Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ens.ewi.tudelft.nl/

CAS-MS-2012-11

M.Sc. Thesis

The Impact of Low-Power Design Methodology on Digital Libraries

Ahmad Hamdy Elsayed

Abstract

In recent years, exciting new low-power design methods have been introduced, such as: multiple supply voltages, body bias techniques and power shut-off. In order to use these low power design methods, strict requirements for both libraries and tools are needed. An additional challenge is the introduction of more accurate characterization models for newer technologies (current source models like ECSM and CCS). This has made the task of library checking a serious issue that needs to be automated.

The main part of this thesis presents a checker tool that is used to verify the consistency of the different library formats (views) in standard cell libraries. The layout consistency checker in our tool checks the consistency of the layout of pins between GDSII and LEF library views; we devised a new algorithm,*Grid Formation and Centre Inclusion*, for this checker. The tool also verifies the pin consistency and availability of cells across other library formats, such as: Verilog and Liberty. The tool was tested using different technology libraries (such as 90nm and 40nm), provided by different vendors (such as GLOBALFOUNDRIES); multiple interfacing errors were caught using our library checker tool.

A second part at the end of the thesis shows experiments with some of the low-power design techniques used during the design of a digital block, using -for implementation- standard cells from one of the libraries that have been checked with the library checker tool. Benefits of using these techniques are evaluated and trade-offs are discussed. Power-Shut Off (PSO) design technique proved to be the most effective in reducing power consumption, with power savings that reached 20%.

The Impact of Low-Power Design Methodology on Digital Libraries

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Computer Engineering

by

Ahmad Hamdy Elsayed
born in Cairo, Egypt

This work was performed in and sponsored by:

NXP Semiconductors
Eindhoven

In collaboration with:

Circuits and Systems Group
Department of Microelectronics & Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Delft University of Technology
Department of
Microelectronics & Computer Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"The Impact of Low-Power Design Methodology on Digital Libraries"** by **Ahmad Hamdy Elsayed** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 21 September 2012

Chairman:

prof.dr.ir. A.J. van der Veen

Advisors:

dr.ir. Nick van der Meijs

dr.ir. Michel Berkelaar

Committee Members:

dr.ir. Zaid Al-Ars

ir. Eric Seelen

dr.ir. Philippe Soulard

Abstract

In recent years, exciting new low-power design methods have been introduced, such as: multiple supply voltages, body bias techniques and power shut-off. In order to use these low power design methods, strict requirements for both libraries and tools are needed. An additional challenge is the introduction of more accurate characterization models for newer technologies (current source models like ECSM and CCS). This has made the task of library checking a serious issue that needs to be automated.

The main part of this thesis presents a checker tool that is used to verify the consistency of the different library formats (views) in standard cell libraries. The layout consistency checker in our tool checks the consistency of the layout of pins between GDSII and LEF library views; we devised a new algorithm,*Grid Formation and Centre Inclusion*, for this checker. The tool also verifies the pin consistency and availability of cells across other library formats, such as: Verilog and Liberty. The tool was tested using different technology libraries (such as 90nm and 40nm), provided by different vendors (such as GLOBALFOUNDRIES); multiple interfacing errors were caught using our library checker tool.

A second part at the end of the thesis shows experiments with some of the low-power design techniques used during the design of a digital block, using -for implementation- standard cells from one of the libraries that have been checked with the library checker tool. Benefits of using these techniques are evaluated and trade-offs are discussed. Power-Shut Off (PSO) design technique proved to be the most effective in reducing power consumption, with power savings that reached 20%.

Acknowledgments

This project would not have been possible without the support of many people. I wish to thank, first and foremost, my supervisor dr.ir. Michel Berkelaar who offered me all the support and advice throughout my thesis, and guided me through difficult times to get this thesis work done in the best possible shape. I would also like to offer my sincerest gratitude to my mentors at NXP Semiconductors: ir. Eric Seelen , dr.ir. Philippe Soulard and dr. Ananta Majhi, who were abundantly helpful and offered me invaluable assistance, guidance and support throughout the whole thesis work.

Special thanks go to the people at NXP, who did not hesitate to help me, offering their valuable advice. I would like to thank Hans van Walderveen for his help and explanations about layer mapping between GDSII and LEF. Special thanks go also to Theo Beelen for his valuable discussions and feedback about the algorithm for the layout consistency checker. I also thank Andries van der Veen for his valuable help and discussions about the redhawk library format and the proper implementation of its checkers.

Deepest gratitude is also due to Matthias Koefferlein, the developer of the tool "Klayout - Layout Viewer and Editor", for his keen help in the conversion of binary GDSII files into readable ASCII format, and also his help in flattening the hierarchy of cells in GDSII format to match the flat hierarchy in LEF.

I would also like to convey thanks to Bob Stein (who is now an independent consultant specializing in hyper-useful user interfaces) for his help and valuable information about solving the Point-In-Polygon (PIP) problem during the implementation of our layout consistency checker algorithm.

I would like to thank prof.dr.ir. A.J. van der Veen , dr.ir. Nick van der Meijs , dr.ir. Zaid Al-Ars , ir. Eric Seelen and dr.ir. Philippe Soulard for their valuable time spent in my thesis defence committee.

I would also like to express my sincerest gratitude to NXP semiconductors for hosting this thesis work at its facilities in Eindhoven. The cheerful environment and friendly attitude of everyone in the company helped me a lot to adapt to the new city and environment during my stay in Eindhoven for the thesis period.

Finally I am truly and sincerely thankful for my family for supporting me every step of the way on continuing my studies in Delft University of Technology. Special thanks goes to my father, who has always encouraged me to do my best during my studies, and encouraged me on continuing my studies abroad despite his illness, but sadly he will not be able to enjoy the results with me for he passed away towards the middle of my thesis work. I dedicate this thesis work to him.

Ahmad Hamdy Elsayed
Delft, The Netherlands
21 September 2012

Contents

List of Figures

List of Tables

1 Introduction

In this chapter, we give an introduction about the importance of low-power design in our life these days, and how reducing the power reflects on our daily use of electronic products. We also give an overview about how the rest of this thesis work is organized and presented.

1.1 Motivation

Battery operated devices (cell phones, tablets, laptops) have an increasing demand on energy. And despite the efforts that have been made in researching new battery technologies to cover this demand, the need for low-power designs became a necessity, since battery capacity doubles approximately every 10 years according to Jan Rabaey [1] (that is 3-7% every year depending on the introduction of new technologies); and such growth lags behind *Moore's law* (which demands doubling the computational complexity every 18 months), and the resulting need for more energy resources to feed those hungry devices. The computational density of these devices, and hence their power usage (for computations and cooling), is of growing importance.

From a design perspective, energy usage can be optimized at different levels of abstraction starting from the system and RTL description level, reaching to the circuit level. Major gains can be achieved in reducing energy and power consumption of devices at each level of these levels; and due to the importance of low-power design, considering the direct impact it has on individuals (battery life) and industries (cooling costs), design for low-power is becoming increasingly important in the IC-design industry, making it a key competitive metric in this field, beside the usual metrics such as the performance (throughput, latency and frequency) and costs (area and packaging), which have been traditionally grabbing the major portion of attention of the designers.

1.1.1 Low Power in System on Chip (SoC) designs

From the previous discussion in Section 1.1, we can see that managing energy is one of the most important concerns nowadays. As the power density of SoC increases at such an alarming rate (as revealed by Figure 1.1), power management has become one of the major design concerns in SoC design. This has been pointed out by Fred Pollack of Intel as early as 1999 in his keynote at MICRO-32 [2].

There is an increasingly widening gap between the power density trend and the power design requirements, as illustrated in Figure 1.2. Aveek Sarkar from *Apache Design Solutions Inc.* explains this figure in his article "An RTL to GDSII approach for low power design: A design for power methodology" [3] as follows: the red curve illustrates the increasing levels of power that a design generates to meet advanced func-

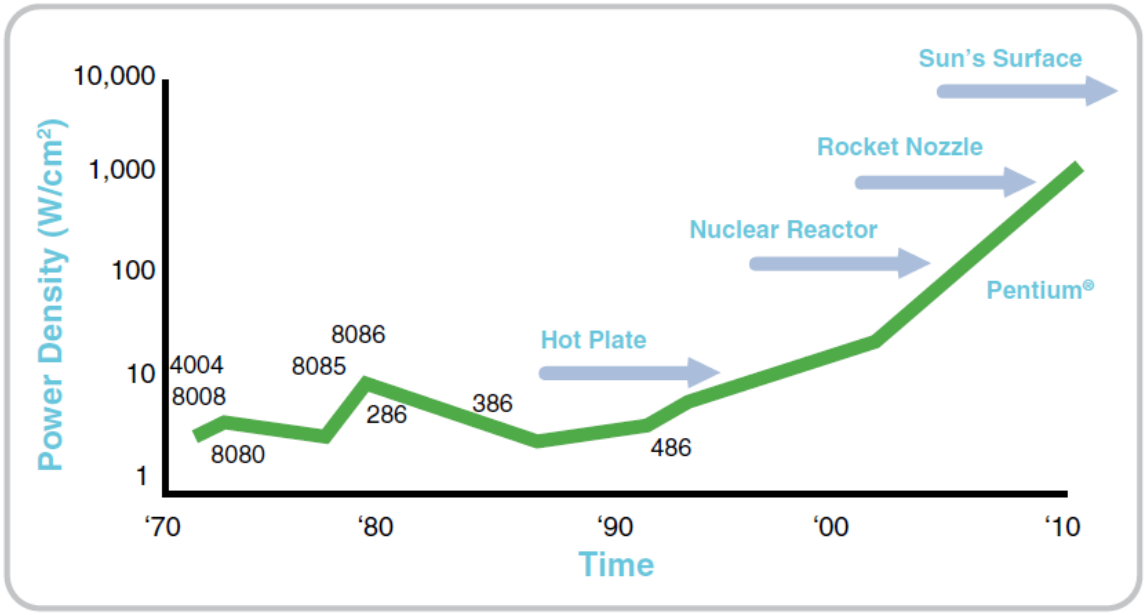


Figure 1.1: SoC Power Density. Courtesy Intel Corporation [2]

tionality needs (assuming no low power approaches are used), versus the yellow curve that bounds the maximum power consumption allowed in such systems (as specified by power budgets for these systems to achieve operational and standby power targets). This wide gap represents a huge problem for the designers of electronic products (wired and wireless) nowadays. The graph also shows that more design efforts should be directed towards managing power, which will certainly have a direct impact on engineering productivity, as it impacts schedules and risks for meeting deadlines and keeping pace with the fast moving demands of the market.

