timing Constraint" and inserting the *.sdc* file with the clock period and the input-to-output timing constraints, generated by the Design Compiler at the synthesis step. For a post-synthesis optimization, the designer should also select the "Include In-Place Optimization" option, appearing when clicking on the "Place->Standard-cells" button of the menu. This window is also shown here.


Place

[✓ Run Full Placement] Run Incremental Placement Run Placement In Floorplan Mode

Optimization Options

[✓] Include Pre-Place Optimization

[ ] Include In-Place Optimization

Number of Thread(s): 1

Figure 5.7. The general option pane of placement modes in Encounter.

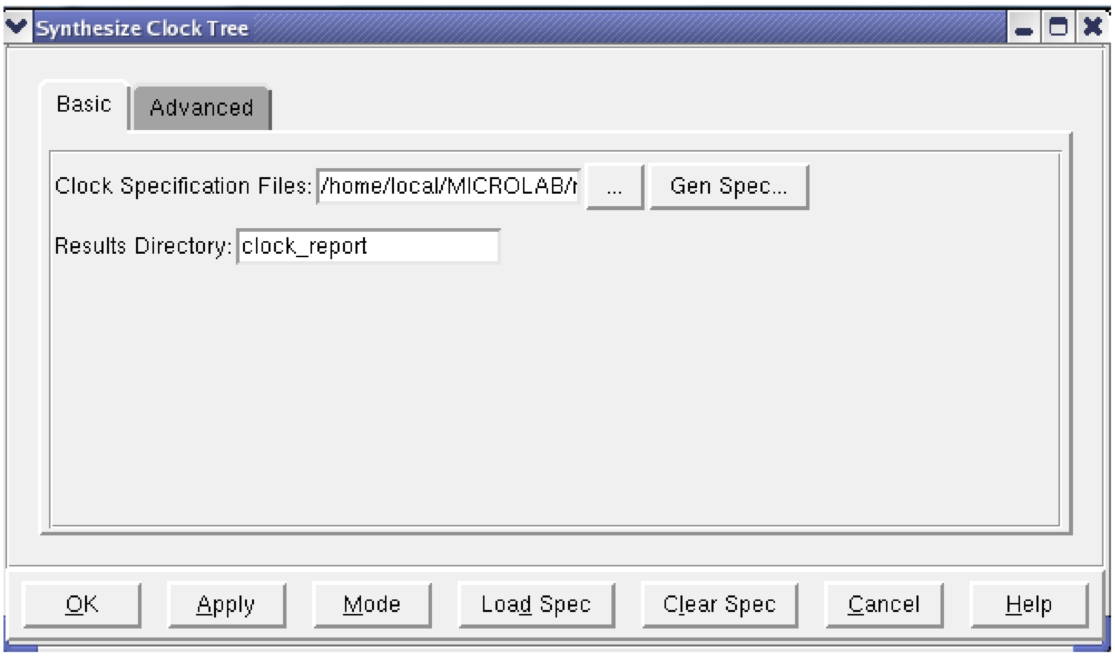
In the above figure, timing-driven placement should be accompanied by the In-Place optimization, in order to achieve a larger timing slack before the clock-tree synthesis and the routing steps that follow. On the other hand, when selecting a fully congestion-aware placement approach, post-place optimization refers to congestion avoidance and it can also be included in the options. The congestion effort of the placer is determined as "High" for a congestion-aware placement strategy (or as "Medium", for a rather typical effort), by clicking the corresponding option, as it can be seen in Figure 5.6. By clicking the "Ok" button of Figure 5.7, the user can launch the placer and proceed to the clock-tree synthesis step, after achieving a legal placement.

5.2.3.3 Clock-tree synthesis

The construction of the clock-tree is usually performed before the routing of the signal nets and deals with the design and implementation of the H-tree of clock buffers that drive the clock signals to the whole chip, wherever flip-flops are placed. Hence, placement is compulsory before the clock-tree synthesis and routing. In this thesis, our focus was not on the clock tree network and consequently, we will not explore any special strategies regarding the synthesis of the design's clock-tree.

Nevertheless, a properly synthesized clock tree is likely to eliminate any problems that could occur in the timing of register-to-register paths because of excessive clock latency, so it is necessary not to introduce any delay overhead in the design's initial timing due to clock-tree malfunctions.

Due to the aforementioned reasons, we selected all the available CMOS inverters and buffers from the TSMC 45nm standard-cell library, as well as the existing clock-tree buffers, specialized for the H-tree construction, and we let Encounter allocate the proper ones, based on the timing constraints of the design. This task is performed by selecting "Clock->Design Clock->Gen. Spec" from Encounter's menu tab, which opens a window with the available library's buffers. We used all the available buffers except for those with the prefix "DEL", which are cells of merged consecutive buffers, and those with the prefix "G", which are mainly used for a post-layout optimization purposes (ECO flow). The selection of inverters and buffers is dumped into a text file that we named *Clock.ctstch* (the default name, which of course can be changed by the user), that is then used by Encounter to guide the clock-tree synthesis. The main window containing the clock-tree synthesis options is shown below.


Clock Specification Files: /home/local/MICROLAB/r

Results Directory: clock_report

Figure 5.8. The clock-tree synthesis main window in SoC Encounter.

By clicking on the "Mode" button shown in Figure 5.8, the designer can further elaborate on the diverse clock-tree synthesis strategies, regarding the metal layer of mapping the clock-tree root, the resizing or not of the clock-tree buffers while performing the clock-tree optimization, as well as the tool that will perform the clock-tree routing, if it is desired at this level. If not, the clock-tree buffers can be kept in the design and their interconnection with the remaining circuit will be performed together with that of the signal nets at the routing stage, where the Nanoroute tool is employed to finalize the physical design's implementation. Although the clock-tree synthesis finishes with a trial routing of all the designs' nets, namely both of the clock-tree and of the signal nets, this router (TrialRoute) does not take all the technology constraints into account during the pin connections, neither does it report all the possible design rule violations, regarding spacing, samenet and shorts. Therefore, the flow continues by performing the final routing of the design's nets, in order to end up with a better quality layout and consequently results of higher accuracy in terms of timing, due to the rerouting of nets and the construction of different wires by Nanoroute. The design is also optimized in terms of area, due to the possible resizing of cells while performing post-routing timing optimizations on the design.

5.2.3.4 Routing

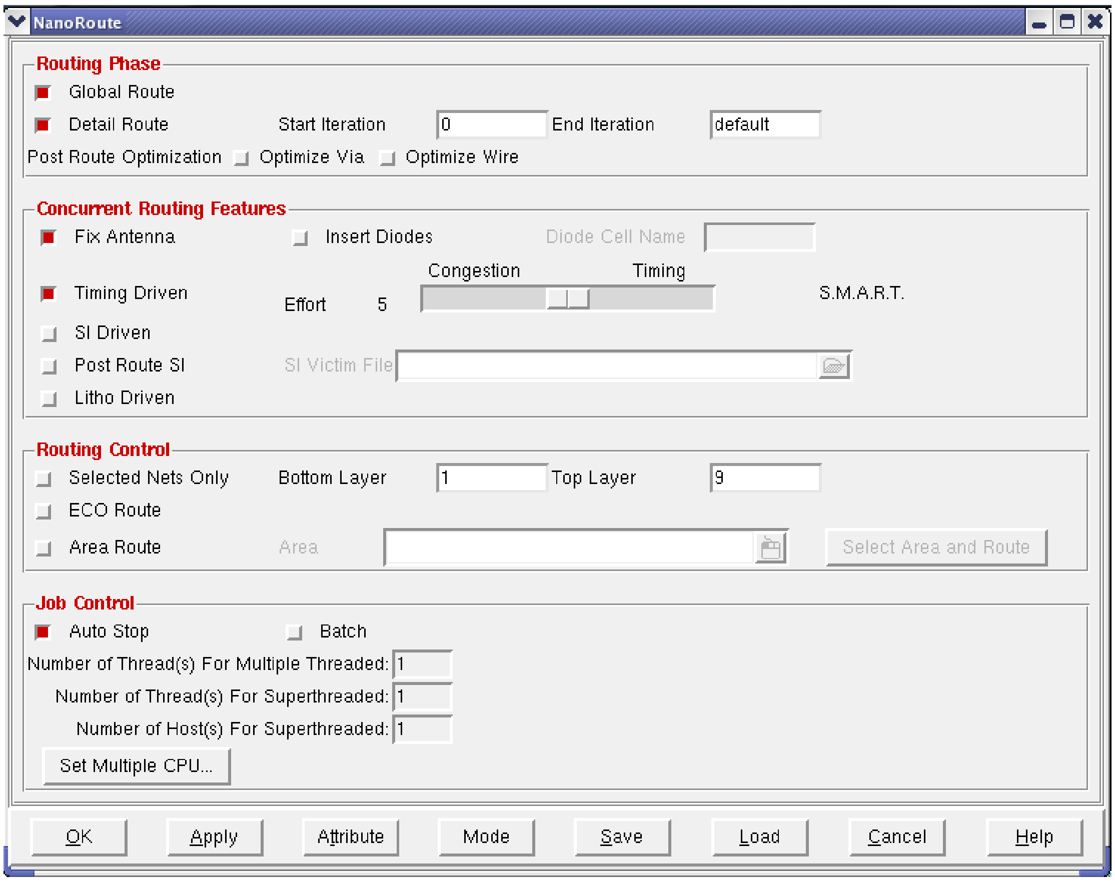
Instead of the clock-tree synthesis step, the routing stage presents the designer with the opportunity to select between different physical implementation scenarios regarding either timing or congestion. In perspective, routing in general deals with the interconnection of data signals in VLSI circuits and mainly with the selection of metal layers that should be used to connect pins of standard-cells and macros. It is a problem of high complexity, depending on the number of pins that have to be connected with each other and also on the available metal layers, which depend on the standard-cell library used. The complexity of most academic and industrial routing algorithms, based on which state-of-the-art tools have been developed, classify them among the NP-hard problems and consequently, complex, MP-SoC designs require great runtimes to be successfully routed.

At the same time, the successful completion of routing depends on the available die size, as well as on the congestion of the design, regulated during the placement. Therefore, over-congested designs may lead to a significant degradation of the routing process, as the router cannot perform the layer assignment in the congested areas. Moreover, misaligned pins, which are not on the manufacturing grid, may cause problems. Therefore, it is necessary that the standard-cell library files are properly setup and the previous steps of the physical implementation flow avoid the threat of congestion and the problems that origin from such a bad management of the die's area. The worst impact of congestion is the incomplete routing in the areas where no more wires can come through due to high cell and wire density.

Therefore, the congestion-avoidance scenario is a standard and well-known practice that designers should always follow, in order not to face the aforementioned problems. However, there are practices that manage the layout's area, in combination with a more "aggressive" approach, which also considers the performance constraints imposed on the various paths of the design. A timing-driven routing strategy is different from a more conservative, congestion-aware approach, as it usually assigns the wider metal layers, which are near the top of the metal stack, to the timing-critical paths of the design, avoiding detours where possible. The remaining metal layers are used for the less critical parts of the design. This policy should mitigate the impact of the EM, as the usage of top metal layers, which are wider, leads to a decrease in the current density. However, the harder the timing constraints, the more the router optimizes the netlist by upsizing the logic cells used and therefore, a trade-off is developed. Hence, in very strict timing constraints, relatively shorter wires may suffer from high current densities due to the large fanout of driving cells used to achieve the target performance, whereas in the congestion-aware scenario, longer wires are not that susceptible due to coarser placement, as the design is not pushed in terms of timing.

On the other hand, dependencies between routing implementation scenarios and interconnect reliability wear-out mechanisms are also suspected to exist while considering the impact of wires' congestion on TDDB. In perspective, in densely routed areas of the layout, which are produced when following a timing-driven placement and routing strategy, the spacing between wires tends to be reduced, as each routing bin (e.g. rectangular block granule of the core area used at the global and detailed routing for the congestion metric's definition) contains more wires coming through its boundaries. Hence, specific wires may have greater probability of showing a timing degradation due to TDDB, because of the increased number of adjacent wires and the reduced spacing. Similarly, a sparse placement and routing strategy should lead to designs where the inter-metal wire spacing is larger and therefore the impact of TDDB is not as significant within the desired system's lifetime. However, all of the above are just assumptions that comprise our motivation to perform a more extended exploration and will be confirmed or refuted by the corresponding results, based on a wide range of designs, before they can be considered accurate and appropriate as reliability-aware implementation strategies.

In this thesis, we considered five place-and-route implementation scenarios in order to explore the impact that different physical implementation strategies could have on parametric interconnect reliability wear-out phenomena like EM and TDDB. In perspective, we initially implemented the design by selecting a medium congestion effort in both placement and routing, while we did not enable the timing-driven option in placement. Also, routing was of medium effort regarding performance. This design was considered to be a neutral one, as a strategy of medium effort was followed in both physical implementation stages. In order to set the routing strategy's options in SoC Encounter's Nanoroute, the designer should select the option "Route->NanoRoute -> Route" from Encounter's menu and the window with the main options, shown in the next figure, should appear.


NanoRoute

Routing Phase
☑ Global Route
☑ Detail Route   Start Iteration 0   End Iteration default
Post Route Optimization ☐ Optimize Via ☐ Optimize Wire

Concurrent Routing Features
☑ Fix Antenna   ☐ Insert Diodes   Diode Cell Name
☑ Timing Driven   Effort 5   Congestion — Timing   S.M.A.R.T.
☐ SI Driven
☐ Post Route SI   SI Victim File
☐ Litho Driven

Routing Control
☐ Selected Nets Only   Bottom Layer 1   Top Layer 9
☐ ECO Route
☐ Area Route   Area   Select Area and Route

Job Control
☑ Auto Stop   ☐ Batch
Number of Thread(s) For Multiple Threaded: 1
Number of Thread(s) For Superthreaded: 1
Number of Host(s) For Superthreaded: 1
Set Multiple CPU...

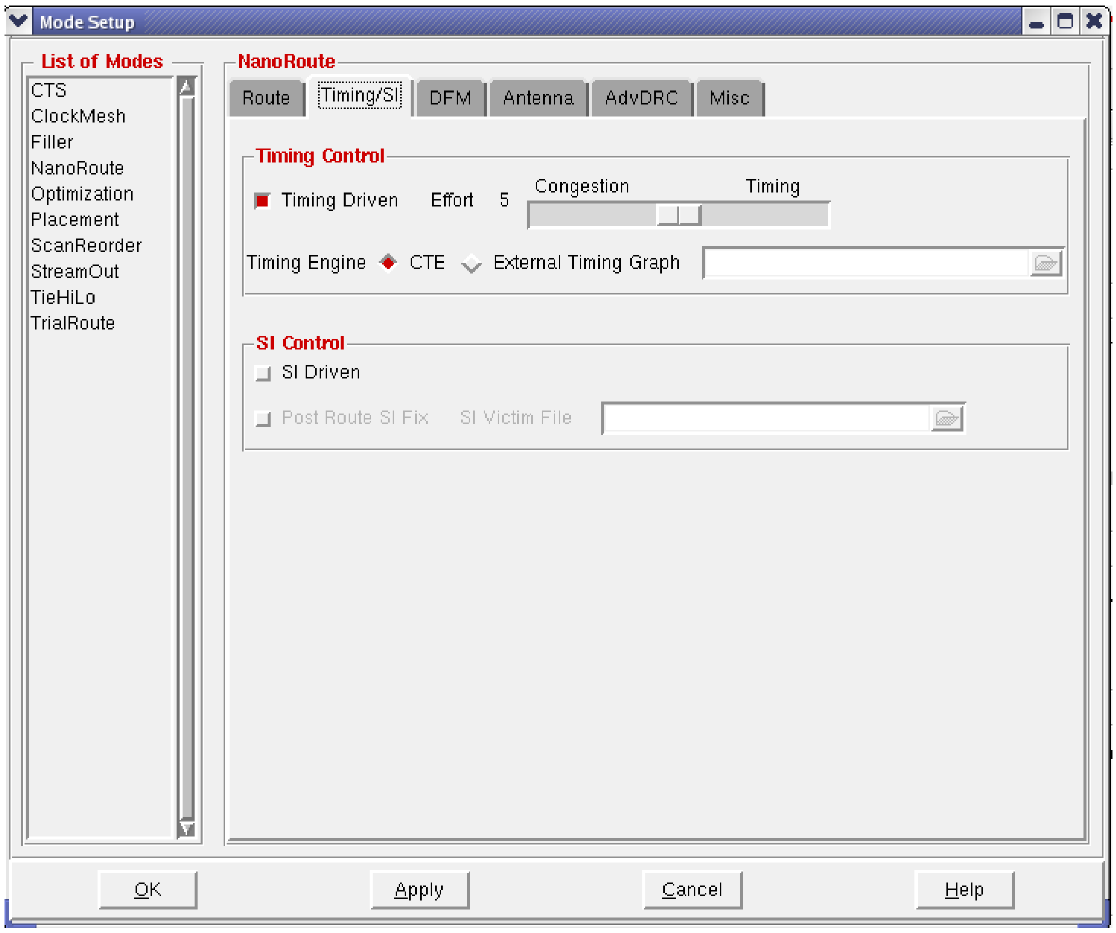
OK   Apply   Attribute   Mode   Save   Load   Cancel   Help

Figure 5.9. The Nanoroute's main window with the most important options.

In Figure 5.9, it is visible that the designer can determine most of the critical routing options, like the granularity of routing (“Global Route” and “Detail Route”), as well as the fixing of antenna process violations, the congestion and the timing optimization effort and also Signal Integrity and Lithography features. At the bottom of the pane, the number of threads executing the routing process can also be adjusted, as the designer can choose to map the application to more than one CPU, if these exist on the host where Nanoroute is running.

From the aforementioned options, we have focused on timing and congestion, as these parameters are the critical ones for the implementation of our routing scenarios. Hence, by clicking on the “Mode” button, another window, where each of the separate Nanoroute features can be configured, appears.

Regarding the dependence between timing and congestion aware optimization, we configured the slide bar of the next figure to be either on the right end, for a full timing-driven routing, or on the left end, to perform a totally congestion-driven routing. Considering a rather normal timing and congestion effort, the slide bar should be placed in the middle (Effort 5), as it is shown below. The timing engine of the SoC Encounter, CTE (Common Timing Engine) is enabled for the post-routing timing analysis. Through the other illustrated panes (Route, DFM, Antenna, AdvDRC and Misc), the designer can configure parameters like the metal layers that should be used for routing, as well as DRC and Antenna process violation issues.


Mode Setup

List of Modes
- CTS
- ClockMesh
- Filler
- NanoRoute
- Optimization
- Placement
- ScanReorder
- StreamOut
- TieHiLo
- TrialRoute

NanoRoute

Route | Timing/SI | DFM | Antenna | AdvDRC | Misc

Timing Control

[✓ Timing Driven] Effort 5 Congestion Timing

Timing Engine [✓ CTE] External Timing Graph

SI Control

SI Driven

Post Route SI Fix SI Victim File

Figure 5.10. The timing and congestion optimization slide bar of Nanoroute.

In our case study, we have selected the slide bar to be at the medium effort for the normal place-and-route design, as well as for the layouts of timing-driven and congestion-driven placement respectively. For the exclusively performance-optimized routing, the slide bar was set to Effort 10, which is the maximum, whereas Effort 0 corresponds to the completely congestion-optimized physical design. After the violation-free completion of the routing's design rule check, the layout is almost fully constructed and after the filler cell insertion, which is required to cover the white space of the core area, the design is almost finalized. Its final view is similar to the one illustrated in Figure 5.11, where the design's placement is timing-driven and the routing effort is the standard one.

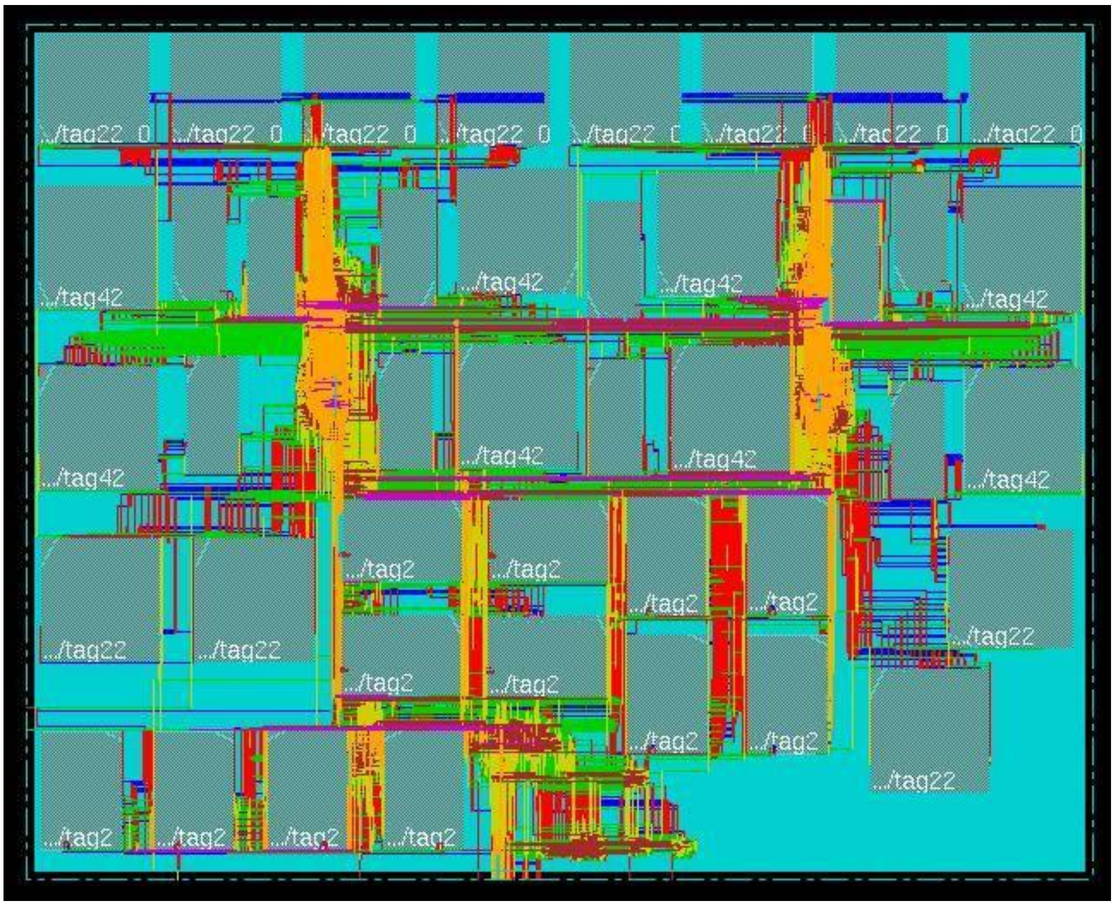


Figure 5.11. The design's layout with timing-driven placement and normal routing.

Therefore, the next step is the post-layout static timing analysis from within the Encounter's environment, by selecting "Timing->Analyze Timing-> Post-Layout". The reports provide the timing information about the longest path delay in all types of paths and of course in the register-to-register paths, constrained by the desired operating clock period, set by the user in the *.sdc* constraints file. If the desired performance is not achieved, Nanoroute is able to perform incremental post-routing optimization iterations in order to fix the setup and hold timing violations. This task is performed through the menu of Encounter's suite, by following the steps "Timing->Optimize->Post-Route", and selecting either setup or hold type violation fixing, as well as the optimization of fanout and capacitance of nets in the design. After the completion of each post-route optimization step, Encounter repeats the static timing analysis to report the new, possibly reduced longest path delay of the target design.

When the desired performance constraint is met or if the delay saturates at a certain value after a number of post-route optimization iterations, the layout is finalized and can then be exported to different formats, while the design can be saved as an *.enc* file, which is the format of the Encounter's designs. In our flow, we exported the design into the Verilog format by selecting "Design->Save->Netlist" from the menu and the verilog netlist's name can be given in the appearing window. The design is saved as an *.enc* file by selecting "Design->Save Design As->SoCE". The Verilog netlist should be exported, so that it can be used as input in the PrimeTime PX power analysis tool for the power estimation process, as well as in the Cadence ETS timing analysis engine.

Regarding the aforementioned tools, PrimeTime PX is used for power analysis, which is performed by reading the design's post-layout Verilog netlist and the *.vcd* file containing the design's activity, obtained from ModelSim, using the command *read_vcd* as follows.

```
$ read_vcd –strip_path $testbench/$top_design $top_design_activity.vcd
$ update_power
```

In the *read_vcd* command, the *strip_path* flag defines the hierarchy level of the target design in the testbench used for the simulation. The *$top_design_activity.vcd* file is the *.vcd* containing the activity of all the signals of the target design. The signal nets are written with the hierarchical names having the "*$testbench/*" as prefix, as the top design is instantiated into the testbench and therefore it is one hierarchical level below. The power analysis is performed through the $*update_power* command.

Apart from the power analysis, which is a necessary input to HotSpot, the timing reports are equally important, as they reveal the initial and the final timing slack. These are derived from the ETS, which is the timing analysis engine of SoC Encounter. Unlike PrimeTime PX, which requires the used standard-cell libraries to be listed in the *.synopsys_pt.setup,* the Design Compiler requires to read them from the *.synopsys_dc.setup*. The library definition is rather simple in the ETS and requires the .lib files of the standard-cell and macro (SRAM) libraries to be read. The design is read in its Verilog format, along with the parasitic delay information, provided either from the design's SPEF or its SDF file. The Tcl script template that implements all the aforementioned tasks is provided in the Appendix, named as *ets.tcl*. The corresponding script of PrimeTime PX, named as *leon3mp_px.tcl* is also included.

The above steps complete the physical implementation flow, as the layout is now constructed and its relative design formats can be used for extracting the design's timing and power. It must be noted that the current section is not a complete and detailed guide for neither the SoC Encounter, nor the Design Compiler or any other tools. A detailed description of the features of the tools used in our flow is provided in the related *.pdf* documents referenced in the corresponding section. Among these documents, we provide the Mentor Graphics ModelSim user's guide, as a reference for the RTL and the post-layout simulation of VHDL-based or standard-cell-based designs, which is summarized in the next section.

5.3 Simulation and Software tools

The extraction of the design's power profile and the thermal simulation in HotSpot require the a priori knowledge of the switching activity of the signal nets of the layout's netlist, while a certain application is being executed by either both or by a single of the two Leon3 processors of our MP-SoC system. For the switching activity extraction, the design's post-layout Verilog netlist must be simulated and the values of all the signals can be dumped into the design's Value Change Dump file (*.vcd*), so that the power analysis tool can compute both the dynamic and the static power for the simulated time interval.

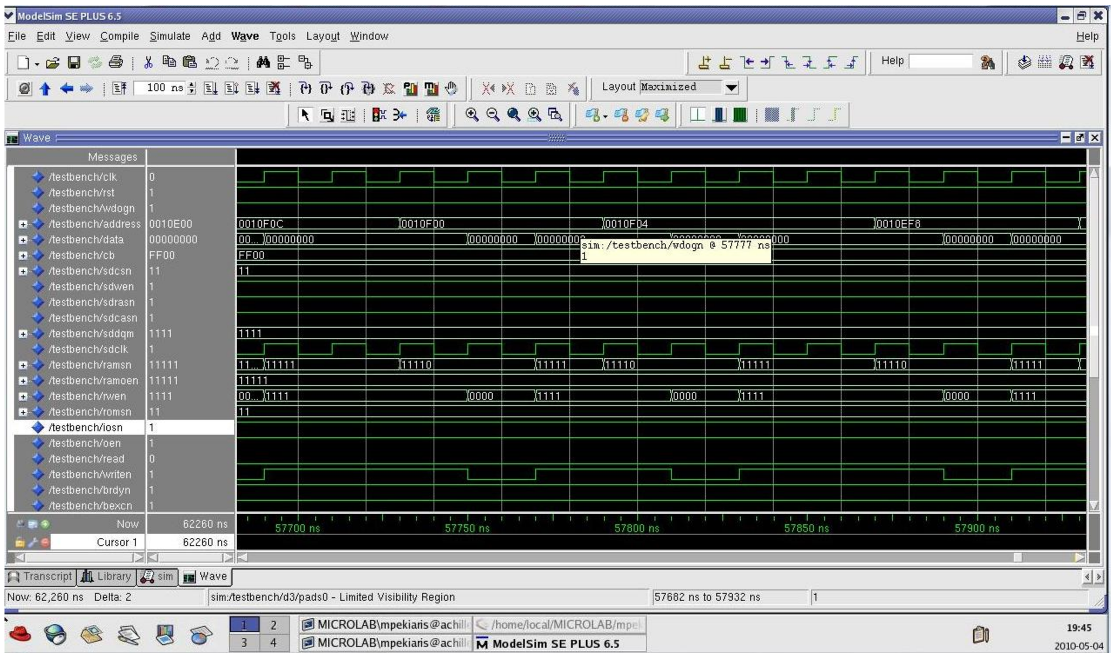
In this work, we have used Mentor Graphics ModelSim [25] for the post-layout, as well as for the RTL simulation of the Leon3-based MP-SoC design. As the RTL code is used only for the functional verification of the design before its implementation, we will not elaborate any further on this stage of simulating the design, as the obtained *.vcd* cannot be used for the post-layout netlist's power analysis. Hence, in this section, we will address the steps for performing a post-layout simulation of the design's Verilog netlist together with the VHDL files of the external Programmable ROM (Prom), SRAM and SDRAM components provided by Gaisler.

The testbench used in the simulations is also in VHDL and it instantiates the target design, along with the Prom and either the SRAM or the SDRAM, depending on the corresponding parameter values on the *config.vhd* package, defined in the respective section that was previously presented. The simulation is performed in interactive mode, after compiling the standard-cell library Verilog simulation models, composing the design's netlist, using the following command.

```
$ vlog $top_design.v –v $standard_cell_library.v
```

The above command first compiles the standard-cell library's Verilog simulation models, and then the *$top_design.v*, which is the post-layout netlist of the Leon3-based MP-SoC design. The external memory files and the packages required for the debugging messages to be displayed are compiled next through the *vcom* command, used to compile VHDL files in ModelSim. It is noted that during these experiments, we were obliged to follow a mixed-language simulation approach, as the external memories were written in VHDL. ModelSim's simulator is invoked through the *vsim* command, which is used as shown in the Appendix, where the whole script, named *run_leon3mp_vlog*, is presented. During the simulations, the waveforms of the main input and output signals, as well as of internal nets of the design, were created on-the-fly, in order to monitor the simulation's progress. A screenshot of the waveforms produced during a post-layout simulation in ModelSim is shown in Figure 5.12.

As far as the simulation is concerned, there were two problems that we had to overcome in order to obtain a functional post-layout Verilog netlist operating at a clock frequency close to the maximum of 200MHz, which corresponds to a clock period of 5ns.



Messages	
/testbench/clk	0
/testbench/rst	1
/testbench/wdogn	1
/testbench/address	0010E00
/testbench/data	00000000
/testbench/cb	FF00
/testbench/sdcsn	11
/testbench/sdwen	1
/testbench/sdrasn	1
/testbench/sdcasn	1
/testbench/sddqm	1111
/testbench/sdclk	1
/testbench/ramsn	11111
/testbench/ramoen	11111
/testbench/rwen	1111
/testbench/romsn	11
/testbench/iosn	1
/testbench/oen	1
/testbench/read	0
/testbench/writen	1
/testbench/brdyn	1
/testbench/bexcn	1
Now	62260 ns
Cursor 1	62260 ns

sim:/testbench/wdogn @ 57777 ns
1

Now: 62,260 ns Delta: 2 sim:/testbench/d3/pads0 - Limited Visibility Region 57682 ns to 57932 ns 1

Figure 5.12. A simulation waveform sample of the design in ModelSim.