1.1.2 Wide Range of Applications

The explosive growth of applications depending mainly on batteries also made power management an inevitable must. Quoting *Silicon Integration Initiative Inc. (Si2)* in their book "A Practical Guide to Low-power Design" [4]: today's portable products are expected not only to be small, cool, and lightweight, but also to provide extremely long battery life; and even wired communications systems must pay attention to heat, power density and low-power requirements. Jan Rabaey [1] mentioned that in the year 2000, 400 Millions of personal computers worldwide were being used, which was assumed to consume 0.16 Tera kWh per year, the power equivalent to 26 nuclear power plants; and this needed over 1 Giga kWh per year just for cooling. He also talked about centralized data centres, and explained (quoting Luis Barosso from Google) that the cost of a data centre is determined solely by the monthly power bill, not by the cost of hardware or maintenance. This bill results from both the power dissipation in the electronic systems and the cost of removing the dissipated heat –that is, air conditioning. All of this further proves that power management is becoming a necessity for wired electronics as

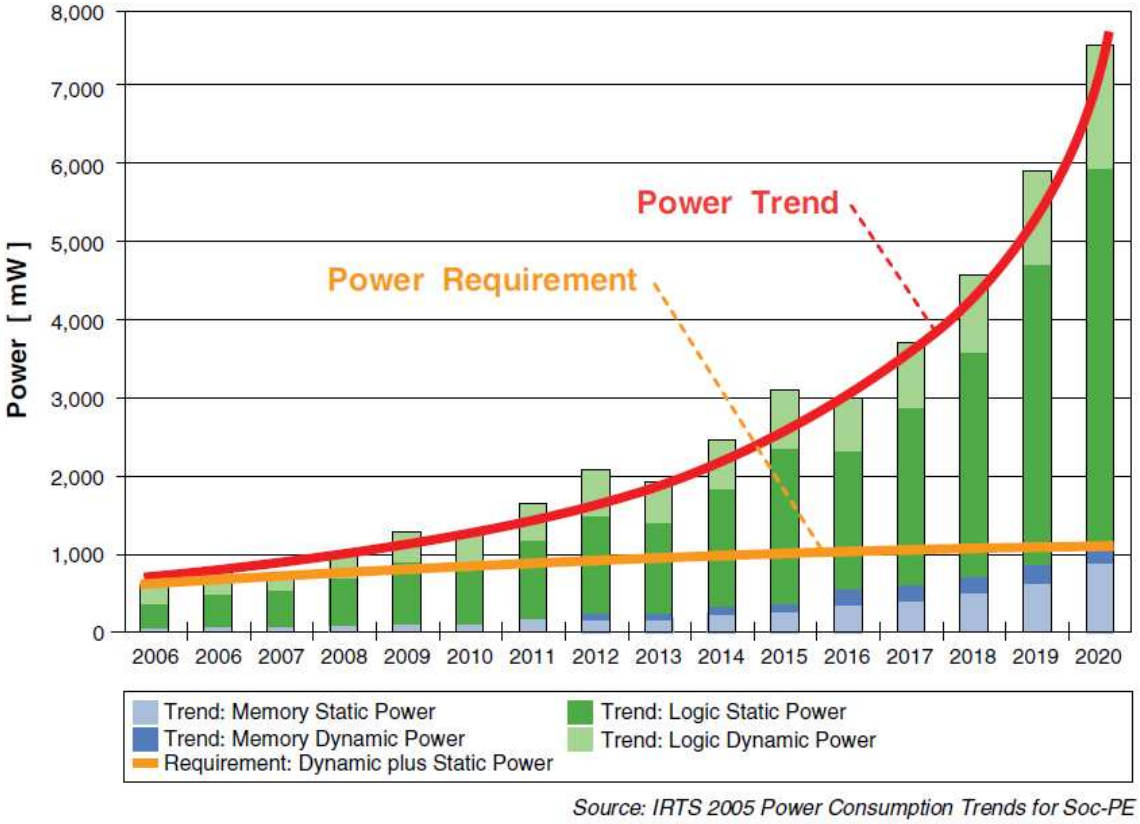


Figure 1.2: IC power trends: actual vs. specified. [3]

much as it is a must for wireless electronics. Si2 also categorizes the products requiring low-power management in their book [4] as follows:

1. Consumer, wireless, and handheld devices: cell phones, personal digital assistants (PDAs), MP3 players, global positioning system (GPS) receivers, and digital cameras.

2. Home electronics: game consoles for DVD/VCR players, digital media recorders, cable and satellite television set-top boxes, and network and telecom devices.

3. Tethered electronics such as servers, routers, and other products bound by packaging costs, cooling costs, and Energy Star requirements supporting the Green movement to combat global warming.

1.1.3 Effective Low Power Management

Many power reduction efforts have been targeting low levels of abstraction in the design (circuit level); this has been sufficient for a number of years, since managing power at low abstraction levels has been paying off, reducing the power within acceptable limits; however, an effective power and energy management for a System-on-Chip must start

at earlier stages of the design, as early as the system and design-architecture phases. Standard cell methodology plays an important role in implementing designs with mostly digital-logic features. As explained by Wikipedia [5], "cell-based methodology (the general class to which standard cells belong) makes it possible for one designer to focus on the high-level (logical function) aspect of digital design, while another designer focuses on the implementation (physical) aspect".
In order to be able to apply low-power design techniques (some of which are discussed in later chapters), standard cells have to be adapted to enable this application, and also new cells (such as retention and isolation cells, which are discussed later in more details) have to be added to the old standard cell libraries. Caution must be paid in making sure that standard cells have consistent descriptions and interfaces across all library formats (discussed in later chapters) present in the standard cell library, since any inconsistency that may exist across any of those library views can cause serious troubles for chips, ones that might not even be discovered until the chip is already put into production, and then the costs for fixing the design flaw resulting from the inconsistency across library views would be enormous (not to mention losing credibility of design companies at their customers). All this creates the need for developing *Library Checker* tools, which will make sure all library views of standard cells are consistent, to avoid the risk of design flaws for something that can be fixed in advance.

1.2 Project Sketch

The main contribution of this thesis work will be in the field of standard cell library checking. Due to the huge size of standard cell libraries, and the new libraries frequently produced following new process technologies, the process of checking these libraries manually became an impossible task, and this raised the need for automated tool that does the job. This will be the primary part of this thesis work -building an automatic tool which will check the different views of libraries and make sure those views are all consistent for all the standard cells in those libraries, and report any missing cells or attributes or any inconsistency that might be found. A number of library views (explained more in Section 2.2) have been checked for consistency, namely:

- GDSII
- LEF
- Liberty
- Verilog
- Netlist (CDL)
- Redhawk (APL)

The most important and the highest weight checker is the layout-consistency-checker (LEF-GDSII checker), since it requires a lot of file manipulation and the development of a new algorithm to compare geometrical shapes. This checker (as will be discussed

in more details in Chapter 3 & Chapter 4) mainly checks the consistency and matching of the layout of pins in the two formats: LEF and GDSII. What we need to do for checking the consistency of layers between LEF and GDSII is to check that all of the polygons representing the layout of each pin in LEF are completely included inside the corresponding GDSII polygons for each layer of each pin in each standard cell. The approach for such a checker (from algorithm to implementation) is presented and discussed. A second part of the thesis work employs some of the standard cells (checked by the checker) from one of the recent standard cell libraries in a simple low-power incrementer design, using two of the most common low-power design techniques, and comparing the power figures with the baseline design (the one that does not use any low-power design techniques), in order to assess the effectiveness of these cells in achieving the low power intents described by the low-power techniques used.

1.3 Thesis Overview

The organization of the report is as follows: some background about the most common low-power design techniques and different library views is given in Chapter 2. Chapter 3 discusses the approaches suggested to implement the *Library Checker* tool, and some details about preparations for this tool; it also talks more about the different types of checkers implemented in the *Library Checker* tool, and results from running the tool on existing standard cell libraries. Chapter 4 presents the algorithm for implementing the Layout Consistency Checker. Chapter 5 summarizes the low-power incrementer design experiment, using actual standard library cells (checked by the *Library Checker* tool) to assess the effectiveness of low-power techniques in reducing power consumption, and the ability of these standard cells in achieving low-power design intents and properly implementing low-power techniques. At the end, conclusions and possible future work are discussed in Chapter 6.

2 Low-Power Design

In order to be able to reduce the power consumption in designs, we have to be aware of the sources of power dissipation. Equation (2.1) shows the main factors contributing to power dissipation. This equation reveals the main components of the *total power* dissipation in CMOS SoCs as: *dynamic power* and *leakage power*.

$$Power = P_{switching} + P_{short-circuit} + P_{leakage} \tag{2.1}$$

Dynamic power is mainly the sum of switching power and short-circuit power. The former comes from charging or discharging of internal gate capacitances and net capacitances, and thus it depends mainly on the switching activity, switching frequency, supply voltage and effective capacitance, as shown in (2.2); while the latter is dissipated as a result of the instantaneous short-circuit connection between the supply voltage and the ground at the switching-time of the gate, and thus depends on the switching activity, the switching frequency, the supply voltage and the amount of short-circuit current, as shown in (2.3). Figure 2.1 illustrates the two components of dynamic power consumption in CMOS.

$$P_{switching} = \alpha \times f \times V_{dd}^2 \times C_{eff} \tag{2.2}$$
$$P_{short-circuit} = \alpha \times f \times V_{dd} \times I_{sc} \tag{2.3}$$

where α is the switching activity, f is the switching frequency, V_{dd} is the supply voltage, C_{eff} is the effective capacitance and I_{sc} is the average short-circuit current during switching.