The selection of such an operating frequency for the simulation was necessary, in order to obtain a realistic figure for the design's power dissipation, as it would be otherwise underestimated. Therefore, we had to modify the simulation models of the external Prom and SRAM memory provided by Gaisler, so that their response time is less than 5ns, specifically 4ns in our experiments. The simulation's duration was set to approximately *50ms* in ModelSim, corresponding to about an hour of simulation time. This was the threshold beyond which the size of the *.vcd* file would become extremely large, introducing problems to its use in PrimeTime PX to perform the power analysis.

The design's operation begins by reading the contents of the Prom, which start at the address 0x00000000 and include the necessary commands that initialize the Leon3 processors and various system registers. Among those, we can distinguish the processor's Status and the Interrupt Request register, as well as others, which regulate the functionality of the Memory controller's module. A detailed description of the commands contained in the Prom is included in the corresponding assembly file, provided by Gaisler along with the RTL code's files. After the execution of the initialization commands, the Prom's code performs a jump operation to the SRAM's address, so that the execution of the user's application can start. The application should be loaded into the SRAM, which is mapped to the address 0x40000000, while the SDRAM itself, which is not used in the simulations as external memory, starts at 0x60000000. Both the application and the Prom's contents are loaded into SRECORD files (*.srec*), written into a format that resembles the hexadecimal. In the performed simulations, which include an application initializing the system with the two processors, and a matrix multiplication, the Prom's *.srec* file used, was the one included in Gaisler's suite.

The *.srec* files of the aforementioned applications have been generated through the Leon3 BCC cross-compiler, provided among the other Gaisler's software support tools, by using the proper commands, explained analytically in the BCC's compiler user guide [26], as well as in the referenced diploma thesis [27]. For the generation of the *.srec* files, it is required to produce the executable of the application from the cross-compilation of the BCC, before using the *sparc-elf-objcopy* command, which converts the generated *.exe* into a *.srec* file. The produced *.srec* file is then given as a parameter to the testbench and its contents are read by Gaisler's behavioral external Prom, SRAM and SDRAM components. Consequently, the designer has to change only the name of the *.srec* file that corresponds to the application to be executed, in order to extract an alternate switching activity from a different application and therefore, new power and temperature profiles for the analyzed design's layout.

The presentation of the post-layout simulation tools and of the software support for the generation of the simulated applications concludes this chapter, which was dedicated to the EDA design flow for both the implementation of the layouts and for the verification of their functionality. In this chapter of the thesis, we attempted to introduce the different physical implementation scenarios of the Leon3 design that were implemented and used as testbenches. Meanwhile, we also attempted to explain our motivation for the selection of these specific place-and-route scenarios, by trying to justify the anticipated impact of congestion-driven and timing-driven placement and routing on EM and TDDB. The next chapter presents the results obtained from our interconnect reliability framework using the previously described layouts of the Leon3 MP-SoC system, and evaluates which of the explored place-and-route approaches should be prefferred for the mitigation of the EM and/or TDDB. The evaluation of the prominent physical implementation strategies is followed by possible extensions and improvements on the proposed reliability analysis flow, as well as by solutions that may mitigate the problems induced by the two studied interconnect wear-out mechanisms.

6

Experimental results & conclusions

6.1 Experimental results

Experiments were conducted on five flavors of the Leon3 MP-SoC platform presented in the previous chapter, produced using a variety of placement and routing strategies. To be precise, the five designs, each named after its placement and routing strategy, are the following:

- Timing driven placement & timing driven routing (TP-TR)
- Timing driven placement & neutral routing (TP-NR)
- Neutral placement & neutral routing (NP-NR)
- Congestion driven placement & neutral routing (CP-NR)
- Congestion driven placement & congestion driven routing (CP-CR)

Their results were then analyzed in order to discover any trends and determine the design strategies that minimize the impact of each reliability-threatening phenomenon. Additionally, an exploration was conducted to determine the thermal conditions under which each phenomenon's impact started to affect the system early in its lifetime. The three thermal scenarios tested are presented below:

- Standard environmental conditions with the ambient temperature at 45^{o}C, absence of heatspreader and heatsink.
- Constant and uniform temperature of all hierarchical units at 100^{o}C.
- Military environmental conditions with the ambient temperature at 85^{o}C, absence of heatspreader and heatsink.

The power profile of the design, used to estimate the temperature of each hierarchical unit in HotSpot, was extracted from a post-layout simulation at a frequency close to the maximum achievable for each design. The thermal simulations resulted in a mean temperature of 67.97^{o}C under the standard thermal conditions and of 102.97^{o}C under the military thermal conditions. Moreover, the highest temperatures of all versions of the design were consistently observed in the cache memories and in the Memory Management Units (MMUs) of both processor cores, which indicates the presence of hotspots at these specific areas of the design.

6.1.1 EM results

The EM flow was executed for all five designs and under all three thermal scenarios described above, and the parameters that quantify the impact of the phenomenon are summarized in the tables below. All experiments were conducted using both current estimation methods presented in Chapter 3 and further elaborated in Chapter 4. The results derived from the more accurate method, which is based on Spice simulations, are denoted by I_{spice}, whereas the results derived from the approximate method through the analytical formula, are denoted by I_{net}.

The tables can be divided in groups of three. The first table of each group lists all EM parameters for a specific physical implementation scenario of the design, whereas the other two tables only show those parameters that change due to the different thermal conditions. So, the first table records the minimum Blech length, the number of wires that are shorter than the Blech length, the minimum t_{50}, the maximum EM resistance slope and the maximum current density that was calculated for the examined paths. The other two tables only show the minimum t_{50} and the maximum EM slope resistance, as the rest of the parameters are unaffected by temperature.

The results reveal certain trends on both the placement and routing strategy and on the temperature, regarding the EM. The impact of the phenomenon appears to be smaller in the timing-driven designs in contrast to the congestion-driven ones. This is justified as the former strategy leads to shorter wires in order to minimize the propagation delay, whereas the later leads to longer wires as it performs detours in order to decrease the congestion of the design, resulting in more wires being in excess of the Blech length. In addition to that, while performing a timing-driven routing, Nanoroute is likely to perform a layer assignment favoring the usage of higher metal layers, as they are wider and therefore have less resistance, leading to reduced interconnect delay. The usage of wider metal layers in the timing-driven approach, compared to the congestion-aware, reduces the impact of EM on the corresponding wires, as the current density is reduced.

Table 6.1. NP-NR, standard thermal conditions.

I_{wire} estimation	Blech length (min) (um)	# wires > Blech length	t_{50} (min) (years)	EM slope (max) (Ohms/year)	max current density (A/um2)
I_{net}	4.60899424708	396 (in 14 nets)	16.46607558	13.058981204	0.0802778177114
I_{spice}	28.9616613419	9 (in 9 nets)	144.954540306	2.18529108558	0.0127755102041

Table 6.2. NP-NR, 100^{o}C constant and uniform.

I_{wire} estimation	t_{50} (min) (years)	EM slope (max) (Ohms/year)
I_{net}	1.18872489664	180.891451094
I_{spice}	10.7880294778	28.7872870135

Table 6.3. NP-NR, military thermal conditions.

I_{wire} estimation	t_{50} (min) (years)	EM slope (max) (Ohms/year)
I_{net}	0.952917020983	225.654665379
I_{spice}	8.60708917216	36.8031349983

Table 6.4. CP-NR, standard thermal conditions.

I_{wire} estimation	Blech length (min) (um)	# wires > Blech length	t_{50} (min) (years)	EM slope (max) (Ohms/year)	max current density (A/um2)
I_{net}	5.89985269566	380 (in 12 nets)	22.1448176016	10.2017410174	0.0627134301628
I_{spice}	26.8393782383	11 (in 10 nets)	157.144463704	1.94640662026	0.0137857142857

Table 6.5. CP-NR, 100^{o}C constant and uniform.

I_{wire} estimation	t_{50} (min) (years)	EM slope (max) (Ohms/year)
I_{net}	1.598686699	141.313300593
I_{spice}	9.84647766899	31.0635980473

Table 6.6. CP-NR, military thermal conditions.

I_{wire} estimation	t_{50} (min) (years)	EM slope (max) (Ohms/year)
I_{net}	1.281554522	176.282546058
I_{spice}	8.86140780057	34.5167530234

Table 6.7. CP-CR, standard thermal conditions.

I_{wire} estimation	Blech length (min) (um)	# wires > Blech length	t_{50} (min) (years)	EM slope (max) (Ohms/year)	max current density (A/um2)
I_{net}	5.98325789221	382 (in 13 nets)	22.5210156338	10.0595311662	0.061839219814
I_{spice}	26.8393782383	10 (in 9 nets)	157.144463704	1.94640662026	0.0137857142857

Table 6.8. CP-CR, 100^{o}C constant and uniform.

I_{wire} estimation	t_{50} (min) (years)	EM slope (max) (Ohms/year)
I_{net}	1.6258453237	139.343426684
I_{spice}	9.84647766899	31.0635980473

Table 6.9. CP-CR, military thermal conditions.

I_{wire} estimation	t_{50} (min) (years)	EM slope (max) (Ohms/year)
I_{net}	1.30332567848	173.825209158
I_{spice}	8.86140780057	34.5167530234

The results reveal a larger number of wires being in excess of the Blech length in the congestion-driven designs, as a result of the wires being longer on average. Despite the fact that less EM-susceptible wires are found in the timing-driven designs, as they are usually shorter and thus less likely to exceed the Blech length, they tend to develop EM earlier. This happens because the timing-driven designs are pushed harder in order to minimize the delays and satisfy the desired performance constraint, possibly leading to the up-sizing of cells with large fanout. The use of wider transistors at the outputs of these cells enables them to reduce the charge and discharge times of their loads by providing higher currents.

As a result, the wires that exceed the Blech length despite being relatively short, tend to develop EM considerably earlier, because they are driven by standard-cells of high driving strength, compared to those of the congestion-driven case study. However, the number of wires over the Blech length in the congestion-driven designs is larger, as the standard-cells are scattered relatively uniformly across the area of the chip and therefore longer wires are needed to connect them, in comparison to the timing-driven designs. Thus, in the specific case study of the presented Leon3-based MP-SoC platform, the congestion-driven place&route strategy is likely to produce physical designs whose interconnects are more seriously affected by EM, compared to the timing-driven strategy. This is revealed by the results through the number of wires with t_{50} smaller than the target lifetime, instead of through the final timing analysis reports, as the number of affected wires is inadequate for any measurable performance drift to be observed.

Table 6.10. TP-NR, standard thermal conditions.

I_{wire} estimation	Blech length (min) (um)	# wires > Blech length	t_{50} (min) (years)	EM slope (max) (Ohms/year)	max current density (A/um2)
I_{net}	4.7051875472	110 (in 7 nets)	16.8793233164	2.7920021547	0.0786366103983
I_{spice}	35.6538839725	1 (in 1 net)	165.189464365	1.959824545	0.0103775510204

Table 6.11. TP-NR, 100^{o}C constant and uniform.

I_{wire} estimation	t_{50} (min) (years)	EM slope (max) (Ohms/year)
I_{net}	1.21855822701	177.193289125
I_{spice}	13.8446562031	23.3839224383

Table 6.12. TP-NR, military thermal conditions.

I_{wire} estimation	t_{50} (min) (years)	EM slope (max) (Ohms/year)
I_{net}	0.976832300612	221.041360015
I_{spice}	9.80859548853	33.0059861492

Moreover, the results highlight the important role that temperature plays, as EM is exponentially dependent on it. Hardly any wires have a t_{50} that is less than 10 years in standard thermal conditions. But as the temperature rises (100^{o}C constant and uniform, and even higher under the military thermal conditions), the phenomenon seems to be significantly accelerated and amplified, as the same wires now start to develop EM-induced voids before completing 10 years of operation. Even a relatively small temperature difference of 5-10^{o}C, such as those observed between the 100^{o}C and the military thermal conditions experiments, is able to sink the t_{50} by a significant number of years.

Table 6.13. TP-TR, standard thermal conditions.

I_{wire} estimation	Blech length (min) (um)	# wires > Blech length	t_{50} (min) (years)	EM slope (max) (Ohms/year)	max current density (A/um2)
I_{net}	2.95370462525	177 (in 7 nets)	9.65390855993	20.3773826021	0.125266418598
I_{spice}	16.4146672703	5 (in 4 nets)	85.0795302106	3.2583632715	0.0225408163265

Table 6.14. TP-TR, 100^{o}C constant and uniform.

I_{wire} estimation	t_{50} (min) (years)	EM slope (max) (Ohms/year)
I_{net}	0.696938465956	282.265074954
I_{spice}	5.45798653697	50.7916270071

Table 6.15. TP-TR, military thermal conditions.

I_{wire} estimation	t_{50} (min) (years)	EM slope (max) (Ohms/year)
I_{net}	0.55868647882	352.11410297
I_{spice}	4.81781969086	57.5405544796

Despite finding EM-susceptible wires that present problems before 10 years of operation, the static timing analysis shows that they have negligible impact on the system's timing or even no impact at all. This happens because the problematic wires that are on the same register-to-register path are very few and therefore, the total resistance rise due to EM is not enough to intoduce any measurable delay overhead. The existence of more such wires in interconnects that belong to the same path, would cause a significant total resistance and consequently delay rise. This calls for either a type of design that features longer wires, or a high performance design that would present both high current densities and thus small Blech lengths, classifying more wires as susceptible to EM, and high power dissipation and thus high temperature that would cause the discovered wires to develop problems early in the system's lifetime.

In conclusion, EM is favored by congestion-driven placement and routing in designs, whereas timing-driven placement and routing mitigate its impact. Furthermore, the impact of the underlying wear-out becomes significant enough only when the die's temperature reaches a relatively high level, as t_{50} and R_{slope} are exponentially dependent on temperature. Consequently, the gradual impact of EM on the performance of digital systems should rather be considered in high performance, "energy-hungry" designs that operate in higher frequencies, than in low-power embedded platforms like the Leon3 design used in these experiments.

6.1.2 TDDB results

The TDDB flow was executed for all five designs and under all three thermal scenarios described above, and the effect of the phenomenon on the system's timing over time is summarized in the tables below. Each table contains the latest arrival time of data in the most timing-critical path of the design after a certain number of operating years, as well as the available slack in the parentheses. A negative slack means that the delay is so large that it causes certain data not to arrive within the same clock cycle, which suggests a timing violation. The presented data are recorded at the beginning of the system's lifetime and after 1, 3, 5, 8 and 10 years of operating time respectively. This enables us to predict the time period in the system's lifetime, when a timing violation is first introduced and at the same time the required performance tradeoff in order to prolong the system's lifetime by a certain amount of time. If, for instance, a timing violation of *x* ns is detected between the first 5 and 8 years of operation, an increase in the clock period by *x* ns will extend the system's lifetime to 8 years. The delay overhead due to the impact of the TDDB-induced inter-metal leakage current is calculated as the difference in arrival times estimated by the static timing analyses performed using the pre- and post-annotated versions of the SDF file. This delay evaluation metric was chosen, because it provides a clear view on the actual timing deterioration of the system, as it is revealed in the graphs that follow each set of result tables.

It must be noted that the remark "*Wire charge failure*" appears in two tables. This means that the leakage current due to the TDDB is so strong in one or more interconnects, that the driving cell is unable to charge them. The inability to charge a wire leads to a permanent failure in the system, as it renders one or more particular interconnects useless. The code of that wire in SoC Encounter, which is retrievable through the proper DBAccess command, described in Chapter 4, is reported by our script, enabling us to identify it.

The results reveal certain trends on both the placement and the routing strategy, as well as on the temperature, regarding the evolution of TDDB. The impact of the phenomenon appears to be smaller in the congestion-driven designs in contrast to their timing-driven counterparts. This can be explained as the former strategy leads to longer wires, as it performs detours in order to decrease the congestion of the design, whereas the later leads to shorter wires in order to minimize the propagation delay. As a result, the impact of the TDDB on the timing of the system is smaller in the congestion-driven designs, which contain fewer adjacent wires in close distance, than the timing-driven designs, in which both cells and wires are stacked close to each other in order to reduce the interconnects' delays. But this undermines the reliability of the system as the TDDB-induced delay overhead is enhanced in congested areas, according to the experimental results. Furthermore, high temperature accelerates the evolution and amplifies the impact of the TDDB, increasing the delay overhead that it introduces to the system, but the phenomenon has significant impact even at relatively low temperatures. However, the fact that the extrapolation of the inter-metal leakage measurements from accelerated to operating conditions may introduce a percentage of uncertainty in the accuracy of the results, should be kept in mind.