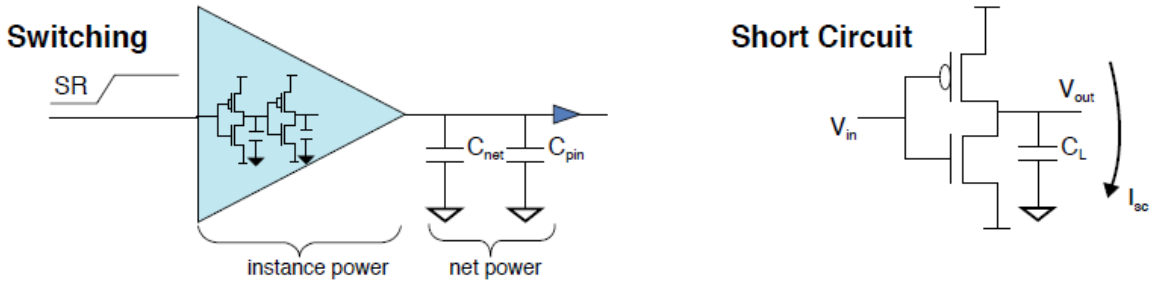


Figure 2.1: Dynamic power in CMOS. Source: A Practical Guide to Low-Power Design [4]

As for the leakage power, this basically depends on the supply voltage V_{dd}, the threshold voltage V_{th} and the sizing (W/L) of the transistor; this can be observed in (2.4), which represents the value of the subthreshold leakage current for a MOSFET

device (as explained in [6]), the most important component contributing to leakage in CMOS.

$$I_{SUBTH} = I_0 \times e^{\frac{V_{gs}-V_{th}}{nV_T}} \times [1 - e^{\frac{-V_{ds}}{V_T}}] \tag{2.4}$$

where $I_0 = \frac{W \times \mu_0 \times C_{ox} \times V_T^2 \times e^{1.8}}{L}$, $V_T = \frac{K \times T}{q}$ is the thermal voltage, V_{th} is the threshold voltage, V_{ds} and V_{gs} are the drain-to-source and gate-to-source voltages respectively. W and L are the effective transistor width and length, respectively. C_{ox} is the gate oxide capacitance, μ_0 is the carrier mobility and n is the subthreshold swing coefficient and is given by: $n = 1 + \frac{C_D}{C_{OX}}$ (according to [7]), where C_D is the depletion channel region capacitance per unit area.

With the shrinking geometrics of process technologies, power management becomes a must for all designs of 90nm and below. At these small geometrics, leakage current becomes of great importance as it becomes comparable with switching currents, which makes it a primary source of power dissipation in CMOS. This is illustrated in Figure 2.2.

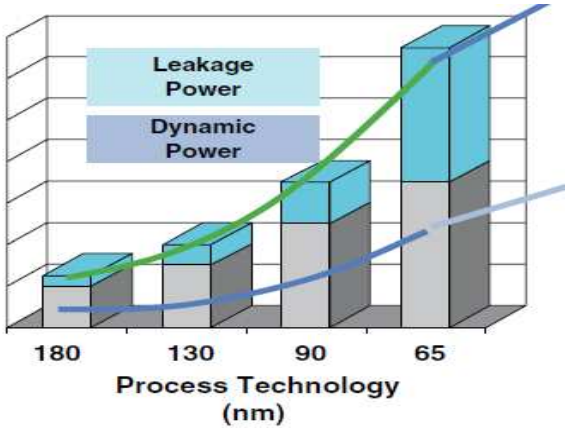


Figure 2.2: Process technology vs. leakage and dynamic power. Source: A Practical Guide to Low-Power Design [4]

In order to be able to reduce power consumption in SoCs, we have to tackle each of the aforementioned parameters of each component contributing to the different types of power consumption. Low-power design techniques work mainly on reducing one or more of these parameters, in order to reduce the two components of power consumption, dynamic and leakage. Keep in mind that leakage currents are becoming of growing importance (with the developments in process technology and the shrinking geometrics of CMOS with each new process technology), which makes low-power design techniques targeting leakage increasingly important at such small geometrics. We will present some of the common low-power design techniques in the rest of this chapter, along with some definitions for the different views in standard cell libraries, which contain the cells helping to put those low-power design techniques in action.

The organization of the chapter is as follows: some common *Low-Power Design Techniques* are presented in Section 2.1. Next, important *Standard Cell Library Views*

are explained in Section 2.2. Afterwards, the importance of *consistency* across different library views, and the impact of inconsistencies on the design cycle are discussed in Section 2.3. Finally, a summary of the chapter is presented in Section 2.4.

2.1 Low-Power Design Techniques

As discussed in the introduction, reducing power consumption requires targeting one or more parameters constituting the different components contributing to the total power consumption in a SoC design. In this section we discuss some of the most important low-power techniques, along with the parameters they tackle.

2.1.1 Power Shut Off (PSO)

This low power technique is considered the most effective in reducing power consumption of the design, and it can be applied at early stages of the design (as early as the RTL level). It allows the designer to completely shut down certain blocks (that are not in use for a certain period of time) in the design in order to save their power consumption, and consequently, this technique greatly reduces the static (leakage) power consumption, targeting the V_{th} parameter.
Having a good idea about the activity figures of instances and functional blocks inside the design, designers can divide the design into different power domains according to their activity, and can completely switch off any of these power domains (whenever they have no activity), which will drastically reduce the power consumption of the design. Specific power-down and power-up sequences (which are explained shortly) need to be followed in order to guarantee the correct functionality of the design when using this low-power technique. Incorrect power-up/down sequences are the main cause for failures in low-power SoC designs. Power sequences must be aggressively verified by verification engineers in order to make sure the design functions properly during power down periods (with the powered-down switchable power domains), and also that the design continues to function as expected on powering up the switchable power domains in the design.
Deploying this low-power design technique requires the usage of special low-power cells, all of which should be readily available in the standard cell library. These are discussed in the following part, along with the correct power-down & power-up sequences for correct functionality of the design.

2.1.1.1 Power Switch Cells (Power Gates)

In a design with power switching, either header or footer type power switch cells are required to supply power for cells that can be powered down. A header type cell connects the power rail (VDD) to the power pins of the cells. A footer type cell connects the ground rail (VSS) to the ground pins of the cells. An enable signal controls the connection of the switch to the respective power or ground rail.
The number of power switch cells inserted into the design must be properly chosen during design time to satisfy IR drop and current density requirements of the design, since

an excessive number of power switch cells will waste the silicon area (and consequently increase the cost of the chip), and also a small number of switch cells might cause a big amount of rush current (the maximum, instantaneous input current drawn by an electrical device when first turned on [8]) to flow through the switches at power-up. Obviously, there will be some power overhead for the insertion of this kind of cells in a low-power design because these cells need to be always-on, which is another reason why careful insertion of these cells should be taken into consideration during design. The insertion of such cells also adds some area overhead to the design, and more verification efforts in order to ensure their correct operation in switching the power on & off.

2.1.1.2 Retention Cells

Retention flops are sequential cells that can hold their internal state when the primary power supply is shut down, and that can restore the state when the power is brought back up. These cells are used (replacing normal flops in the design) in order to save the state of the design before going into power down (sleep) mode, so that we can restore the same state of the design after waking up from the sleep mode, and continue the normal operation of the design without affecting its functionality.
A key area of verification in the design cycle is to make sure that this kind of flops are actually retaining the state during power down periods, and are correctly restoring those states at power-up of their respective switchable domains. It is worth mentioning also that this type of cells requires a secondary power supply to be used during power down periods, so as not to lose their state when the primary power supply is shut-off. This is simply illustrated in Figure 2.3.
The insertion of retention cells also has the drawback of adding some power (since they need to be always powered-up) and area overhead to the design, and also needs a great deal of effort for verifying their correct operation when integrated within the design.

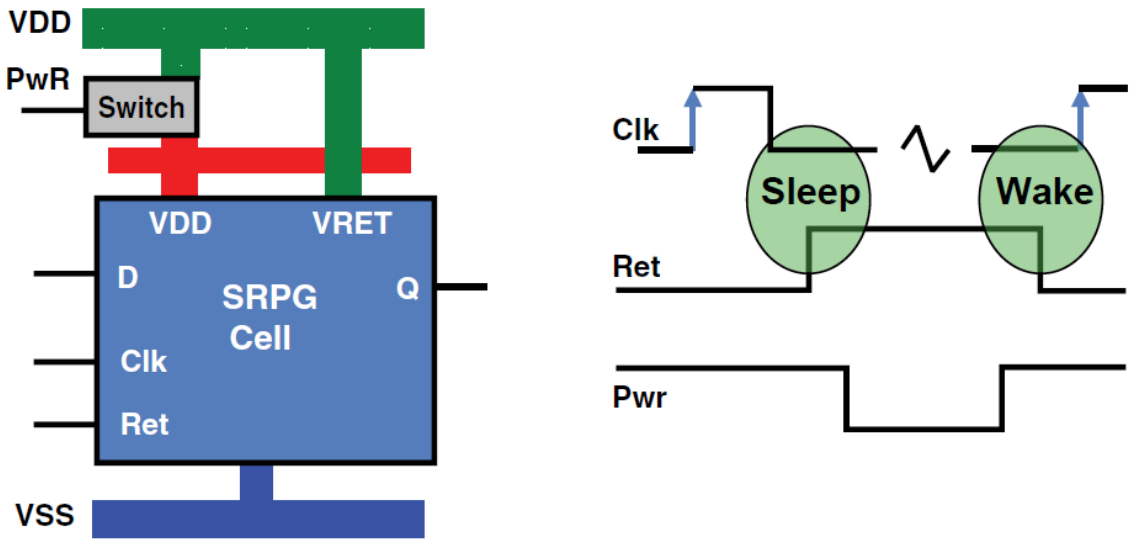


Figure 2.3: State Retention Power Gating. Source: A Practical Guide to Low-Power Design [4]

2.1.1.3 Isolation Cells

These are special cells required at the interface between blocks which are shut-down and always-on blocks to prevent floating states of unpowered signals from propagating from a power domain that is powered-down to a power domain that remains on. They clamp the outputs of power-down blocks to a known voltage. These cells can be placed in always-on or switchable regions, and the enable signals of the isolation cells are necessary to isolate floating inputs. Clamp values might be *1*, *0* or *latch* (retaining the final value on the latch before isolation) state. The location where the isolation cells are placed (source or destination block) is usually provided by the standard cell library vendor. Figure 2.4 illustrates a simple example for the usage of an isolation cell.
The same arguments (like the ones discussed previously for power-switch and retention cells) for deployment costs also hold here for isolation cells.

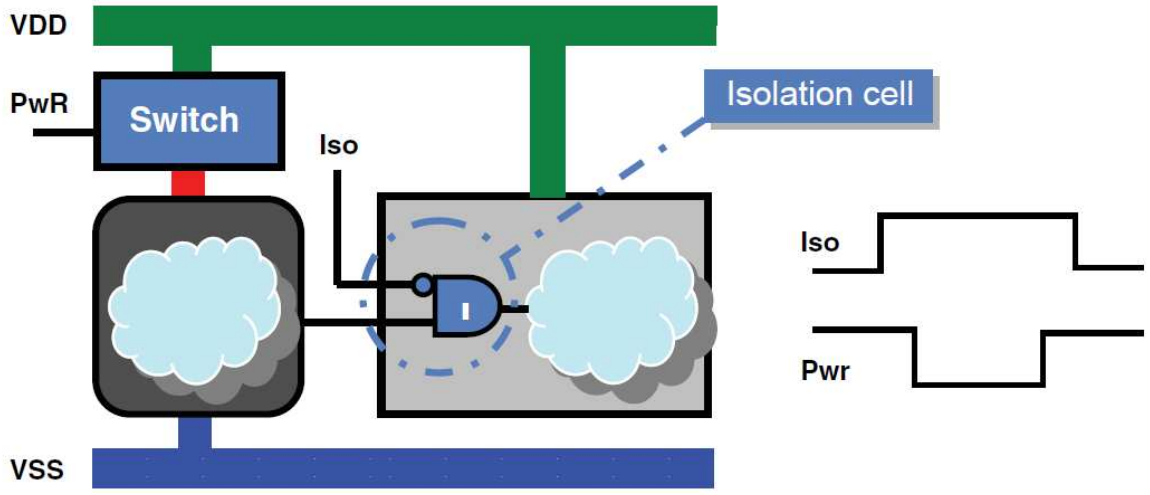


Figure 2.4: Isolation cell and Power Gating. Source: A Practical Guide to Low-Power Design [4]

2.1.1.4 Always-On Cells

Always-on cells are cells that remain active even if the main power for the block where they are placed is switched off. This is done through the use of a backup power rail (VDDR in Figure 2.5). These cells are useful for the cases of signals that need to be always active, regardless of the activity of their respective power domains (like reset signals passing through an instance in a switchable power domain and feeding a successive instance in an always-on power domain).

2.1.1.5 Power Cycle Sequence

As we mentioned before, for a correct operation of a design using this low-power design technique, a certain sequence must be followed during the power-down and power-up phases. For power-down, the general sequence followed is: first activating isolation

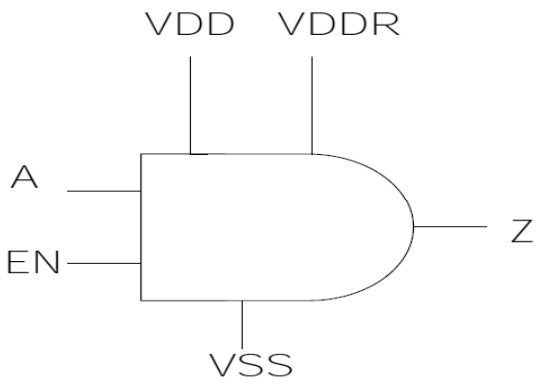


Figure 2.5: Always-On Cell

(in order to preserve the outputs of switchable power domains at expected values, and prevent floating values from propagating to non-switchable power domains); next, state retention is activated to preserve the states of key control registers in the design that are required to be restored at power-up; and finally power shut-off, as shown in Figure 2.6. For the power-up cycle, the reverse sequence needs to be followed. This way we guarantee the power cycle sequence will complete without any disturbances in the functionality of the design.

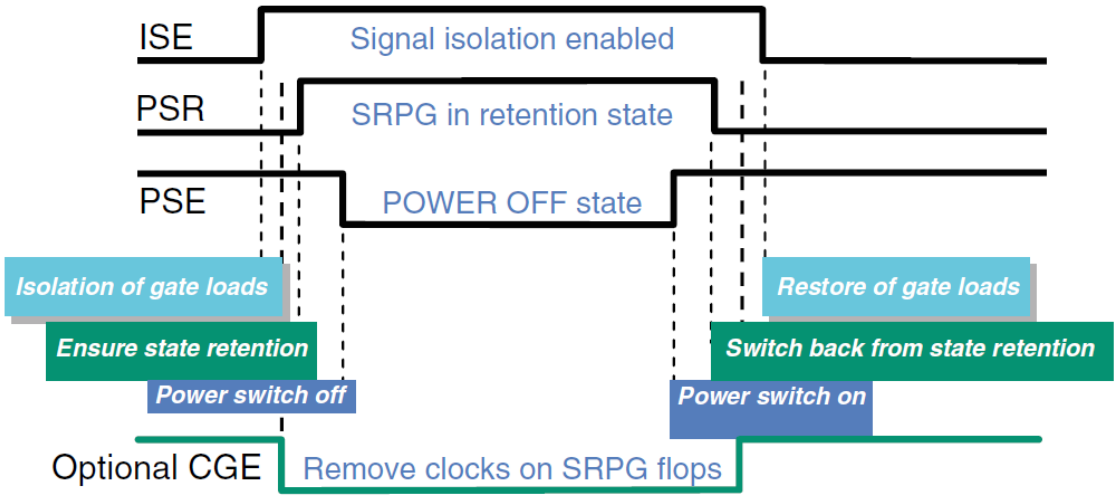


Figure 2.6: Power-down/up sequence. Source: A Practical Guide to Low-Power Design [4]

2.1.2 Multiple Supply Voltage (MSV)

In this technique, the design has more than one static voltage source and each domain is assigned the lowest voltage that still makes timing; thus this low-power design technique targets the V_{dd} parameter. The reduction in the supply voltage has a

squared effect in reducing the dynamic (active) power consumption (as can be observed from (2.2)) and also contributes greatly in reducing the leakage power. However, scaling down the supply voltage has a down-side on degrading the performance, that's why designers use different supply voltages for different parts of the chip depending on their performance requirements, and thus the name: "Multiple Supply Voltage". For the correct application of this low-power design technique, we also need to use special type of low-power cells to guarantee the correct functionality of the design, and in this case, we only need to use *Level-Shifter* cells.

Level-Shifter Cells

In a multi-voltage design, a level shifter is required where a signal crosses from one power domain having a certain supply voltage to another one having a different supply voltage. The level shifter operates like a buffer or an inverter with one supply voltage at the input and a different supply voltage at the output. Thus, a level shifter converts the voltage of the signal from one level to another, in order to save the logical value of the signal.
These cells have three categories, namely: up (shifting from low to high voltage), down (shifting from high to low voltage) and up/down level shifters. The placement of these level-shifter cells (whether in the source or the destination power domain) also depends on the constraints imposed by the library-vendor.

2.1.3 Multiple Threshold Voltage

Scaling down the threshold voltage of transistors has a great impact on lowering the delay in CMOS circuits. However, scaling down V_{th} is accompanied with an exponential increase in the sub-threshold leakage currents. To tackle the problem of high leakage, dual (or multiple) V_{th} techniques have been proposed. Low-V_{th} cells are used on the critical path of signals where speed is of great concern, while high-V_{th} cells are used on the non-critical paths. That is why standard cell libraries' vendors have to provide libraries that have cells with different threshold voltages. Designers mainly rely on synthesis tools for choosing the suitable V_{th}-cells from the available multi-threshold library-cells in order to achieve the area and performance targets, while minimizing power dissipation. This technique works on reducing the leakage power, targeting the parameter V_{th}.

2.1.4 Dynamic Voltage & Frequency Scaling (DVFS)

Dynamic voltage frequency scaling (DVFS) reduces the power in the chip by scaling down the voltage and frequency when peak performance is not required. Thus, this technique mainly aims at reducing dynamic power consumption, and a bit of leakage power consumption, targeting mainly f and V_{dd} parameters. A design using DVFS can be seen as a special case of an MSV design operating in multiple design modes.

- In a pure MSV design different portions of the design operate on different voltages and these portions remain operating at their respective operating voltage.

- In a DVFS design, in addition some portions can dynamically change to other voltages depending on the design mode or can even be switched off.

Consequently, a DVFS design must satisfy different constraints in different design modes. DVFS designs require variable power supply(ies) that can generate the required voltage levels with minimal transition energy losses and a quick voltage transient response. When scaling the voltage, the frequency must be scaled accordingly to meet signal propagation delay requirements. A power scheduler is usually used to compute the appropriate frequency and voltage levels needed to execute the various applications.

2.1.5 Other Low-Power Techniques

Multiple other low-power design techniques exist in the continuous efforts of reducing the power consumption of SoC designs. Some of them are:

- Clock Gating: in which clock signals are disabled at specific times where there is no activity to reduce the dynamic power consumption, targeting the parameter α.
- Body Biasing: substrate is biased to a voltage different from V_{dd} in PMOS, and to a voltage different from V_{ss} in NMOS, in order to reduce the leakage power consumption targeting the parameter V_{th}.
- Operand Isolation: whenever data-path elements are not active, prevent it from switching using an enable signal. This reduces the dynamic power consumption, targeting the parameter α.

2.2 Standard Cell Library Views

It can be easily observed from the previous discussion in Section 2.1 that deploying low-power design techniques requires some adaptations of standard libraries. These adaptations include the addition of new low-power cells (like retention, isolation, and level-shifter cells). Adaptations also include changing the modelling of some cells to achieve low-power, such as: changing the cell-drive strength and supply/bias pin modelling. Standard Libraries have many different views for describing the different attributes of cells. The following section gives more information about the different library views in standard libraries.