Table 6.16. CP-NR, standard thermal conditions.

Operating time	Delay due to TDDB only (ns)	delay change due to TDDB (ps)
Initial	6.273 (0.719)	0
1 year	6.353 (0.639)	80
3 years	6.514 (0.478)	241
5 years	6.675 (0.317)	402
8 years	6.916 (0.075)	644
10 years	7.079 (-0.088)	807

Table 6.17. TP-NR, standard thermal conditions.

Operating time	Delay due to TDDB only (ns)	delay change due to TDDB (ns)
Initial	3.687 (0.27)	0
1 year	3.787 (0.17)	100
3 years	3.984 (-0.027)	297
5 years	4.196 (-0.239)	509
8 years	4.552 (-0.595)	865
10 years	4.852 (-0.901)	1171

Table 6.18. NP-NR, standard thermal conditions.

Operating time	Delay due to TDDB only (ns)	delay change due to TDDB (ns)
Initial	4.53 (0.064)	0
1 year	4.585 (0.009)	55
3 years	4.715 (-0.121)	185
5 years	4.866 (-0.271)	335
8 years	5.146 (-0.551)	615
10 years	5.406 (-0.811)	875

Table 6.19. CP-CR, standard thermal conditions.

Operating time	Delay due to TDDB only (ns)	delay change due to TDDB (ns)
Initial	4.665 (0.555)	0
1 year	4.739 (0.48)	75
3 years	4.886 (0.334)	221
5 years	5.036 (0.184)	371
8 years	5.266 (-0.046)	601
10 years	5.420 (-0.201)	756

Table 6.20. TP-TR, standard thermal conditions.

Operating time	Delay due to TDDB only (ns)	delay change due to TDDB (ns)
Initial	3.264 (0.303)	0
1 year	3.346 (0.221)	82
3 years	3.515 (0.052)	251
5 years	3.687 (-0.121)	424
8 years	3.976 (-0.409)	712
10 years	4.205 (-0.638)	941

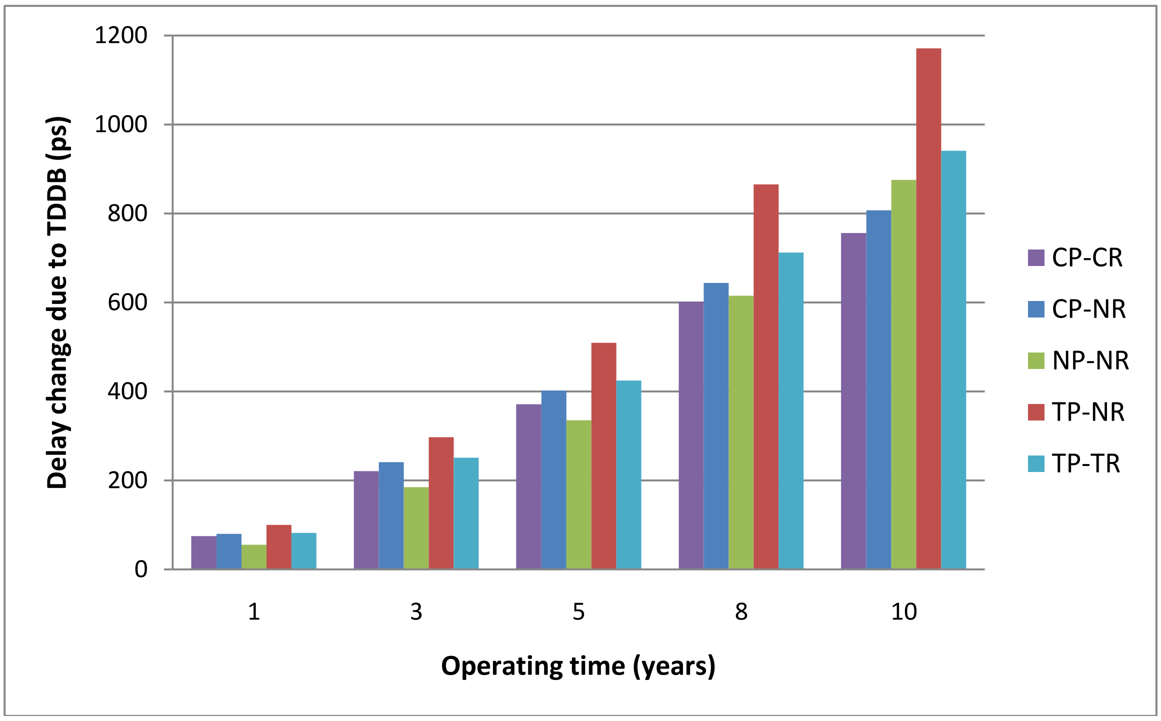


Figure 6.1. Progressive impact of TDDB on timing (standard thermal conditions).

The results clearly and consistently classify the impact of the TDDB on the timing-driven designs as larger than on the congestion-driven counterparts, under all explored thermal scenarios. This is justified by the different placement styles. In the case of the timing-driven approach, cells are placed close to each other, which minimizes the delays, but at the same time increases the congestion of the layout's bins, in specific areas. In the case of congestion-driven placement, the density of cells located into the bins is more balanced in order to avoid design-rule violations and crosstalk while routing the design. Hence, timing-driven placement increases the probability of wires having more adjacent wires and also in closer distance (spacing), compared to the congestion-driven approach. Therefore, designers should be aware of this trade-off, in order to evaluate the pros and cons of each placement strategy and balance performance and congestion.

The routing style, namely timing-driven, congestion-driven or neutral, may be the cause of rather contradicting results for TDDB in the timing-driven design, compared to its congestion-driven or neutral counterpart. The fact that timing-driven routing reduces the impact of the underlying phenomenon could be due to a different layer assignment by Nanoroute, favoring the usage of higher metal layers as they are wider and therefore have less resistance, which results in reduced interconnect delay. But at the same time, this change in layer assignment actually reduces congestion, as formerly adjacent wires could be moved to different metal layers, which eliminates their impact. Consequently, cell placement is not the only factor affecting the impact of TDDB, as routing may change the distribution of wires in metal layers, depending on the timing constraints and the chip's size.

These conclusions, as well as those extracted from the experiments evaluating EM, should be cross-checked by performing experiments on several testbenches, of different logic styles and implementations, in order to ensure their global validity. In order to accomplish that, complex designs like MP-SoC platforms with processors interconnected through cross-bars or Networks-on-Chip should be examined. The proposed framework should focus on their interconnection backbone and explore alternative implementation strategies, favoring either performance or congestion, in order to draw secure conclusions about a reliability-aware placement-and-routing strategy. These conclusions could then be back-annotated to higher levels of design abstraction, to aid in the construction of a reliability-aware design flow, by taking advantage of the known interdependence between performance and reliability. This feedback could be used to provide a range of implementation strategies, possibly *given as Pareto curves between performance and reliability*, which concurrently satisfy various design constraints, while also diminishing the threats of system-level timing failures due to EM or TDDB and therefore guaranteeing the desired system's lifetime.

Table 6.21. CP-NR, 100^{o}C constant and uniform.

Operating time	Delay due to TDDB only (ns)	delay change due to TDDB (ps)
Initial	6.273 (0.719)	0
1 year	6.364 (0.627)	92
3 years	6.548 (0.444)	275
5 years	6.732 (0.260)	459
8 years	7.009 (-0.017)	736
10 years	7.213 (-0.221)	940

Table 6.22. TP-NR, 100^{o}C constant and uniform.

Operating time	Delay due to TDDB only (ns)	delay change due to TDDB (ns)
Initial	3.687 (0.27)	0
1 year	3.801 (0.156)	114
3 years	4.027 (-0.070)	340
5 years	4.276 (-0.319)	589
8 years	4.709 (-0.758)	1028
10 years	5.157 (-1.207)	1477

Table 6.23. NP-NR, 100^{o}C constant and uniform.

Operating time	Delay due to TDDB only (ns)	delay change due to TDDB (ns)
Initial	4.53 (0.064)	0
1 year	4.593 (0.001)	63
3 years	4.746 (-0.152)	216
5 years	4.925 (-0.331)	395
8 years	5.285 (-0.690)	754
10 years	5.813 (-1.219)	1283

Table 6.24. CP-CR, 100^{o}C constant and uniform.

Operating time	Delay due to TDDB only (ns)	delay change due to TDDB (ns)
Initial	4.665 (0.555)	0
1 year	4.750 (0.470)	85
3 years	4.917 (0.302)	253
5 years	5.090 (0.129)	426
8 years	5.354 (-0.134)	689
10 years	5.533 (-0.313)	868

Table 6.25. TP-TR, 100^{o}C constant and uniform.

Operating time	Delay due to TDDB only (ns)	delay change due to TDDB (ns)
Initial	3.264 (0.303)	0
1 year	3.358 (0.209)	94
3 years	3.551 (0.015)	288
5 years	3.751 (-0.184)	487
8 years	4.103 (-0.537)	840
10 years	4.390 (-0.824)	1127 (Wire charge failure)

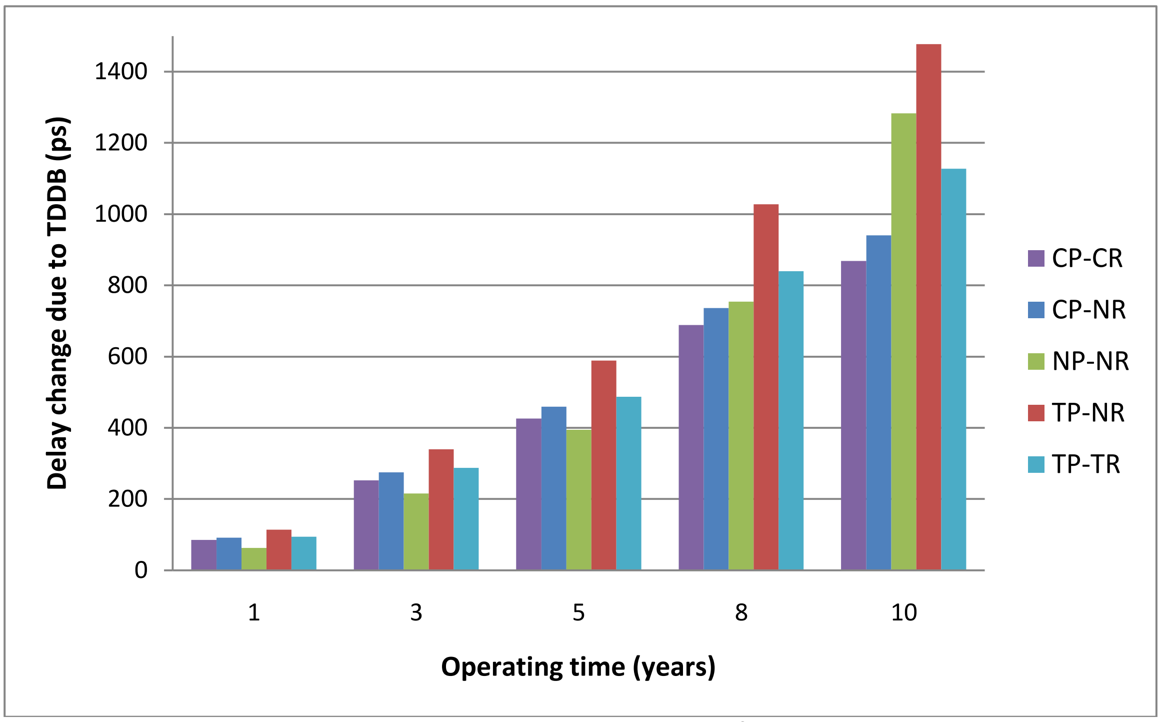


Figure 6.2. Progressive impact of TDDB on timing (100^{o}C constant and uniform).

The effect of temperature on the underlying phenomenon is definite as well, namely rise of temperature leads to rise in TDDB's impact on timing. This relation is also confirmed by the following 3D plot produced using data from the TDDB model:

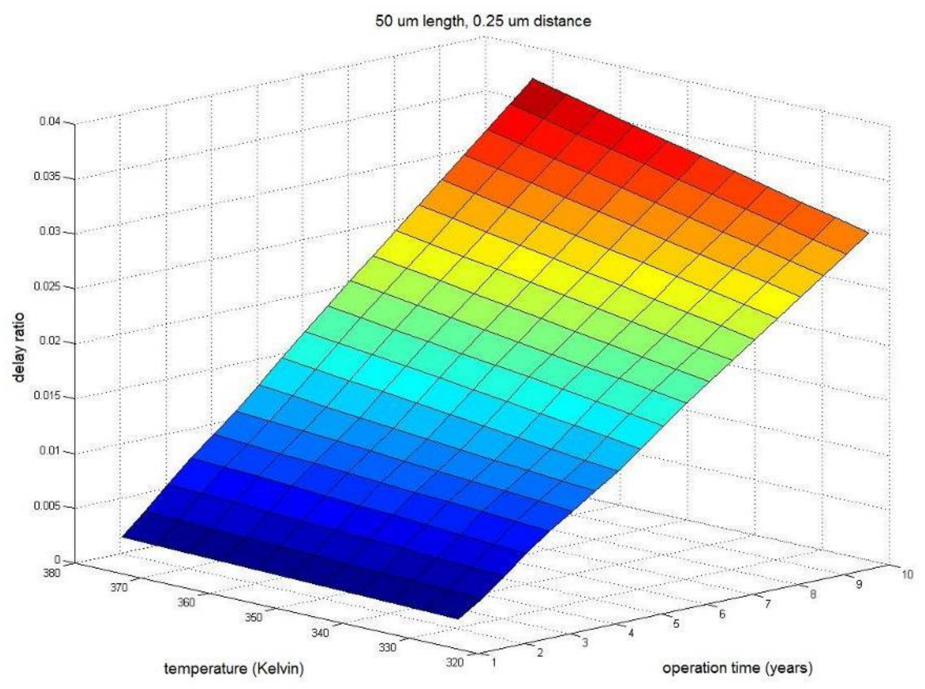


Figure 6.3. Delay impact on a wire due to TDDB, depending on temperature and operation time.

Table 6.26. CP-NR, military thermal conditions.

Operating time	Delay due to TDDB only (ns)	delay change due to TDDB (ps)
Initial	6.273 (0.719)	0
1 year	6.365 (0.626)	93
3 years	6.551 (0.441)	278
5 years	6.737 (0.254)	465
8 years	7.017 (-0.026)	745
10 years	7.226 (-0.234)	953

Table 6.27. TP-NR, military thermal conditions.

Operating time	Delay due to TDDB only (ns)	delay change due to TDDB (ps)
Initial	3.687 (0.27)	0
1 year	3.802 (0.155)	115
3 years	4.031 (-0.074)	344
5 years	4.284 (-0.326)	596
8 years	4.725 (-0.775)	1045
10 years	5.193 (-1.243)	1513

Table 6.28. NP-NR, military thermal conditions.

Operating time	Delay due to TDDB only (ns)	delay change due to TDDB (ps)
Initial	4.53 (0.064)	0
1 year	4.594 (0.001)	63
3 years	4.749 (-0.155)	219
5 years	4.931 (-0.336)	400
8 years	5.298 (-0.703)	767
10 years	5.868 (-1.273)	1337

Table 6.29. CP-CR, military thermal conditions.

Operating time	Delay due to TDDB only (ns)	delay change due to TDDB (ps)
Initial	4.665 (0.555)	0
1 year	4.751 (0.469)	86
3 years	4.920 (0.299)	256
5 years	5.095 (0.124)	431
8 years	5.362 (-0.142)	697
10 years	5.543 (-0.324)	879

Table 6.30. TP-TR, military thermal conditions.