- Graphic Database System (GDSII) Format: a binary file format representing planar geometric shapes, text labels, and other information about the layout in hierarchical form. In standard libraries, GDSII files are used to describe the layout of each layer for each pin of the cell.
- Library Exchange Format (LEF): includes the abstract of cells and the rules for routing, but no information about the internal netlist of the cells.

- Liberty Format: liberty is the gate-level modelling technology; an open industry standard for longer than a decade, the liberty library format is used as the library model exchange for timing, noise, power and test behaviour.
- Verilog Format: is most commonly used in the design, verification, and implementation of digital logic chips at the RTL level of abstraction.
- Netlist (CDL) Format: usually conveys connectivity information and provides nothing more than instances, nets, and some attributes (such as 'W' & 'L' of transistors).
- RedHawk (APL) Format: a binary file format that provides an integrated environment for analysing power, noise and reliability issues in SoC designs.

2.3 Consistency Across Different Library Views

Different library views have to be consistent in their description and specification of standard cells. Such consistency is represented in checking the following conditions:

- Existence of cells: All functional cells must be present in all library views (formats).
- Consistency of cell names: Each cell must have the exact same name (case-sensitive) across all formats
- Consistency of pin-interface of each cell: Each cell must have the same pin-count and exact pin-names in all library formats; also the modelling of pins (pin type, direction, etc..) must be consistent for each cell across all library formats.
- Matching layout geometrics of pins for each cell: Geometrics of each layer of each pin in each cell has to be matching in formats describing the layout of cells (basically LEF & GDSII formats)

Inconsistencies across the different library views might cause errors during the design cycle at different abstraction levels. Figure 2.7 shows a mismatch in the layout of a standard cell that was caught by our library checker tool; this kind of mismatches can cause serious routing problems. Some of these library-inconsistency problems might not even be caught during verification, and might leak into a design that is already put into production; this might cause the chip to fail when used by the customers, causing a potential loss of reputation and credibility for design companies.
Absence of body-bias pin modelling caused a big design company two major failures last year. In that case there were two different always on cells, one with a separate well for the *PMOS* transistors and another with abutted well. Due to the lack of bias modelling, design tools could not distinguish between the two, and picked (based on smaller area) the wrong one, effectively making the bias pin operate in forward mode around 0.6 volts, creating too much leakage current. So the designer tried to design a low leakage design using low power design techniques, ending up with a design that was actually leaking way more; and this came as a result of lack of consistency of cells' pin-interface.

From this discussion, we conclude that the need for a library checker is crucial; one thing that saves a lot of effort hunting for bugs in the design due to inconsistencies in the libraries. And due to the huge and ever-increasing size of libraries, these checkers need to be automated in order to be run directly to check the consistency of libraries provided from library vendors in a simple run. And this reveals the importance of the job done here in this thesis work.

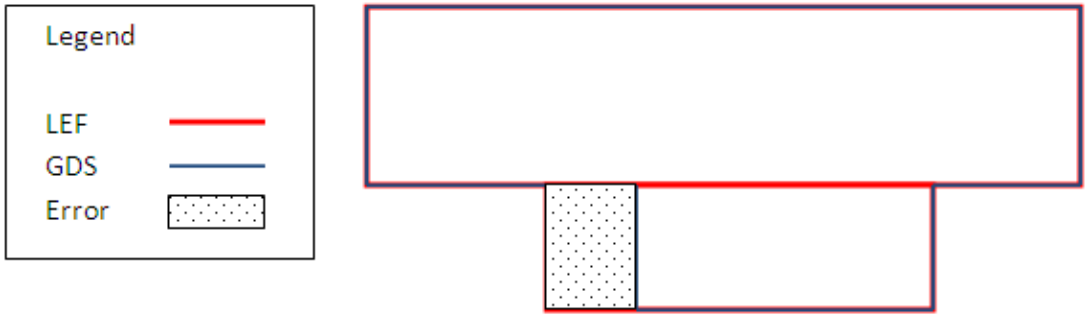


Figure 2.7: Example layout mismatch in one of the libraries checked by our tool

2.4 Summary

Various low power design techniques were introduced in this chapter. The deployment of these low-power design techniques requires certain adaptations for standard libraries. A brief overview about the most important library views (formats) was introduced. In the end, the importance of the consistency of the different library formats was discussed, which calls for an immediate action implementing an automated tool to do the library-checking job, which is the core of this thesis work.

3 Library Checkers

The previous discussion in Section 2.3 reveals the importance of verifying the consistency of the description and modelling of standard cells in the different library views. This verification process needs to be automated, the reason for which is the huge size of standard libraries. This initiated the idea for building our *Library Checker*, which is the core of this thesis work, and the main topic of the rest of this chapter and the next one as well.

The organization of the chapter is as follows: an overview about the studied approaches for implementing the Library Checker tool is given in Section 3.1, followed by different types of checkers implemented in the tool given in Section 3.2. Preparations for the tool from manipulating files to building parsers and similar issues are discussed in Section 3.3. Some results from running the checkers are presented in Section 3.4. It is worth mentioning here that this chapter does not discuss the algorithm used for the *Layout Consistency Checker*; this will be discussed in the following chapter -Chapter 4, since this part is of great importance and contains a lot of details. Results for running the *Layout Consistency Checker* are also presented in the following chapter. This chapter ends with a summary about what has been discussed in Section 3.5.

3.1 Suggested Approaches

The most important part of the checker tool is the *Layout Consistency Checker*; that's why all the initial thoughts and brainstorming ideas about the implementation of the tool were directed towards finding the best solution to do the layout consistency job between LEF & GDSII. These thoughts are briefly discussed in this section.

3.1.1 Using Cadence's Abstract Generator Tool

The first idea that popped in mind when brainstorming about the layout checker was to make sure that this work is not already implemented, and to see if there is a working solution that is readily available already, which could be incorporated inside our Library Checker tool. Doing some research, we came to know about *Virtuoso Abstract Generator* [9] provided by Cadence Design Systems as a part of their tool *Virtuoso Layout Suite*. This tool (as described in the manual) "is a library modelling tool that lets you create abstracts for standard cells, macro blocks, and IOs from detailed layout information in Library Exchange Format (LEF), Design Exchange Format (DEF), and Graphics Design Station II (GDSII) Stream formats". A first look at this looked like it was the proper tool for the job. So we first decided to try this path, which will save us the pain of implementing a new tool that does the job that Cadence's tool does already. However, we found multiple issues when comparing the LEF file produced

from *Abstract Generator* and the original LEF file of the library, the most critical of which has to do with discrepancies in the geometry of layers for the power and ground pins between the two files. We tried to tweak the tool's options in order to know the reason for which these differences appeared, but then we decided to drop this path for the following reasons:

1. This solution will mainly depend on a tool provided by Cadence, the licence for which is not free, and needs to be purchased from Cadence.
2. The dependency of a big part of our Library Checker tool (the *Layout Consistency Checker*) on a non-open-source tool limits the ability for further/special developments, manipulations or enhancements for the tool.
3. The complication of usage of *Virtuoso Abstract Generator* as an intermediate step for the implementation of our Library Checker tool was not satisfying.
4. Discrepancies of geometries of layers between original LEF and the LEF generated from *Abstract Generator* would not give correct errors about mismatching geometries, and consequently produce bugs which reduce the reliability of the tool.

3.1.2 Implementing an Independent Tool

After dropping the direction of using the *Abstract Generator* for checking the consistency of the layout of cells, and getting no good results from further searches for finding a solution to do the layout-consistency-checking job, we decided that implementing the whole tool ourselves will be the best option; especially that there has been already some development in-house (at NXP) for the basis of the *Library Checker* tool, which came as a great aid afterwards in completing our tool. The tool's input arguments should be the path (relative or absolute) to the library package that needs verification, and may also include some optional miscellaneous arguments (discussed in more details in Section 3.3.5). Output of the tool should be a list of errors for the inconsistencies revealed in the library package being checked. Different parts of the *Library Checker* tool are illustrated by the block diagram in Figure 3.1. Numbering the rules of the checker plays an important role in grouping similar errors, and displaying them consecutively (according to the rule number), for better categorization of the output errors of the checker.

3.2 Checkers Implemented

The *Library Checker* tool needs to verify the consistency of standard libraries through the verification of the different conditions mentioned in Section 2.3. Based on these conditions, we built various checkers, as explained in the following section. The checkers (as well as the whole tool) are built in Tcl [10], to extend the part of the checker already developed in-house at NXP.

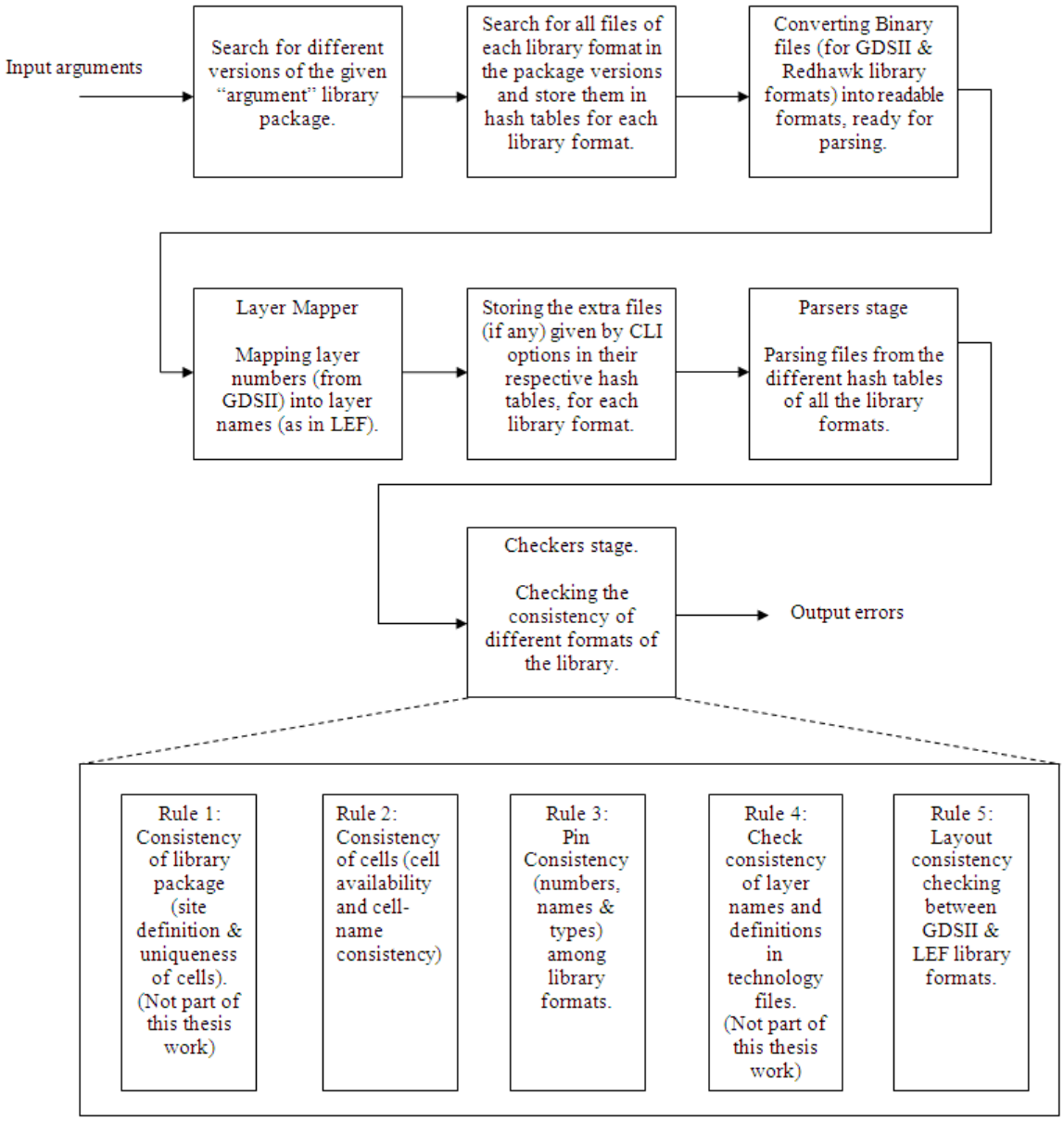


Figure 3.1: Block diagram of the Library Checker tool

3.2.1 Cell Name Consistency Checker

This checker verifies the consistency of the names of different cells across the different views of the standard cell library. In each library view, each cell is given a unique name that gives a hint about the functionality of the cell. This name must be exactly matching to the corresponding name in other library views, including the case-match of the cell name, since a case-mismatch in the cell name will be reported as missing cells by design tools, a scenario that has been reported by designers at NXP for cells having a name-case-mismatch in some standard cell library formats. For each cell in

each view, the checker searches for the same exact name of the cell (case-sensitive) in all other library views; if the exact name was not found, the checker then relaxes the condition and checks that the name does not exist in a different *case*; if the name was found in a different *case*, the checker reports an error that there is a case-mismatch for this cell, and reports which library views have the defect. Otherwise (if the cell name was not found, even with a mismatching case), nothing is displayed. Cell-availability is checked in a separate checker (explained in Section 3.2.2) due to error-tagging purposes (grouping similar errors under the same error-tag to be displayed together).

3.2.2 Cell Availability Checker

This checker is very similar to the previous one. It checks the availability of functional cells in all the different library formats. For each cell in each library format, the checker first checks the availability of the same cell (with the exact same name) in all other library formats. If the cell was not found in a specific format, the checker then checks that the name of the cell does not exist in a mismatching-case (to make sure that the cell really does not exist), and if that last check gave positive results, the cell is reported missing in that specific library format where it does not exist.

3.2.3 Pin Consistency Checker

This checker verifies the pin-interface of all cells in the library. It basically carries out the following checks:

1. Verifying matching number of pins for each cell across all library formats.
2. Verifying matching names (exact names) of each pin in each cell across all library formats.
3. Verifying the consistent modelling of pins across different library formats. This is done through verifying that supply pins, bias pins, and signal pins have the same functionality and direction in all definitions of the same cell in all the different library formats.

Any pin for any cell that is found missing in any of the library formats is reported along with the name of its respective cell, the library format that has this discrepancy, the version of the library and the library package name. Any modelling inconsistency found (for example a pin defined as a supply pin in one library format, but defined as a signal pin in another) is reported as well.

3.2.4 Layout Consistency Checker

Each cell has a number of pins, each of which has a number of layers describing its layout, along with the geometries (coordinates) of these layers. LEF & GDSII are the only two library formats that have this piece of information. This checker verifies matching layouts of cells between the two library formats: LEF & GDSII. This checker is the most important part of our tool, and consumed much time and effort during the

development of the tool. It also contains a lot of details, and that's why it will be further discussed separately in Chapter 4.

3.3 Preparations for the tool

In order to be able to build the checkers discussed in Section 3.2, we needed to capture important information such as cell names, names of the pins for each cell and the layout information of each of those pins for each cell in the library. These pieces of information are the basic building blocks upon which the checkers mainly rely to verify the consistency of different library formats. For this purpose, multiple *parsers* were built to help capturing this important information. Some library formats (GDSII and Redhawk) required even some pre-processing in order to convert them into readable formats. All of these details are discussed in the following section.
It is worth mentioning here that the parsers and the checkers and all intermediate file manipulation were built in Tcl, the reason for which was to extend a part of the library checker that has been built already in-house at NXP, and also Tcl's ability to store arrays as hash tables, which was really useful in storing different cells' information which were used by the checkers.

3.3.1 Converting GDS Binary Files into Readable ASCII Files

GDSII files are normally binary files. In order to be able to parse those, we had to find a way to convert those binary files, into some readable format, for which a parser could be easily built. Multiple tools were found that do the job, some of which are mentioned below:

1. GDS Utilities 1.3, from GB Research [11]. But this solution was dropped because the tool is not free (a licence needs to be purchased), and also the solution works only for the *Windows* platform.

2. OwlVision GDSII Viewer [12]. This solution is free, and can be run on all platforms; however it was also dropped for the reason that not all of the GDSII records are supported.

3. Glade (GDS, LEF And DEF Editor) [13]. This one was not used because it was more of a *Graphical User Interface* (GUI) tool that is hard to be used in a *Command Line Interface* (CLI).

4. LayoutEditor [14]. This one was not used because it was more of a *Graphical User Interface* (GUI) tool that is hard to be used in a *Command Line Interface* (CLI).

5. KLayout - High Performance Layout Viewer And Editor [15]. This one was found the best tool because it is a free tool that works on all platforms, and could be easily used in the CLI mode. It also produced the most readable output ASCII code representing the GDSII binary files, which made constructing the parsers for reading this output ASCII code really easy.

And thus, we settled on using klayout to do the conversion step from Binary-GDSII into ASCII-GDSII. In order to be able to do this conversion step from within our tool (without having to invoke klayout's user interface), we extracted (with the help of klayout developers) an executable file from the installation package of klayout that can be used directly from within our Tcl scripts to do the conversion.
We also faced another problem in the representation of filler cells in the output ASCII code resulting from klayout's conversion process. Filler cells are used to maintain the Nwell continuity in the standard cell and to fill the gaps in the rows of the standard cell. In the output GDSII-ASCII from klayout, these cells are represented as arrays of an additional smaller cell that is defined implicitly in klayout, but not defined in LEF. So we needed to get rid of those arrays, and get a single definition for the filler cells, to match that in LEF. And this required some tweaking to the source code of klayout in order to produce a definition similar to that in LEF, with no additional cells, and with no arrays. The tweaking was done (with the help of klayout developers) and we managed to produce a definition for filler cells, that corresponds to the one in LEF, which made the job much easier for the layout-checker in verifying the match in the layout for filler cells, between LEF and GDSII.

3.3.2 Converting Redhawk Binary Files into Readable ASCII Files

Redhawk is also a library format whose files need some manipulation in order to be parsed easily. Redhawk files are also available in binary format, yet luckily enough, there exists a utility named "aplreader" which can convert binary redhawk files into ASCII files that can be easily parsed afterwards.

3.3.3 Building Parsers

As we previously mentioned in this section's introduction, building parsers for the different library formats is a must in order to facilitate the process of capturing the information required by the checkers in order to do their job. For this purpose, different parsers have been built. Liberty and Verilog (and partly the LEF) parsers were already built by people at NXP, and those have been reused in our tool. The contribution of this thesis work is in building the following parsers:

1. LEF parser: this captures the geometries of layers of each pin in the cell, and the names of the different pins in the cell, and stores them in a hash table for each cell in the library.

2. GDSII parser: this one parses the GDSII-ASCII file, which is output from the klayout conversion process. It mainly captures the same information as that captured in the LEF parser, for the checker to check the matching between the geometries of pins of the same cell, between LEF and GDSII formats.

3. CDL parser: it parses the information about the available pins, and their names from the netlist of each cell.

4. Three Parsers for APL Redhawk: these are the *CURRENT*, *CAPACITANCE* & *PWC* parsers; each provides different information about the cell behaviour under

different operating conditions, and each has a different format for the APL file, and that's why we had to build a different parser for each of the three types. The information captured is mainly the names of the pins of each cell in the library (along with the name of the cell), to check the pin-interfacing and the availability of cells and their consistent naming with the rest of the library formats.

3.3.4 Building Layer Mapper for the Layout Consistency Checker

In the GDSII library format, layers (metal, poly-silicon, etc..) are represented as *layer numbers* plus *datatype numbers*, while text layers are represented as *layer numbers* plus *texttype numbers*. But this is not the case for the LEF library format, in which layers are represented by their names directly. And hence some mapping was required, in order to translate the layer numbers in GDSII into layer names as in LEF. The mapping file is provided by the library vendor, and has a standard format. A *Layer Mapper* is built within our tool, which reads in the mapping file provided by the tool vendor, and translates the layer numbers and data/text type from GDSII into layer names (matching those in LEF), so that we can carry out the verification and matching process by the checker.