Operating time	Delay due to TDDB only (ns)	delay change due to TDDB (ps)
Initial	3.264 (0.303)	0
1 year	3.359 (0.208)	95
3 years	3.555 (0.012)	291
5 years	3.757 (-0.190)	493
8 years	4.115 (-0.549)	852
10 years	4.408 (-0.842)	1145 (Wire charge failure)

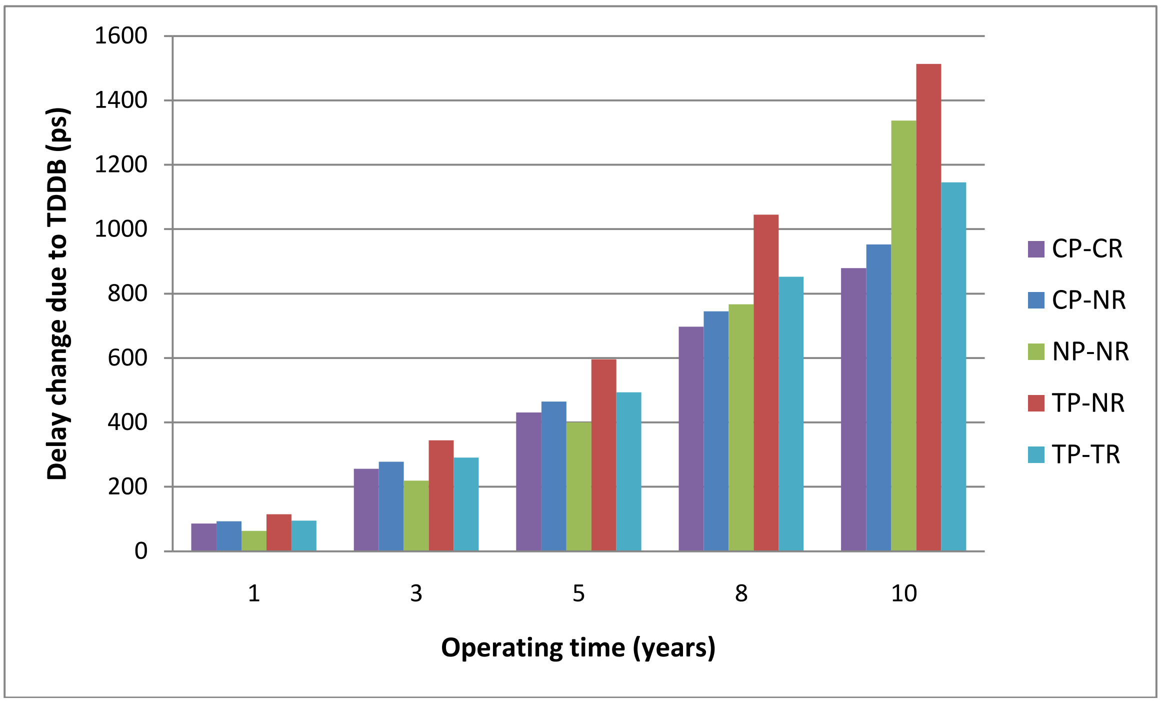


Figure 6.4. Progressive impact of TDDB on timing (military thermal conditions).

In conclusion, timing-driven placement and routing enhance the evolution of TDDB and lead to greater impact on the design's performance, whereas congestion-driven placement and routing mitigate its impact and thus considerably extend the system's lifetime. Furthermore, high temperature amplifies and accelerates the design's performance degradation due to the underlying wear-out mechanism, although significant impact is observed even at relatively lower temperatures.

6.2 Conclusions, hints for future extensions and proposed solutions

6.2.1 Conclusions

The result of this work is an automated, extensible and customizable interconnect reliability analysis framework that is compatible with state-of-the-art industrial and academic EDA tools. As a consequence, its features may prove to be useful for designers and engineers working on the reliability field, as it enables the accurate estimation of the progressive impact that interconnect reliability wear-out mechanisms have on a design's timing and can predict whether and for how long the system will function correctly under the target performance specifications. Improved accuracy could be achieved through the use of more accurate models for the examined wear-outs, perhaps in conjunction with extended sets of experimental results under stress conditions, mainly regarding TDDB, in order to reduce the error that is introduced by their extrapolation.

Furthermore, the experiments we conducted in the context of this work to compare the effect different placement and routing strategies have on the two examined wear-outs, namely EM and TDDB, lead to interesting conclusions.

Specifically, on one hand, the TDDB seems to be mitigated by congestion-driven placement and routing, as this strategy reduces the design components' congestion, resulting in less adjacent wires. However, on the other hand, the EM seems to be favored by congestion-driven placement and routing, because longer wires are generated on average and more of them exceed the Blech length. Nevertheless, the intensity of the EM heavily depends on the specific design and implementation technology, which suggests that this conclusion on the EM, which is only based on the current results, could be potentially overthrown under different circumstances.

One case, where the impact of the EM is anticipated to be greater, is that of a high-performance design as a test platform, instead of the currently used Leon3-based design. This shift would cause higher temperatures to be achieved even at standard thermal conditions, which would significantly amplify the underlying phenomenon's intensity, as the results presented in the previous section indicate. Simultaneously, the current density would increase in order for the tighter target timing constraints to become satisfied, which would further amplify the phenomenon. A potential candidate for the required shift to high performance computing designs is the OpenSparc microprocessor [28] platform.

Another case, in which we could expect a potentially greater impact from EM, is the migration from the 45nm node to the new state-of-the-art, such as the 32nm and very soon the 22nm [16]. As we argued in the introduction of this thesis, the scaling of manufacturing technology enhances the significance of progressive wear-out phenomena and consequently the resistance rise of wires due to EM. This can be justified by the fact that the current density increases with the technology scaling, because dimensions keep shrinking, whereas the supply voltage saturates. Hence, EM is expected to demonstrate a more aggressive impact on a design's interconnects, leading to greater delay degradation in the affected wires. The aforementioned allegation is confirmed by the following experiment, which is based on appropriate scaling to a 32nm technology. The scaling performed on both the transition times of cells and on the physical dimensions of wires has been strongly based on the assumptions shown in [9]. Even if the referenced paper is addressing scaling for technology nodes not in the deep-deep submicron regime, we selected this approach in order to obtain a rough estimation of the trend of current density and also of EM-related parameters, while moving from the 45nm to the 32nm node. Regarding the implementation of this experiment, the characteristics of some interconnects and cells were manually scaled in order to estimate the results the EM flow would produce for a 32nm technology and these are directly juxtaposed to the corresponding results in our 45nm technology below:

<u>45nm:</u>

```
###################################################################################
Net:            core0/leon3core0/leon3s0_1/p0/c0mmu/dcache0/FE_OFN80_n3802
Temperature(C): 104.65
R  Blech_length(um)  t50(years)     R_slope(Ohm/year)  R_jump(Ohm)   J(A/um2)
11 37.7826404085     10.5154865958  31.1463568411      345.708101423 0.00979285714286
###################################################################################
Net:            core0/leon3core0/ahbctrl0/n395          Temperature(C): 98.39
R  Blech_length(um)  t50(years)     R_slope(Ohm/year)  R_jump(Ohm)   J(A/um2)
56 32.8739800544     14.1799040352  22.4633918955      345.708101423 0.0112551020408
###################################################################################
Net:            core0/leon3core0/leon3s0_1/p0/iu0/FE_OFCN12273_n2439
Temperature(C): 101.41
R  Blech_length(um)  t50(years)     R_slope(Ohm/year)  R_jump(Ohm)   J(A/um2)
13 40.7369958432     14.6205119727  22.741200493       345.708101423 0.00908265306122
###################################################################################
```

32nm:

```
##################################################################################
Net:            core0/leon3core0/leon3s0_1/p0/c0mmu/dcache0/FE_OFN80_n3802
Temperature(C): 104.65
R  Blech_length(um)  t50(years)     R_slope(Ohm/year)  R_jump(Ohm)    J(A/um2)
9  35.66791265       9.81322691786  32.9930044438      676.282285274 0.0103734693878
11 20.6939847049     5.10612634485  56.8663607971      676.282285274 0.0178795918367
##################################################################################
Net:            core0/leon3core0/ahbctrl0/n395          Temperature(C): 98.39
R  Blech_length(um)  t50(years)     R_slope(Ohm/year)  R_jump(Ohm)    J(A/um2)
56 16.4968152866     6.19912910104  44.7638580112      676.282285274 0.0224285714286
98 9.98898071625     3.39528282494  73.9275726026      676.282285274 0.0370408163265
##################################################################################
Net:            core0/leon3core0/leon3s0_1/p0/iu0/FE_OFCN12273_n2439
Temperature(C): 101.41
R  Blech_length(um)  t50(years)     R_slope(Ohm/year)  R_jump(Ohm)    J(A/um2)
13 21.5090758097     6.79394274518  43.0705716112      676.282285274 0.0172020408163
16 15.0831946755     4.43780061499  61.4198921304      676.282285274 0.0245306122449
##################################################################################
Net:            core0/leon3core0/ahbso_0__HRDATA__10_           Temperature(C): 102.97
R  Blech_length(um)  t50(years)     R_slope(Ohm/year)  R_jump(Ohm)    J(A/um2)
11 19.7645263273     5.46753255779  52.6216028321      676.282285274 0.0187204081633
##################################################################################
```

The observed impact seems to be significantly larger in the 32nm technology in comparison to the 45nm node. The number of EM-affected wires seems to approximately double, as more wires are classified as longer than the Blech length. These belong to interconnects that already presented milder problems in the 45nm technology and others that previously had no EM-related problems at all. The current density appears to approximately double as well, which in part leads to the above deterioration. The t_{50} sinks significantly, shifting the time instant that the resistance step occurs to before 10 years of operation, while the reduction of the Cu interconnect barrier's thickness almost doubles its value. The combination of a greater number of wires, a smaller t_{50} and a higher resistance rise, because of both R_{jump} and R_{slope}, increases the impact of the underlying phenomenon to a level that is highly likely to considerably affect the examined system's timing.

From a holistic point of view, the framework presented in this thesis is a useful as well as extensible tool for the estimation of the lifetime of VLSI systems and has lead to interesting conclusions regarding the mitigation of the impact of EM and TDDB. Some of them have also been presented in a paper that has been submitted to the 2010 PATMOS Workshop and has been accepted for oral presentation in the Workshop's Proceedings [36]. In the next section, we elaborate on the possible enhancements and extensions that should be considered regarding the proposed analysis framework presented in this work, in order to either improve the efficiency of the algorithms applied to the investigation of the impact of the EM and TDDB on a design. Then, closing this chapter, we also attempt to take a step towards the possible design- and run-time solutions that may mitigate the impact of the studied phenomena.

6.2.2 *Future extensions of the analysis framework*

The accuracy and usefulness of our interconnect reliability estimation framework could increase in several ways. In perspective, this section is dedicated to the presentation of possible future improvements, which are summarized in the following bullets:

- The development and incorporation of more advanced interconnect wear-out mechanism models would improve the accuracy of the performed predictions. Three highly recommended model-related improvements are the following:
    - The replacement of the currently used DC-mode EM model with a pulsed-DC-mode or even better, with an AC-mode EM model.
    - The incorporation of a model for the estimation of EM-induced problems in vias.
    - The expansion of the library of TDDB-induced leakage current measurements to cover a longer period of time, in order to reduce the error introduced during their extrapolation to operating conditions and time.

- The development and implementation of advanced path-selection algorithms that would target at the most vulnerable interconnects, which are both tightly timed and likely to suffer from EM or TDDB the most. Currently, a trivial selection of a number of paths with the least timing slack is performed, as these theoretically have a higher probability of introducing a timing violation. However, there is always the possibility that initially less timing-critical paths contain interconnects that particularly favor EM and/or TDDB, which renders them candidates for suffering from a significant delay degradation, because of the progressive wear-out impact. Consequently, non-timing critical, relaxed timing paths may become timing-critical after the annotation of the delay degradation on the affected interconnects, due to EM and TDDB. Hence, the classification of paths in terms of timing criticality may change, considering the delay impact that the studied wear-out mechanisms may have on the design's interconnects within the desired system's lifetime [3]. Some ideas on potential criteria to guide path selection algorithms, are the following:

    - High temperature has been shown to enhance both examined wear-out mechanisms. As a result, high temperature regions of the design, also known as "hotspots", should be considered.
    - Regarding EM, regions of any design which naturally feature particularly long wires, are good candidates for selection and analysis by our reliability framework. An example of such a region is a system's bus controller, which contains nets that drive several standard-cell input pins and in general any high-fanout interconnect.
    - Regarding TDDB, regions with high congestion such as bus controllers or timing-critical parts of a processor's pipeline like the ALU, should be considered for investigation.

6.2.3 Proposed solutions to mitigate the reliability problems

The prediction of potential reliability problems and the estimation of the system's lifetime due to any of them are certainly important. But there are cases, where the predicted lifetime is unacceptable and needs to be extended, which calls for novel solutions that could mitigate the foreseen problems. Concluding this work, we are going to propose some potential solutions, which can be divided in two categories, design-time and run-time.

- Design-time solutions:
    - The insertion of buffers, in order to achieve the segmentation of long wires, will reduce the length of those wires below the Blech length, thus avoiding EM. However, there is a certain penalty in area, power and performance due to the buffer insertion in signal nets and consequently, the various constraints of the design should be taken into consideration.
    - The down-sizing of the driving cells of EM-affected wires, if their timing constraints allow it, will decrease the current density of the driven interconnects and help reduce the impact of EM on the respective wires.
    - The EM- and TDDB-aware metal layer assignment will mitigate the problem by altering the routing of susceptible wires. This is achieved in the case of EM through the reduction of the current density by moving susceptible wires to higher metal layers and in the case of TDDB by moving adjacent wires to different layers.

- Run-time solutions:
    - As temperature is a key factor in the evolution of EM and TDDB, thermal monitoring could be performed on the design's units and the hottest units could be relieved. This could be achieved through either task scheduling, or dynamic voltage scaling. Regarding task scheduling, the target is the load balancing between the processes assigned to two or more processor cores, in order to maintain their temperature at reasonable levels. Another example is the reduction of the supply voltage of the respective component sacrificing a part of its performance, in conjunction with an appropriate decrease in the operating frequency.
    - The TDDB-induced leakage current leads to a potentially noticeable increase in the dissipated power of the affected regions of the design. So, if the power dissipation is monitored and an increase is detected, run-time mechanisms could be triggered to mitigate the phenomenon. One potential solution could be the voltage scaling of the respective component, which would reduce the intensity of the electric field and thus decelerate the dielectric breakdown, accompanied by an appropriate reduction in the operating frequency.

Finally, we would like to point out the need for the development of cost-effective techniques that alleviate these progressive wear-out phenomena, as their impact will probably keep gaining in significance with the advent of the sub-45nm deep-submicron technology nodes.