3.3.5 Miscellaneous Checker options

Some options are also added to the tool for more convenience of usage. One example of these options is the ability to specify certain files to be taken into consideration in the verification process of the library. Normally, the user only specifies the name of the library package(s), and the tool then searches all the versions of the package for all the files of different library formats, in order to be parsed and checked for consistency. But for example, if the library provided by the vendor is missing a certain file that is found at a different location, then these options (built for all library formats that are being checked in the tool) will allow the user to specify the path for that missing file, and then it will be accumulated with the list of files already in the library package, and checked for consistency as well with the rest of the library formats. These options are as follows:

1. <-lef "lef_files">: to specify extra LEF files.
2. <-gds "gds_txt_files">: to specify extra GDSII files.
3. <-liberty "lib_files">: to specify extra liberty files.
4. <-verilog "ver_files">: to specify extra verilog files.
5. <-cdl "cdl_files">: to specify extra netlist files.
6. <-rehawk_cap "redhawk_capacitance_files">: to specify extra redhawk-capacitance files.
7. <-redhawk_cur "redhawk_current_files">: to specify extra redhawk-current files.
8. <-redhawk_pwc "redhawk_pwc_files">: to specify extra redhawk-pwc files.

Another example of extra options is the one that provides the capability to skip checking certain directories for consistency. This option is provided as: <-*skip_directories "comma-separated-directory-list"*>. This option is mainly implemented in order to overcome some memory problems that we faced while running the checker on some libraries; the checker complained that there is a memory overflow problem, because of the huge size of the libraries. The important thing is that some of the directories in the library package were just duplications of other files, but with some variations in operating conditions or modelling style, and had nothing to do with the interface description (the main target of our library checker tool). An example for these is liberty files, which have multiple duplications, such as: ECSM, CCS and SI. This option is practical in these cases; we can check those duplicate files sequentially (one at a time, depending on their directories) with the rest of the library formats, in order to overcome the memory problems; and consequently, we implemented this option in order to avoid a potential tool-crash, and to provide a work-around for the memory problem, in case this was a limitation in the run-environment.

3.4 Results

As we pointed out previously in this chapter's introduction, the results for running the *Layout Consistency Checker* are presented in the next chapter, along with the algorithm used for implementing this checker.
Here we present some of the results we obtained from running the different types of checkers on the latest *GLOBALFOUNDRIES* 40nm Low-Power (GF40LP) library.

3.4.1 Cell Name Consistency Checker

Following is a snippet of the errors produced by this checker:

ERROR 2-2 Package 'GF40LP_gf40npkhdst/a02p4' Cell 'SEH_TIE0_G_1' in LEF has a name case mismatch with redhawk_CAP ('seh_tie0_g_1') !
ERROR 2-2 Package 'GF40LP_gf40npkhdst/a02p4' Cell 'SEH_TIE0_G_1' in LEF has a name case mismatch with redhawk_PWC ('seh_tie0_g_1') !
ERROR 2-2 Package 'GF40LP_gf40npkhdst/a02p4' Cell 'SEH_TIEDIN_G_1' in LEF has a name case mismatch with redhawk_CAP ('seh_tiedin_g_1') !
ERROR 2-2 Package 'GF40LP_gf40npkhdst/a02p4' Cell 'SEH_TIEDIN_G_1' in LEF has a name case mismatch with redhawk_PWC ('seh_tiedin_g_1') !
ERROR 2-2 Package 'GF40LP_gf40npkhdst/a02p4' Cell 'SEH_DCAP16' in LEF has a name case mismatch with redhawk_CAP ('seh_dcap16') !
ERROR 2-2 Package 'GF40LP_gf40npkhdst/a02p4' Cell 'SEH_DCAP16' in LEF has a name case mismatch with redhawk_PWC ('seh_dcap16') !

Here, we tag the error resulting from this checker with the tag "2-2"; this tagging helps to group similar errors in one place, which is already done by the tool, grouping similar errors and displaying them at once following their error tag. The error shows the mismatch between LEF and Redhawk formats in the naming of some cells (mainly

case-mismatch). This error has been causing troubles to designers with design tools complaining about the absence of these cells in redhawk format, because the tools expect the exact same name in all library formats. The error mentions the package (library name) and the version (*GF40LP_gf40npkhdst/a02p4*), and the expected cell name (for example *SEH_DCAP16* in LEF), and the wrong cell name (*seh_dcap16* in redhawk). This helps the designer to easily spot this kind of errors, and report them to the library vendor or fix them himself as a workaround.

3.4.2 Cell Availability Checker

Here is a snippet of the errors produced by this checker:

ERROR 2-1 Package 'GF40LP_gf40npkhdst/a02p4' Cell 'SEH_RAIL3T16' exists in LEF but not in GDS
ERROR 2-1 Package 'GF40LP_gf40npkhdst/a02p4' Cell 'SEH_RAIL3T16' exists in LEF but not in redhawk_CAP
ERROR 2-1 Package 'GF40LP_gf40npkhdst/a02p4' Cell 'SEH_RAIL3T16' exists in LEF but not in redhawk_PWC

As we can see, we tag the error resulting from this checker with the tag "2-1". The error shows the existence of some filler cells only in the LEF library format, and their absence in other formats (GDSII and some redhawk formats where this kind of cells are expected to be available). The error mentions the package (library name) and the version (*GF40LP_gf40npkhdst/a02p4*), the cell that has the error (*SEH_RAIL3T16*), and the formats having the problem (GDS and redhawk in this example).

3.4.3 Pin Consistency Checker

Following is a snippet of the errors produced by this checker:

ERROR 3-1 Library package 'GF40LP_gf40npkhdet/a02p3' Bias pin 'VBP' of cell 'EEH_DCAP1' in LEF is not found in Liberty
ERROR 3-1 Library package 'GF40LP_gf40npkhdet/a02p3' Bias pin 'VBP' of cell 'EEH_DCAP1' in LEF not found in GDS
ERROR 3-1 Library package 'GF40LP_gf40npkhdet/a02p3' Bias pin 'VBN' of cell 'EEH_DCAP1' in LEF is not found in Liberty
ERROR 3-1 Library package 'GF40LP_gf40npkhdet/a02p3' Bias pin 'VBN' of cell 'EEH_DCAP1' in LEF not found in GDS

The error resulting from this checker is tagged with the tag "3-1". The error shows the absence of bias pins in some library formats (GDS and Liberty in the example here). The error mentions the package (library name) and the version (*GF40LP_gf40npkhdet/a02p3*), the name of the cell having the error (*EEH_DCAP1*), and the format having the problem (GDS and Liberty in this example). We should point out here that these are just some examples of the defects that the checker found in one standard cell library. Multiple similar errors were also found in the GF40LP

library and other standard libraries that were tested. The errors reported here are just for illustration.

3.4.4 Run time

Here we give an idea about the run time of the *Library Checker* tool. Running the tool with all of the 4 checkers discussed in Section 3.2 on GF40LP library, we found that the tool takes approximately 8 minutes in order to give the results for this example library, which contains around 1500 standard cells. We think this is an acceptable run-time for this tool.

3.5 Summary

In this chapter, we introduce some of the thoughts we have for implementing the layout consistency checker. Relying on other tools as an intermediate stage in our tool turns out not to be such a good idea, especially if the tool is not free and/or not open source. That's why we decided to implement our own tool without relying on intermediate tools. The chapter presented next the different types of checkers implemented in our *Library Checker tool* with some hints about the implementation of each checker. Afterwards, the preparations for the tool were discussed; from file manipulation to building parsers for parsing the different formats of the standard cell library, and the layer mapper for translating layer numbers in GDSII to layer names matching those defined in LEF. Finally, some results from running our tool on the latest GF40LP library are presented to show how the tool behaves, and how it displays the different errors encountered in standard libraries.

Layout Consistency Checker Algorithm

4

Different layers in LEF are represented in the form of rectangles using the *RECT* construct; and thus, in LEF we have a set of corner points defining the rectangles representing the polygons of each layer. These corner points are parsed by the LEF parser for all the layers representing each pin, and saved in a hash table (for each cell), along with the name of each layer, pin-name, pin-direction and pin-type for each pin of each cell in the standard cell library. An illustrating example of a hash table for one standard cell is shown below:

```
SEH_TIEDIN_G_1 {
    layers {T3 NW M1 M2 CA}
    site GF45_DST
    name SEH_TIEDIN_G_1
    pins {
        X {layers {CA {110.0 360.0 170.0 420.0 110.0 220.0 170.0 280.0}
           M1 {105.0 170.0 175.0 1050.0}} direction input name X}
        VSS {layers {M1 {0.0 -35.0 280.0 35.0} M2 {0.0 -35.0 280.0 35.0}}
             use ground direction inout name VSS}
        VBP {layers {NW {0.0 560.0 280.0 1260.0}} use power direction
           inout name VBP}
        VDD {layers {M1 {0.0 1225.0 280.0 1295.0} M2 {0.0 1225.0 280.0
           1295.0}} use power direction inout name VDD}
        VBN {layers {T3 {0.0 0.0 280.0 560.0}} use ground direction inout
             name VBN}
    }
    class {CORE ANTENNACELL}
}
```

In GDSII, layers are represented by a set of points drawing the shape of the polygon for each layer of each pin in the cell. The GDSII parser reads in these points and saves them (along with the name of each layer) in hash tables (one for each cell). The pieces of information collected by the LEF and GDSII parsers are then used to verify the matching of layouts for standard cells, between LEF and GDSII library formats. In this chapter, we discuss the layout consistency checker (responsible for layout-matching verification) in more details.

The organization of the chapter is as follows: Section 4.1 describes the main task for the layout consistency checker. Different algorithms, which were explored for implementing this consistency checker are presented in Section 4.2. Next, the final algorithm upon which we settled for implementing the checker is presented in Section 4.3. Afterwards, some results from running this checker on the latest GF40LP library are given in Section 4.4. Finally, a summary of the chapter is presented in Section 4.5.