7

Appendix

The various scripts and configuration or parameter files that comprise the presented reliability analysis framework are cited below, so that the reader can delve deeper into its implementation details and enlighten any potentially dark spots that the description provided in Chapter 4 left.

library_para_new:

This file contains the EM parameters derived from EM experimental results.

```
############# stress condition #############
current_density 40 mA/um2
temperature 300 degree
slope 0.05 ohm/s
activation_energy 0.9 ev
n_t50 1.2
t50_current_density 2 mA/um2
t50_stress 50 hours
############# operation condition #############
voltage 0.9 v
temperature 100 degree
```

step1_sp_generation.pl:

This script generates the Spice netlists for the simulations, whose results are used to build the TDDB LUT library.

```
#!/usr/local/bin/perl

# script to generate the .sp file for TDDB simulation
# updated, to generate the RC tree netlist

############### PARAMETERS 45nm  ############################
$stages=10;
$Thickness=0.14e-6;           # m
$epsilo=8.85e-12;             # SI
$dielectric_constant=2.3;
$cap_load=3e-15;
$R_per_length=4;              # ohm/um
$C_per_length=0.13e-15;              # F/um


#################### reading lib_input ####################
open(mylib, "./lib_input");

# Get the interconnect min,max length and the steps
$stream = <mylib>;
chomp($stream);
@data = split(/:/,$stream);
@length_index = split(/ /,$data[1]);

# Get the offset min length and the steps
$stream = <mylib>;
chomp($stream);
@data = split(/:/,$stream);
@offset_index = split(/ /,$data[1]);

# Get the leakage current min,max value and the steps
$stream = <mylib>;
chomp($stream);
@data = split(/:/,$stream);
@leakage_index = split(/ /,$data[1]);

# Get the distance min value and the steps
$stream = <mylib>;
chomp($stream);
@data = split(/:/,$stream);
@distance_index = split(/ /,$data[1]);

close (mylib);


########### Write simulations file ###########
open(spice_sim, ">./spice_sim");
```

```
print(spice_sim "#!/bin/bash\n");

print (spice_sim "cd source\n");

# Loop so that every valid combination of parameters gets calculated
# i: wire length , n: neighbour length , s: offset , d: distance , l: leakage current
for ($i=0; $i<scalar(@length_index); $i++) {

    for ($n=0; ($n<scalar(@length_index) && $length_index[$n]<=$length_index[$i]) ;
$n++) {

        for ($s=0 ; ($s<scalar(@offset_index) && $length_index[$i] >=
$length_index[$n]+$offset_index[$s]); $s++) {

            for ($d=0; $d < scalar(@distance_index); $d++) {

                for ($l=0; $l < scalar(@leakage_index); $l++) {

                        # Wire-ground capacitance and the value of each stage of it
                        $cap_ground = $C_per_length*$length_index[$i];
                        $cap_ground_seg = $cap_ground/($stages-1);

                        # Adjacent wire capacitance

        $cap_neighbour=$Thickness*$length_index[$n]*$epsilo*$dielectric_constant/$dista
nce_index[$d];


                        # Write the spice netlist file

        $temp="l".$length_index[$i]."n".$length_index[$n]."s".$offset_index[$s]."i".$le
akage_index[$l]."d".$distance_index[$d];
                        open(spice_file, ">./source/$temp.sp");
                        print (spice_sim "hspice ./$temp.sp\n");

                        # Calculate R1 R2 R3,which are the various stage resistances
along the wire (pre-offset,adjacent,rest)
                        $R1=$offset_index[$s]*$R_per_length;
                        $R2=$length_index[$n]*$R_per_length;
                        $R3=$R_per_length*$length_index[$i]-$R1-$R2;

                        # Calculate the number of stages for each part of the wire
                        $R1_stages=int($R1/($R1+$R2+$R3)*$stages);
                        $R3_stages=int($R3/($R1+$R2+$R3)*$stages);
                        $R2_stages=$stages-$R1_stages-$R3_stages;

                        $stage_counter=1;

                        print(spice_file " .OPTIONS LIST NODE POST\n.OP\nVIN 1 0 PULSE
(0 1 0.1N 0.2N 0.2N 2.1N 5N)\n");

                        # R1 stages
                        if ($R1_stages>0) {
                                $R_seg=$R1/$R1_stages;
                                $cap_seg=$cap_ground_seg;
                                for ($x=0;$x<$R1_stages; $x++) {
                                    print(spice_file "R$stage_counter $stage_counter
",$stage_counter+1," $R_seg\n");
                                    print(spice_file "C$stage_counter
",$stage_counter+1," 0 $cap_seg\n");
                                    $stage_counter++;
                                }
                        }

                        # R2 stages
                        if ($R2_stages>0) {
                                $R_seg=$R2/$R2_stages;
                                $leakage=$leakage_index[$l]*1e-6;
                                $leakage_seg=$leakage/$R2_stages;
                                $cap_seg=$cap_ground_seg+$cap_neighbour/$R2_stages;
                                for ($x=0;$x<$R2_stages; $x++) {
                                    print(spice_file "R$stage_counter $stage_counter
",$stage_counter+1," $R_seg\n");
                                    print(spice_file "Ileakage$stage_counter
",$stage_counter+1," 0  $leakage_seg\n");
                                    if ($stage_counter<$stages) {
                                       print(spice_file "C$stage_counter
",$stage_counter+1," 0 $cap_seg\n");
                                    }
                                    $stage_counter++;
                                }
                        }

                        # R3 stages
                        if ($R3_stages>0) {
                                $R_seg=$R3/$R3_stages;
                                $cap_seg=$cap_ground_seg;
                                for ($x=0;$x<$R3_stages; $x++) {
                                    print(spice_file "R$stage_counter $stage_counter
",$stage_counter+1," $R_seg\n");
                                    if ($stage_counter<$stages) {
                                       print(spice_file "C$stage_counter
",$stage_counter+1," 0 $cap_seg\n");
                                    }
                                    $stage_counter++;
                                }
                        }

                        #Add the load capacitance to the spice netlist
                        print(spice_file "C_load  $stage_counter 0 $cap_load\n");

                        print(spice_file ".option post\n.TRAN 0.01N 15N\n.PRINT TRAN
V(1) V($stage_counter)\n.MEASURE TRAN tdelay_rise TRIG V(1) VAL=0.9 TD=0.01n RISE=1\n+
TARG V($stage_counter) VAL=0.9          RISE=1\n.MEASURE TRAN tdelay_fall TRIG V(1)
VAL=0.1 TD=0.01n FALL=1\n+                        TARG V($stage_counter) VAL=0.1
FALL=1\n.END");

                        close (spice_file);
                }
            }
        }
    }
}


print (spice_sim "cd ..\n");
close (spice_sim);
```

step3_reading_mt0.pl:

This script builds the TDDB LUT library.

```
#!/usr/local/bin/perl

# script to read the .mt0 file to generate the Look UP Table

#################### reading lib_input ####################
open(mylib, "./lib_input");

# Get the interconnect min,max length and the steps
$stream = <mylib>;
chomp($stream);
@data = split(/:/,$stream);
@length_index = split(/ /,$data[1]);

# Get the offset min length and the steps
$stream = <mylib>;
chomp($stream);
@data = split(/:/,$stream);
@offset_index = split(/ /,$data[1]);

# Get the leakage current min,max value and the steps
$stream = <mylib>;
chomp($stream);
@data = split(/:/,$stream);
@leakage_index = split(/ /,$data[1]);

# Get the distance min value and the steps
$stream = <mylib>;
```

```
chomp($stream);
@data = split(/:/,$stream);
@distance_index = split(/ /,$data[1]);

close (mylib);

# Write spice simulations results to create the lookup table

open (TDDB_LUT, ">./TDDB_LUT.lib");

print (TDDB_LUT "Length(um) Length(neighbour um) start_point(um) Leakage(uA)
distance(um): delay_change_ratio\n");

# i: wire length , n: neighbour length , s: offset , d: distance , l: leakage current
for ($i=0; $i<scalar(@length_index); $i++) {

    for ($n=0; ($n<scalar(@length_index) && $length_index[$n]<=$length_index[$i]) ;
$n++) {

        for ($s=0 ; ($s<scalar(@offset_index) && $length_index[$i] >=
$length_index[$n]+$offset_index[$s]); $s++) {

            for ($d=0; $d < scalar(@distance_index); $d++) {

                for ($l=0; $l < scalar(@leakage_index); $l++) {


$temp="l".$length_index[$i]."n".$length_index[$n]."s".$offset_index[$s]."i".$leakage_i
ndex[$l]."d".$distance_index[$d];

                    # Calculate the delay time change ratio
                    open (mt0_file, "./source/$temp.mt0");
                    $stream = <mt0_file>;
                    $stream = <mt0_file>;
                    $stream = <mt0_file>;
                    $stream = <mt0_file>;
                    # Trim leading and trailing white spaces
                    for ($stream) {
                        s/^\s+//;
                        s/\s+$//;
                    }
                    @data = split(/ /,$stream);
                    # If there is no leakage,set delay time as reference
                    if ($leakage_index[$l]==0) {
                        $reference_delay=$data[0];
                        $delay_change=0;
                    } else {
                        $delay_change=($data[0]-$reference_delay)/$reference_delay;
                        if ($delay_change==-1) {
                                $delay_change=-999999999;
                        }
                    }
                    close (mt0_file);

                    print (TDDB_LUT "$length_index[$i] $length_index[$n]
$offset_index[$s] $leakage_index[$l] $distance_index[$d] : $delay_change\n");
                }
            }
        }
    }
}

close (TDDB_LUT);
```

lib_input:

This file contains the parameters for the wire patterns to be included in the TDDB LUT library.

```
Length(um):10 20 30 40 50 75 100 150 200 300 400 500 600 700 800 900 1000
Offset(um):0 10 20 30 40 50 75 100 150 200 300 400 500 600 700 800 900
Leakage(uA):0 0.25 0.5 0.75 1 2.5 5 10 15 20 25 30 40 50 60 70 80 90 100 125 150
Distance(um):0.06 0.25 0.5
```

format_path.tcl:

This script converts the critical paths from the format of the ETS timing report to that of the extraction script.

```
if { $argc<1 } {
       puts "Usage: tclsh format_path.tcl initial_timing"
       exit
}

set sfile [open "./[lindex $argv 0]" r]
set dfile [open "./critical_paths.txt" w]

while { true } {

    # Search for the pin at the end of the critical path
    gets $sfile line
    while {![eof $sfile] && [string range [string trim $line] 0 4]!="ENDPT"} {
        gets $sfile line
    }

    if {[eof $sfile]}  break

    puts $dfile "Net\tInput pin(net end)\tOutput pin(net start)"

    set line [split $line "{}"]
    set end ""
    if {[lindex $line 1]!=""} {
        set end "[lindex $line 1]/"
    }
    set end "$end[lindex $line 3]"

    # Search for the start of the critical path
    while {![eof $sfile] && [string trim $line]!="DATA_PATH"} {
        gets $sfile line
    }

    gets $sfile line

    # Initialize the process by reading the first output pin and the first net
    gets $sfile line
    set line [split $line "{}"]
    set from "[lindex $line 1]/[lindex $line 7]"
    gets $sfile line
    set line [split $line "{}"]
    set net [lindex $line 11]
    gets $sfile line

    # Continue processing "net to from" lines until the end of the path is reached
    while {![eof $sfile] && [string trim $line]!="END_DATA_PATH"} {
        while {[string range [string trim $line] 0 3]!="INST"} {
            gets $sfile line
        }
        set line [split $line "{}"]
        set to "[lindex $line 1]/[lindex $line 3]"
        puts $dfile "$net $to $from"
        set from "[lindex $line 1]/[lindex $line 7]"
        while {[string range [string trim $line] 0 2]!="NET"} {
            gets $sfile line
        }
        set line [split $line "{}"]
        set net [lindex $line 11]
        gets $sfile line
    }

    # Add the "net to from" line with the end of the critical path
    puts $dfile "$net $end $from"

}

close $sfile
close $dfile

puts 1
```

floorplan_converter.tcl:

This script converts the floorplan coordinates from the format of the SoC Encounter to that of HotSpot.

```
if { $argc<1 } {
        puts "Usage: tclsh floorplan_converter.tcl design_name"
        exit
}

set design_name [lindex $argv 0]

set fpfile [open "./$design_name.fp" r]
set flpfile [open "~/designs/rel_script/HotSpot-4.2/$design_name.flp" w]

puts $flpfile "# Line Format: <unit-name>\t<width>\t<height>\t<left-x>\t<bottom-y>"
puts $flpfile "# all dimensions are in meters"
puts $flpfile "# comment lines begin with a '#'"
puts $flpfile "# comments and empty lines are ignored"

gets $fpfile line

while {1} {
    while {![eof $fpfile] && [string range $line 0 5]!="Guide:"} {
        gets $fpfile line
    }
    if {[eof $fpfile]} { break }
    set data [split $line " "]
    set unit [lindex $data 1]
    set llx [format "%.8f" [expr {[lindex $data 2]*1e-6}]]
    set lly [format "%.8f" [expr {[lindex $data 3]*1e-6}]]
    set urx [format "%.8f" [expr {[lindex $data 4]*1e-6}]]
    set ury [format "%.8f" [expr {[lindex $data 5]*1e-6}]]
    set width [format "%.8f" [expr {$urx-$llx}]]
    set height [format "%.8f" [expr {$ury-$lly}]]
    puts $flpfile "$unit\t$width\t$height\t$llx\t$lly"
    gets $fpfile line
}

close $fpfile
close $flpfile

puts 1
```

power_converter.tcl:

This script converts the power consumption data from the format of the PrimeTimePX to that of HotSpot.

```
# Power converter from PrimeTimePX to HotSpot

if { $argc<1 } {
        puts "Usage: tclsh power_converter.tcl design_name"
        exit
}

set design_name [lindex $argv 0]

set unitsf [open "~/designs/rel_script/HotSpot-4.2/$design_name.flp" r]
gets $unitsf line
set design_units [list]
while {![eof $unitsf]} {
    if {$line!="" && [string range $line 0 0]!="#"} {
        lappend design_units [lindex [split $line "\t"] 0]
    }
    gets $unitsf line
}
close $unitsf

set unit_power [list]
```

```
for {set i 0} {$i < [llength $design_units]} {incr i} {
    set unit [lindex $design_units $i]
    set pfile [open "./$design_name.power.report" r]
    gets $pfile line
    set line [regsub -all {[ \t]+} [string trim $line] { }]
    set temp [split $line " "]
    set hier [split $unit "/"]
    for {set j 0} {$j < [llength $hier]} {incr j} {
        while {![eof $pfile] && [lindex $temp 0]!=[lindex $hier $j]} {
            gets $pfile line
            set line [regsub -all {[ ]+} [string trim $line] { }]
            set temp [split $line " "]
        }
        if {[eof $pfile]} {
            puts "Unit $unit not found in floorplan."
            exit
        }
    }
    set pwr [lindex $temp 5]
    lappend unit_power [format "%.11f" $pwr]
    puts "Unit: $unit Power(W): $pwr"
    close $pfile
}

set ptrace [open "~/designs/rel_script/HotSpot-4.2/$design_name.ptrace" w]
set units [expr {[llength $design_units]-1}]
for {set i 0} {$i <= $units} {incr i} {
        puts -nonewline $ptrace [lindex $design_units $i]
        if { $i==$units } {
                puts $ptrace ""
        } else {
                puts -nonewline $ptrace "\t"
        }
}
for {set i 0} {$i <= $units} {incr i} {
        puts -nonewline $ptrace [lindex $unit_power $i]
        if { $i==$units } {
                puts $ptrace ""
        } else {
                puts -nonewline $ptrace "\t"
        }
}
close $ptrace
```

temp_flow.sh:

This script automates the power and consequently the temperature estimation for the tool flow.

```
#!/bin/bash

source ./flow_conf.sh

if [ ! -s "$design_name.fp" -o ! -s "~/designs/rel_script/HotSpot-
4.2/$design_name.flp" ]; then
        echo "
        restoreDesign $enc_dat $top_module
        saveFPlan $design_name.fp
        exit
        " > commands.tmp
        echo "source commands.tmp" | encounter -nowin
        tclsh floorplan_converter.tcl $design_name
        rm commands.tmp
fi

if [ ! -s "$design_name.power.report" ]; then
        tempvar=`awk '{if ($1=="Guide:") printf("%s ",$2)}' $design_name.fp`
        echo "
        restoreDesign $enc_dat $top_module
        extractRC
        probePower $tempvar
        updatePower -vcd $vcd -vcdTop $vcdtop -noRailAnalysis -report
$design_name.power.report VDD
        exit
        " > commands.tmp
        echo "source commands.tmp" | encounter -nowin
        rm commands.tmp
fi

tclsh power_analysis.tcl $design_name

cd ~/designs/rel_script/HotSpot-4.2

if [ ! -s "$design_name.flp" ]; then
echo "$design_name.flp is missing"
exit
fi

if [ ! -s "$design_name.ptrace" ]; then
echo "$design_name.ptrace is missing"
exit
fi

./hotspot -c hotspot.config -f $design_name.flp -p $design_name.ptrace -o
unit_temps.ttrace -steady_file $design_name.steady
cp $design_name.steady $design_name.init
./hotspot -c hotspot.config -init $design_name.init -f $design_name.flp -p
$design_name.ptrace -o unit_temps.ttrace
```

flow1.sh:

This script automates the execution of the EM and TDDB tool flow except for the SDF file annotation with TDDB and the final static timing analysis to estimate the combined impact on delay.

```
#!/bin/bash

source ./flow_conf.sh

rm $wire_report
rm $deltaR_report
rm $initial_timing
rm $em_timing
rm $tddb_timing
rm $critical_path
rm $em_spef
rm $em_sdf
rm $tddb_sdf

if [ ! -s "$enc_dat" ]; then
echo "$enc_dat is missing"
exit
fi

if [ ! -s "$initial_spef" ]; then
echo "
restoreDesign $enc_dat $top_module
extractRC
rcOut -spef $initial_spef
exit
" > commands.tmp

echo "source commands.tmp" | encounter -nowin
fi

if [ ! -s "$verilog" ]; then
echo "
restoreDesign $enc_dat $top_module
saveNetlist $verilog
exit
" > commands.tmp

echo "source commands.tmp" | encounter -nowin
```

```
fi

if [ ! -s "$initial_sdf" ]; then
echo "
read_lib $libs
read_verilog $verilog
set_top_module $top_module
read_sdc $sdc
set_analysis_mode -checkType setup
set_op_cond NCCOM -library $library
read_spef $initial_spef
create_clock "$clk_name" -name clk -period $clk_period
write_sdf -precision 4 $initial_sdf
exit
" > commands.tmp

echo "source commands.tmp" | ets -nowin

fi

if [ ! -s "$initial_sdf" ]; then
echo "Error producing $initial_sdf"
exit
fi

echo "
read_lib $libs
read_verilog $verilog
set_top_module $top_module
read_sdc $sdc
set_analysis_mode -checkType setup
set_op_cond NCCOM -library $library
read_sdf $initial_sdf
create_clock "$clk_name" -name clk -period $clk_period
report_timing -machine_readable -max_points $paths > $initial_timing
exit
" > commands.tmp

echo "source commands.tmp" | ets -nowin

if [ ! -s "$initial_timing" ]; then
echo "Error producing $initial_timing"
exit
fi

tclsh format_path.tcl $initial_timing

if [ ! -s "$critical_path" ]; then
echo "Error producing $critical_path"
exit
fi

echo "
restoreDesign $enc_dat $top_module
extractRC
delayCal
source extraction_temp_multipath_christos.tcl
exit
" > commands.tmp

echo "source commands.tmp" | encounter -nowin

if [ ! -s "$wire_report" ]; then
echo "Error producing $wire_report"
exit
fi

if [ ! -s "$deltaR_report" ]; then
echo "Error producing $deltaR_report"
exit
fi

perl spef_update_christos.pl $initial_spef $em_spef

if [ ! -s "$em_spef" ]; then
echo "Error producing $em_spef"
exit
```

```
fi

echo "
read_lib $libs
read_verilog $verilog
set_top_module $top_module
read_sdc $sdc
set_analysis_mode -checkType setup
set_op_cond NCCOM -library $library
read_spef $em_spef
create_clock "$clk_name" -name clk -period $clk_period
write_sdf -precision 4 $em_sdf
exit
" > commands.tmp

echo "source commands.tmp" | ets -nowin

if [ ! -s "$em_sdf" ]; then
echo "Error producing $em_sdf"
exit
fi

echo "
read_lib $libs
read_verilog $verilog
set_top_module $top_module
read_sdc $sdc
set_analysis_mode -checkType setup
set_op_cond NCCOM -library $library
read_sdf $em_sdf
create_clock "$clk_name" -name clk -period $clk_period
report_timing -machine_readable -max_points $paths > $em_timing
exit
" > commands.tmp

echo "source commands.tmp" | ets -nowin

if [ ! -s "$em_timing" ]; then
echo "Error producing $em_timing"
exit
fi

rm commands.tmp

cp $em_sdf delays.sdf

echo "Now run TDDB analysis using delays.sdf and $wire_report, copy back delays.sdf
(rename to $tddb_sdf) and continue with flow2.sh script."
```

flow2.sh:

This script automates the SDF file annotation with TDDB and the final static timing analysis to estimate the combined impact on delay.

```
#!/bin/bash

source ./flow_conf.sh

if [ ! -s "$tddb_sdf" ]; then
echo "$tddb_sdf is missing"
exit
fi

echo "
read_lib $libs
read_verilog $verilog
set_top_module $top_module
read_sdc $sdc
set_analysis_mode -checkType setup
set_op_cond NCCOM -library $library
read_sdf $tddb_sdf
create_clock "$clk_name" -name clk -period $clk_period
```

```
report_timing -machine_readable -max_points $paths > $tddb_timing
exit
" > commands.tmp

echo "source commands.tmp" | ets -nowin

if [ ! -s "$tddb_timing" ]; then
echo "Error producing $tddb_timing"
exit
fi

rm commands.tmp
```

flow_conf.sh:

This sample file contains examples of values for the variables for the configuration of the automated tool flow.

```
design_name="fir16_sp"
wire_report="wire.report" #requires changes
deltaR_report="deltaR.report" #requires changes
initial_timing="fir16_sp_initial_timing.txt"
em_timing="fir16_sp_em_timing.txt"
tddb_timing="fir16_sp_tddb_timing.txt"
critical_path="critical_paths.txt" #requires changes
initial_spef="fir16_sp.spef"
em_spef="fir16_sp_em.spef"
initial_sdf="fir16_sp_initial.sdf"
em_sdf="fir16_sp_em.sdf"
tddb_sdf="fir16_sp_tddb.sdf"
libs="tcbn45gsbwptc.lib"
verilog="fir16_sp.v"
top_module="fir16_sp"
sdc="fir16_sp.sdc"
enc_dat="fir16_sp.enc.dat"
library="tcbn45gsbwptc"
vcd="fir16_sp.vcd"
vcdtop="topfir16/fir16_sp"
clk_name="clk"
clk_period="2.2"
paths="1"
```

rhumc.tcl:

The current script is the most important among those implementing the synthesis of Leon3-based MP-SoC, as it invokes all the other scripts of the synthesis flow and it is modified to fit to the TSMC 45nm standard-cell library used.

```
source setup_rhumc_old.tcl
source leon3mp_dc.tcl
set_scan_configuration -style multiplexed_flip_flop
source timing_memories.tcl

#ungroup -flatten -simple_names core0/ringosc0/drx
#ungroup -flatten -simple_names core0/leon3core0/grspw0_0/nrx_clkbuf_0

#ungroup core0/ringosc0/drx  -flatten -simple_names
#ungroup core0/leon3core0/dsu0/x0  -simple_names
#ungroup core0/leon3core0/grspw0_0/ram0  -flatten -simple_names
#ungroup core0/leon3core0/grspw0_0/grspwc0  -flatten -simple_names
#ungroup core0/leon3core0/grspw0_0/nrx_clkbuf_0  -flatten -simple_names
#ungroup core0/leon3core0/grspw0_0/rx_clkbuf_0  -flatten -simple_names
#ungroup core0/leon3core0/grspw0_0/rx_ram0  -flatten -simple_names
#ungroup core0/leon3core0/grspw0_0/rx_ram1  -flatten -simple_names
#ungroup core0/leon3core0/grspw0_0/tx_ram0  -flatten -simple_names

#ungroup core0/leon3core0/grspw0_1/ram0  -flatten -simple_names
#ungroup core0/leon3core0/grspw0_1/grspwc0  -flatten -simple_names
```

```
#ungroup core0/leon3core0/grspw0_1/nrx_clkbuf_0  -flatten -simple_names
#ungroup core0/leon3core0/grspw0_1/rx_clkbuf_0  -flatten -simple_names
#ungroup core0/leon3core0/grspw0_1/rx_ram0  -flatten -simple_names
#ungroup core0/leon3core0/grspw0_1/rx_ram1  -flatten -simple_names
#ungroup core0/leon3core0/grspw0_1/tx_ram0  -flatten -simple_names

#ungroup core0/leon3core0/leon3ft0_0/tbmem0/ram0_0  -flatten -simple_names
#ungroup core0/leon3core0/leon3ft0_0/tbmem0/ram0_1  -flatten -simple_names

#current_instance pads0
#ungroup [find cell "*"] -flatten -simple_names
#current_instance ..
#set_dont_touch pads0

#current_instance core0
#ungroup find(cell, {"clk*"} ) -flatten -simple_names
#current_instance leon3core0
#group [find cell {apb* uart* timer* irq* ahb* rst0 dcom* grg* sr* dsu0 ahbjtag0 }]  -
design_name amod -cell_name amod0
#current_instance leon3ft0_0/p0
#ungroup -all -flatten -simple_names
#current_instance ../rf0
#ungroup -all -flatten -simple_names
#current_instance ../cmem0
#ungroup -all -flatten -simple_names
#current_instance ../fpu0
#ungroup -all -flatten -simple_names
#current_instance ../../ahbuart0
#ungroup -all -flatten -simple_names

#current_instance ../ftmctrl0
#ungroup -all -flatten -simple_names
#current_instance ..
#ungroup ahbctrl0 -flatten -simple_names
#ungroup apbctrl0 -flatten -simple_names

#current_instance ../../..

set compile_auto_ungroup_override_wlm "true"
set compile_auto_ungroup_count_leaf_cells "true"
set compile_auto_ungroup_delay_num_cells 100
set compile_ultra_ungroup_small_hierarchies "false"
set compile_auto_ungroup_area_num_cells 100

set_max_area 0
set_max_transition 1.0 leon3mp

source scan.tcl

#compile_ultra -scan -no_boundary_optimization
#compile_ultra -scan -retime
set compile_seqmap_propagate_constants false
link
compile

#write -f ddc -hier leon3mp -output synopsys/leon3mp.ddc

report_timing
report_timing > synopsys/timing1.log
write_sdc synopsys/leon3mp.sdc
report_area
report_area -hierarchy > synopsys/area1.log
report_power
report_power > synopsys/pow1.log
report_power -hier > synopsys/pow1h.log

change_names -rules verilog -hierarchy
###write -f verilog -hier leon3mp -output leon3mp.v
write -f verilog -hier leon3mp -output leon3mp_newgrlib_new.v
write_sdf leon3mp_initial.sdf

#write -f vhdl -hier leon3mp -output leon3mp.vhd
#source timing3.tcl
source scan2.tcl
quit
```

The invoked scripts, namely both the *leon3mp_dc.tcl* and the *compile.dc*, which perform the elaboration of the RTL code in Design Compiler, along with the script of the timing constraints, *timing_memories.tcl*, are given below.

leon3mp_dc.tcl:

```
sh mkdir synopsys
set objects synopsys
set hdlin_ff_always_sync_set_reset true
set hdlin_ff_always_async_set_reset false
set hdlin_infer_complex_set_reset true
set hdlin_translate_off_skip_text true
set suppress_errors VHDL-2285
set hdlin_use_carry_in true
source  compile.dc
analyze -f VHDL -library work config.vhd
analyze -f VHDL -library work ahbrom.vhd
analyze -f VHDL -library work leon3core.vhd
analyze -f VHDL -library work core.vhd
analyze -f VHDL -library work pads.vhd
analyze -f VHDL -library work leon3mp.vhd
elaborate leon3mp
```

compile.dc:

```
sh mkdir synopsys
sh mkdir synopsys/grlib
define_design_lib grlib -path synopsys/grlib
analyze -f VHDL -library grlib ../../lib/grlib/stdlib/version.vhd
analyze -f VHDL -library grlib ../../lib/grlib/stdlib/stdlib.vhd
analyze -f VHDL -library grlib ../../lib/grlib/sparc/sparc.vhd
analyze -f VHDL -library grlib ../../lib/grlib/sparc/sparc_disas.vhd
analyze -f VHDL -library grlib ../../lib/grlib/sparc/cpu_disas.vhd
analyze -f VHDL -library grlib ../../lib/grlib/modgen/multlib.vhd
analyze -f VHDL -library grlib ../../lib/grlib/modgen/leaves.vhd
analyze -f VHDL -library grlib ../../lib/grlib/amba/amba.vhd
analyze -f VHDL -library grlib ../../lib/grlib/amba/devices.vhd
analyze -f VHDL -library grlib ../../lib/grlib/amba/defmst.vhd
analyze -f VHDL -library grlib ../../lib/grlib/amba/apbctrl.vhd
analyze -f VHDL -library grlib ../../lib/grlib/amba/ahbctrl.vhd
analyze -f VHDL -library grlib ../../lib/grlib/amba/dma2ahb_pkg.vhd
analyze -f VHDL -library grlib ../../lib/grlib/amba/dma2ahb.vhd
sh mkdir synopsys/virage
define_design_lib virage -path synopsys/virage
sh mkdir synopsys/atc18
define_design_lib atc18 -path synopsys/atc18
sh mkdir synopsys/umc18
define_design_lib umc18 -path synopsys/umc18
sh mkdir synopsys/synplify
define_design_lib synplify -path synopsys/synplify
sh mkdir synopsys/techmap
define_design_lib techmap -path synopsys/techmap
analyze -f VHDL -library techmap ../../lib/techmap/gencomp/gencomp.vhd
analyze -f VHDL -library techmap ../../lib/techmap/gencomp/netcomp.vhd
analyze -f VHDL -library techmap ../../lib/techmap/inferred/memory_inferred.vhd
###analyze -f VHDL -library techmap
../../lib/techmap/inferred/memory_inferred_newgrlib.vhd
analyze -f VHDL -library techmap ../../lib/techmap/inferred/ddr_inferred.vhd
analyze -f VHDL -library techmap ../../lib/techmap/inferred/mul_inferred.vhd
analyze -f VHDL -library techmap ../../lib/techmap/inferred/ddr_phy_inferred.vhd
analyze -f VHDL -library techmap ../../lib/techmap/dw02/mul_dw_gen.vhd
analyze -f VHDL -library techmap ../../lib/techmap/virage/memory_virage.vhd
analyze -f VHDL -library techmap ../../lib/techmap/atc18/pads_atc18.vhd
analyze -f VHDL -library techmap ../../lib/techmap/umc18/memory_umc18.vhd
analyze -f VHDL -library techmap ../../lib/techmap/umc18/pads_umc18.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/allclkgen.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/allddr.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/allmem.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/allpads.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/alltap.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/clkgen.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/clkmux.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/clkand.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/ddr_ireg.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/ddr_oreg.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/ddrphy.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/syncram.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/syncram64.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/syncram_2p.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/syncram_dp.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/syncfifo.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/regfile_3p.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/tap.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/techbuf.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/nandtree.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/clkpad.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/clkpad_ds.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/inpad.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/inpad_ds.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/iodpad.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/iopad.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/iopad_ds.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/lvds_combo.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/odpad.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/outpad.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/outpad_ds.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/toutpad.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/skew_outpad.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/grspwc_net.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/grspwc2_net.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/grlfpw_net.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/grfpw_net.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/mul_61x61.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/cpu_disas_net.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/grusbhc_net.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/ringosc.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/ssrctrl_net.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/system_monitor.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/grgates.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/inpad_ddr.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/outpad_ddr.vhd
analyze -f VHDL -library techmap ../../lib/techmap/maps/iopad_ddr.vhd
sh mkdir synopsys/spw
define_design_lib spw -path synopsys/spw
analyze -f VHDL -library spw ../../lib/spw/comp/spwcomp.vhd
analyze -f VHDL -library spw ../../lib/spw/wrapper/grspw_gen.vhd
sh mkdir synopsys/eth
define_design_lib eth -path synopsys/eth
analyze -f VHDL -library eth ../../lib/eth/comp/ethcomp.vhd
analyze -f VHDL -library eth ../../lib/eth/core/greth_pkg.vhd
analyze -f VHDL -library eth ../../lib/eth/core/eth_rstgen.vhd
analyze -f VHDL -library eth ../../lib/eth/core/eth_ahb_mst.vhd
analyze -f VHDL -library eth ../../lib/eth/core/greth_tx.vhd
analyze -f VHDL -library eth ../../lib/eth/core/greth_rx.vhd
analyze -f VHDL -library eth ../../lib/eth/core/grethc.vhd
analyze -f VHDL -library eth ../../lib/eth/wrapper/greth_gen.vhd
analyze -f VHDL -library eth ../../lib/eth/wrapper/greth_gbit_gen.vhd
sh mkdir synopsys/opencores
define_design_lib opencores -path synopsys/opencores
analyze -f VHDL -library opencores ../../lib/opencores/occomp/occomp.vhd
analyze -f VHDL -library opencores ../../lib/opencores/can/cancomp.vhd
analyze -f VHDL -library opencores ../../lib/opencores/can/can_top.vhd
analyze -f VHDL -library opencores ../../lib/opencores/i2c/i2c_master_bit_ctrl.vhd
analyze -f VHDL -library opencores ../../lib/opencores/i2c/i2c_master_byte_ctrl.vhd
analyze -f VHDL -library opencores ../../lib/opencores/i2c/i2coc.vhd
analyze -f VERILOG -library opencores ../../lib/opencores/spi/simple_spi_top.v
analyze -f VHDL -library opencores ../../lib/opencores/ata/ud_cnt.vhd
analyze -f VHDL -library opencores ../../lib/opencores/ata/ro_cnt.vhd
analyze -f VHDL -library opencores ../../lib/opencores/ata/atahost_dma_fifo.vhd
analyze -f VHDL -library opencores ../../lib/opencores/ata/atahost_dma_actrl.vhd
analyze -f VHDL -library opencores ../../lib/opencores/ata/atahost_dma_tctrl.vhd
analyze -f VHDL -library opencores ../../lib/opencores/ata/atahost_pio_tctrl.vhd
analyze -f VHDL -library opencores ../../lib/opencores/ata/atahost_pio_actrl.vhd
analyze -f VHDL -library opencores ../../lib/opencores/ata/atahost_controller.vhd
analyze -f VHDL -library opencores ../../lib/opencores/ata/atahost_pio_controller.vhd
analyze -f VHDL -library opencores ../../lib/opencores/ata/ocidec2_controller.vhd
analyze -f VERILOG -library opencores ../../lib/opencores/ac97/ac97_top.v
sh mkdir synopsys/gaisler
define_design_lib gaisler -path synopsys/gaisler
analyze -f VHDL -library gaisler ../../lib/gaisler/arith/arith.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/arith/mul32.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/arith/div32.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/memctrl/memctrl.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/memctrl/sdctrl.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/memctrl/sdmctrl.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/memctrl/srctrl.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/memctrl/spimctrl.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/leon3.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/mmuconfig.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/mmuiface.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/libmmu.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/libiu.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/libcache.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/libproc3.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/cachemem.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/mmu_icache.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/mmu_dcache.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/mmu_acache.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/mmutlbcam.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/mmulrue.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/mmulru.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/mmutlb.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/mmutw.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/mmu.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/mmu_cache.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/acache.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/dcache.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/icache.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/cache.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/cpu_disasx.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/iu3.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/grfpwx.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/mfpwx.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/grlfpwx.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/tbufmem.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/dsu3x.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/dsu3.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/proc3.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/leon3s.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/leon3cg.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/irqmp.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/grfpwxsh.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/grfpushwx.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/leon3/leon3sh.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/can/can.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/can/can_mod.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/can/can_oc.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/can/can_mc.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/can/canmux.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/can/can_rd.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/misc.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/rstgen.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/gptimer.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/ahbram.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/ahbdpram.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/ahbtrace.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/ahbmst.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/grgpio.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/ahbstat.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/logan.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/apbps2.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/charrom_package.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/charrom.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/apbvga.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/ahbdma.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/svgactrl.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/i2cmst.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/i2cmst_gen.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/spictrl.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/i2cslv.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/wild.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/wild2ahb.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/grsysmon.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/gracectrl.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/misc/grgpreg.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/net/net.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/pci/pci.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/pci/pcilib.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/pci/pciahbmst.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/pci/pcitrace.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/pci/pci_target.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/pci/pci_mt.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/pci/dmactrl.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/pci/pci_mtf.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/pci/pcipads.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/pci/pcidma.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/uart/uart.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/uart/libdcom.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/uart/apbuart.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/uart/dcom.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/uart/dcom_uart.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/uart/ahbuart.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/jtag/jtag.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/jtag/libjtagcom.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/jtag/jtagcom.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/jtag/ahbjtag.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/jtag/ahbjtag_bsd.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/greth/ethernet_mac.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/greth/greth.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/greth/greth_gbit.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/greth/grethm.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/spacewire/spacewire.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/spacewire/grspw.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/spacewire/grspw2.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/spacewire/grspwm.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/usb/grusb.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ddr/ddrrec.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ddr/hs.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ddr/ahb_slv.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ddr/ddrctrl.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ddr/ddr_phy.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ddr/ddrsp.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ddr/ddrsp16a.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ddr/ddrsp32a.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ddr/ddrsp64a.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ddr/ddrspa.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ddr/ddr2sp64a.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ddr/ddr2sp32a.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ddr/ddr2sp16a.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ddr/ddr2spa.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ata/ata.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ata/ata_inf.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ata/atahost_amba_slave.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ata/atahost_ahbmst.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ata/ocidec2_amba_slave.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ata/atactrl_nodma.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ata/atactrl_dma.vhd
analyze -f VHDL -library gaisler ../../lib/gaisler/ata/atactrl.vhd
sh mkdir synopsys/esa
define_design_lib esa -path synopsys/esa
analyze -f VHDL -library esa ../../lib/esa/memoryctrl/memoryctrl.vhd
analyze -f VHDL -library esa ../../lib/esa/memoryctrl/mctrl.vhd
analyze -f VHDL -library esa ../../lib/esa/pci/pcicomp.vhd
analyze -f VHDL -library esa ../../lib/esa/pci/pci_arb_pkg.vhd
analyze -f VHDL -library esa ../../lib/esa/pci/pci_arb.vhd
analyze -f VHDL -library esa ../../lib/esa/pci/pciarb.vhd
sh mkdir synopsys/fmf
define_design_lib fmf -path synopsys/fmf
sh mkdir synopsys/cypress
define_design_lib cypress -path synopsys/cypress
sh mkdir synopsys/hynix
define_design_lib hynix -path synopsys/hynix
sh mkdir synopsys/micron
define_design_lib micron -path synopsys/micron
sh mkdir synopsys/work
define_design_lib work -path synopsys/work



set auto_wire_load_selection "true"
```

```
set_operating_conditions -library tcbn45gsbwptc NCCOM
#set sys_clk_freq 600.0
set sys_clk_freq 100.0
set clock_skew  0.10
set input_setup  1.8
#set output_delay 1.66
set output_delay 10.0

set sys_peri [expr 1000.0 / $sys_clk_freq]
set input_delay [expr $sys_peri - $input_setup]
set tdelay  [expr $output_delay + 1]

create_clock "clka" -name clk -period $sys_peri
set_dont_touch_network clka

set_false_path -from resetn
set_false_path -from testen
set_ideal_network testen
set_false_path -from rxd1
set_false_path -from dsubre
set_false_path -from dsuen
set_false_path -from dsurx
set_false_path -to dsuact

set_critical_range 2.0 leon3mp

set_max_delay $output_delay -from [all_inputs] -to [all_outputs]
```

timing_memories.tcl:

```
set auto_wire_load_selection "true"
set_operating_conditions -library tcbn45gsbwptc NCCOM
set sys_clk_freq 600.0
set clock_skew  0.10
set input_setup  1.8
set output_delay 1.66

set sys_peri [expr 1000.0 / $sys_clk_freq]
set input_delay [expr $sys_peri - $input_setup]
set tdelay  [expr $output_delay + 1]

create_clock "clka" -name clk -period $sys_peri
set_dont_touch_network clka

set_false_path -from resetn
set_false_path -from testen
set_ideal_network testen
set_false_path -from rxd1
set_false_path -from dsubre
set_false_path -from dsuen
set_false_path -from dsurx
set_false_path -to dsuact

set_critical_range 2.0 leon3mp

set_max_delay $output_delay -from [all_inputs] -to [all_outputs]
```

runleon3mp_vlog_simulation:

This script is used for the post-layout simulation of the Leon3-based MP-SoC design and its commands are executed in ModelSim.

```
#vlog leon3mp.v -v tcbn45gsbwp_sim2.v
vlog leon3mp_newgrlib_new.v -v tcbn45gsbwp_sim2.v
#sdfcom leon3mp_initial.sdf leon3mp.sdf

vlib micron
vcom -work micron ../../lib/micron/sdram/components.vhd
vcom -work micron ../../lib/micron/sdram/mt48lc16m16a2.vhd
```

```
vlib contrib
vcom -work contrib ../../lib/contrib/devices/devices_con.vhd

vlib gaisler
vcom -work gaisler ../../lib/gaisler/sim/sram_all_weather.vhd

vlib work
vcom ../../lib/work/debug/debug.vhd
vcom ../../lib/work/debug/grtestmod.vhd
vcom ../leon3-asic/config.vhd

vcom testbench_layout_sram.vhd

vsim -t 1ns -gdisas=1 -do "source wave.do; vcd file dhry_1.vcd; vcd add -r -file
dhry_1.vcd /testbench/*; vcd on; run -all; quit" work.testbench
```

leon3mp_px.tcl:

The current script performs the post-layout netlist's power analysis in Synopsys PrimeTime PX, based on the technology libraries defined in *.synopsys_pt.setup*.

```
read_verilog leon3mp_newgrlib_new.v
current_design leon3mp
set auto_wire_load_selection true
link_design
create_clock "clka" -name clk -period 5
read_vcd -strip_path testbench/d3 stanford.vcd
set power_analysis_mode averaged
update_power
report_power -hier -levels 7 -nosplit > leon3mp_stanford_1.1V.report
report_power
exit
```

ets.tcl:

This script is used for the extraction of the timing reports regarding the Leon3-based post-layout netlist, through the Encounter Timing System (ETS) static timing analysis engine.

```
read_lib tcbn45gsbwptc.lib
read_lib SRAM32x1024_old.lib
read_lib SRAM32x128.lib
read_lib SRAM32x64.lib
read_lib SRAM32x16.lib
read_lib SRAM8x128.lib
read_verilog leon3mp_45nm.v
set_top_module leon3mp
read_sdc leon3mp.sdc
set_analysis_mode -checkType setup
set_op_cond NCCOM -library tcbn45gsbwptc
read_sdf leon3mp_45nm_em.sdf
create_clock "clka" -name clk -period 4.6
report_timing -net
report_timing -machine_readable -max_points 10 > leon3mp_em_timing.txt
```

8

References

[1] Jin Guo, Antonis Papanikolaou, Michele Stucchi, Kristof Croes, Zsolt Tokei and Francky Catthoor, "The analysis of system-level timing failures due to interconnect reliability degradation", IEEE Transactions on Device and Material Reliability, 2009.

[2] Jin Guo, Antonis Papanikolaou, Michele Stucchi, Kristof Croes, Zsolt Tokei and Francky Catthoor, "A tool flow for predicting system-level timing failures due to interconnect reliability degradation", in Proceedings of the 2008 GreatLakes VLSI International Symposium (GLSVLSI), pp. 291-296, Orlando, Florida, USA, 2008.

[3] Dimitris Bekiaris, Antonis Papanikolaou, Dimitrios Soudris, George Economakos and Kiamal Z. Pekmestzi, "Reliability Breakdown Analysis of an MP-SoC platform due to Interconnect Wear-Out", in Proceedings of the 2nd Workshop on the Design For Reliability (DFR), organized in conjunction with the Fifth High-Performance Embedded Architectures and Compilers (HIPEAC) International Conference, Pisa, Italy, 2010.

[4] Yunlong Li, "Low-k dielectric reliability in copper interconnects", Phd Thesis, 2007.

[5] Syed M. Alam, "Design tool and methodologies for interconnect reliability analysis in integrated circuits", Phd Thesis, 2005.

[6] K. Croes, G. Cannata, L. Zhao and Zs. Tokei, “Study of copper drift during TDDB of inter-metal dielectrics by using fully passivated MOS capacitors as test vehicle”, Journal of Microelectronics Reliability, Volume 48, Issues 8-9, pp. 1384-1387, August-September 2008.

[7] Guido Groeseneken, Robin Degraeve, Ben Kaczer and Philippe Roussel, “Recent trends in reliability assessment of advanced CMOS technologies”, in Proceedings of the 2005 International Conference on Microelectronic Test Structures (ICMTS), 4-7 April, pp. 81-88, 2005.

[8] Zung-Sun Choi, “Reliability of copper interconnects in integrated circuits”, Massachusetts Institute of Technology (MIT), Phd Thesis, 2007.

[9] Sherkhar Borkar, “Design challenges of technology scaling”, IEEE Micro, vol. 19, issue 4, 1999.

[10] J. Bhasker and R. Chadha, “Static timing analysis for nanometer designs - A practical approach”.

[11] Robert Doering and Yoshio Nishi, “Handbook of semiconductor manufacturing technology(2nd edition)”.

[12] McPherson, Vijay Reddy, Kaustav Banerjee, and Huy Le, “Comparison of E and 1/E TDDB Models for Si02 under long-term/low-field test conditions”, in Proceedings of the IEEE Electron Devices Meeting (IEDM) 1998, 6-9 December, pp. 171-174, San Fransisco, California, USA, 1998.

[13] Wei Huang Ghosh, S. Velusamy, S. Sankaranarayanan, K. Skadron, K., M. R Stan and Charles L. Brown, “HotSpot: a compact thermal modeling methodology for early-stage VLSI design”, IEEE Transactions on VLSI Systems, vol. 14, no. 5, May 2006.

[14] Lu Zhijian; Wei Huang; Mircea R. Stan, Kevin Skadron and John Lach, “Interconnect Lifetime Prediction for Reliability-aware Systems”, IEEE Transactions on VLSI (TVLSI), vol. 15, issue 2, pp. 159-172, February 2007.

[15] HotSpot Thermal Analysis and Simulation tool, http://lava.cs.virginia.edu/HotSpot/.

[16] International Technology Roadmap for Semiconductors (ITRS), http://public.itrs.net.

[17] Cacti memory model, http://www.hpl.hp.com/research/cacti/.

[18] Cadence Encounter Timing System Text Command Reference, September 2008.

[19] Cadence Encounter Timing System User Guide, September 2008.

[20] Synopsys HSPICE Reference Manual: Commands and Control Options, September 2009.

[21] Cadence Encounter Database Access Command Reference, December 2005.

[22] Cadence Encounter Text Command Reference, September 2008.

[23] Aeroflex Gaisler Research, http://www.gaisler.com.

[24] IEEE Standard for Integrated Circuit (IC) Delay and Power Calculation System, June 1999.

[25] Mentor Graphics ModelSim, http://www.model.com.

[26] Leon3 BCC Cross-Compiler User Guide.

[27] George Faldamis, “Study of 2D and 3D MP-SoC architectures and implementation in FPGA platforms”, Diploma Thesis, National Technical University of Athens, Microprocessors and Digital Systems Lab (MicroLab), November 2009.

[28] OpenSparc Microprocessor Architecture, http://www.opensparc.net.

[29] Synopsys Library Compiler User Guide, http://www.synopsys.com.

[30] Synopsys Armenia (SAED) 90nm CMOS Standard-cell library specification manual.

[31] Synopsys Liberty Format User Guide, vol 2, version 2009.06.

[32] Gaisler GRLIB IP User Guide.

[33] N. Weste, D. Harris, "VLSI Design; A circuits and systems perspective", Addison-Wesley, 2004.

[34] A. Kahng and Q. Wang, "An analytic placer for mixed-size placement and timing-driven placement", in Proceedings of the 2004 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 565-572, November 7-11, San Jose, California, USA, 2004.

[35] Synopsys Design Compiler User Guide, http://www.synopsys.com.

[36] Dimitris Bekiaris, Antonis Papanikolaou, Christos Papameletis, Dimitrios Soudris, George Economakos and Kiamal Z. Pekmestzi, "A Temperature-Aware Time-Dependent Dielectric Breakdown Analysis Framework", accepted for presentation in the 2010 Workshop on Power and Timing, Modeling and Simulation (PATMOS), to be held in Grenoble, France, 7-10 September, 2010.