4.1 Problem Formulation

GDSII library format is the actual format used to describe the layout of standard cells, and it is the format used for manufacturing chips. However, GDSII files have a huge size, and they cannot be used during the routing phase, because they are not easily manipulated. That is why the LEF format is used, in order to capture the necessary information of the standard cell's layout, in a small-size file that can be used and easily manipulated by the router.
What we need to do for checking the layers' consistency between LEF and GDSII is to check that all of the polygons of LEF are completely included inside GDSII polygons for each layer of each pin in each standard cell. This is because LEF should be just an abstraction for the layout of the standard cell described in GDSII. At the routing phase, the router uses the LEF library format; and thus if any part of the LEF polygon is located outside of the corresponding GDSII polygon, then the router might use this part as a contact, which will result in an open connection. But if the LEF polygon is completely included inside the GDSII polygon, we can guarantee that this open-connection case will not happen.
On the other hand, if GDSII polygons are larger than LEF polygons (such that LEF polygons are still included inside GDSII polygons), this will not create a problem for the router, because it will still use the LEF library format for routing, and thus a contact at any point in the LEF polygon, will still be a valid (correct) connection because it will be inside the GDSII polygon. And consequently, the criterion for an error that should be produced by the checker is: any point that is *inside LEF and outside GDSII*.
Having matching layouts guarantees that we will not have any problems during the routing phase of the design, and consequently we reduce the risks of bugs, which will cost designers a lot of time to discover, and which might leak through verification processes and into production, resulting in faulty chips that would be rejected by customers, costing design companies a lot of money.
Our quest here was to find a suitable algorithm that will do the matching-procedure in a precise and efficient way, and make sure to report any inconsistency between the two layouts in LEF and GDSII.

4.2 Explored Algorithms

Before reaching the final algorithm that we used for implementation, we considered multiple other algorithms, which proved to have pitfalls that did not make them suitable for usage in our implementation of the consistency checker; however, those algorithms were useful in reaching the final algorithm used for implementation. We discuss these algorithms in this section.

4.2.1 Corner Points' Inclusion

Our first thought was to take each of the corner points defining the LEF rectangles, and make sure that each of them is included inside the corresponding GDSII polygon. This inclusion problem is known in the field of computational geometry as the *Point*

In Polygon (PIP) problem. As defined by wikipedia [16], the point-in-polygon (PIP) problem asks whether a given point in the plane lies inside, outside, or on the boundary of a polygon.
This algorithm is simple and least time consuming from the implementation point of view, since this problem is widely known in the field of computational geometry, having multiple algorithms for solution (like the *Ray Casting* and *Winding Number* algorithms); however, deeper analysis for the correct functionality of this algorithm in solving our layout-matching problem revealed some scenarios that act as counter examples for this proposed algorithm, two of which are shown in Figure 4.1 and Figure 4.2.

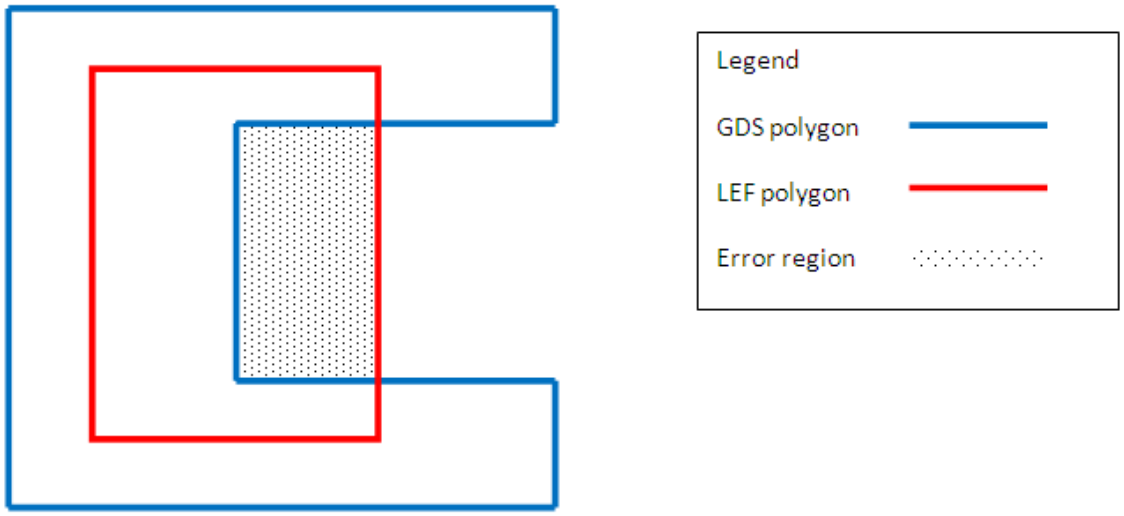


Figure 4.1: Corner Points' Inclusion; Counter Example 1

We notice here from the example in Figure 4.1 that despite having all of the corner points of the LEF rectangle completely included inside the GDSII polygon, the dotted part still represents an error that should be reported by our checker, because this part is not covered inside the GDSII polygon; we can also see from Figure 4.2 that the whole LEF polygon is outside the GDSII polygon, but yet the algorithm will not report this error, because the points on the border of the GDSII polygon are considered inside it, and consequently, this simple *Corner Points' Inclusion* algorithm fails to catch and report the errors for these two counter examples, and thus, we had to think of another algorithm that will solve this problem; which brings us to our next thought.

4.2.2 Polygon Overlap and Area Subtraction

We thought of this algorithm in order to solve the problem we had in the previously mentioned counter examples in Section 4.2.1. So what this algorithm aims at is to subtract the area of the GDSII polygon (for the same layer of the same pin) from that of the LEF polygon, and keep track of the resulting areas (which now represents the parts of the LEF polygon that are not covered by GDSII polygons); we then collect all the subtraction results, and use the results to confirm that the whole LEF polygon

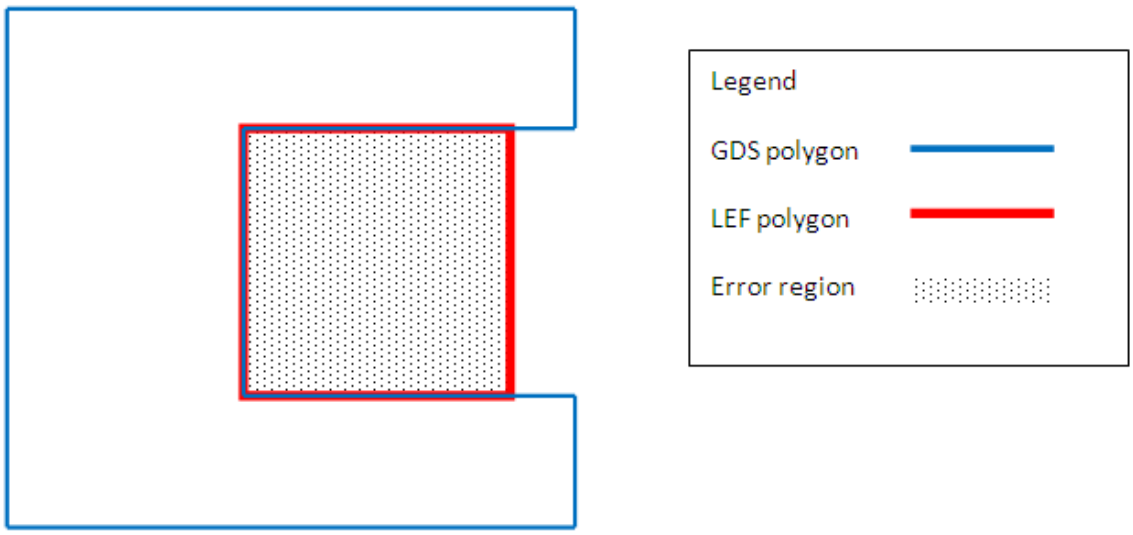


Figure 4.2: Corner Points' Inclusion; Counter Example 2

is contained inside the GDSII polygon, by simply making sure that the subtraction results are always empty, indicating that there are no LEF areas existing outside the GDSII polygons. However, the practical implementations of this algorithm deal only with the case of *Convex Polygons*. A convex polygon (as explained by Wikipedia [17]) is a simple polygon whose interior is a convex set, for which the following properties of a simple polygon apply:

1. Every internal angle is less than or equal to 180 degrees.
2. Every line segment between two vertices remains inside or on the boundary of the polygon.

Figure 4.3 shows a simple example of a convex polygon. Using this algorithm for solving our problem will be too complex, since the overhead of converting all the concave polygons of GDSII into convex ones will represent a great deal of effort that is not aimed at solving the main problem (the layout consistency checking), which will in turn complicate the algorithm to an unacceptable extent. So, we decided to drop this idea as well; which brings us to our third thought described in the next section.

4.2.3 Crossing Points and Centre Inclusion

In this algorithm, we try to find all the crossing points between GDSII and LEF polygons, then depending on the results of these crossings, we can determine whether the LEF polygon is completely included inside the GDSII polygon or not.

We start by finding the crossing points between LEF and GDSII polygons through comparing each side of the LEF polygon, with all of the sides of the GDSII polygon to determine if there exists any sort of intersection , as shown in Figure 4.4. This means that the complexity of this algorithm is of order O($m \times n$), considering 'n' as

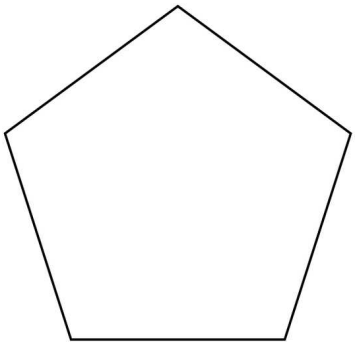


Figure 4.3: An example of a convex polygon: a regular pentagon. Source: Wikipedia [17]

the number of sides of the LEF polygon, and 'm' as the number of sides of the GDSII polygon. Depending on the resulting crossing points, we proceed as follows:

- If no crossings are detected, we check the inclusion of any point (a corner for example) of the LEF polygon inside the GDSII polygon using one of the PIP algorithms; if this point is included inside the GDSII polygon, then the LEF polygon is included entirely inside the GDSII polygon; otherwise, the whole LEF polygon will be outside the GDSII polygon which can be flagged as an error afterwards.
- If there exists some crossings, we find the centre of the resulting collection of points (weather it is a polygon or a line), then we make sure that the centre of this shape is included inside the GDSII polygon.

Figure 4.4: Intersection cases for the Crossing points Algorithm. Completely intersecting (left) and partially intersecting (right) lines.

This algorithm solves the problems we had in the counter examples illustrated previously in Section 4.2.1; however, after deeper analysis, we found another counter example (illustrated in Figure 4.5) that shows the flaw in this algorithm, and its inability to solve our layout-matching problem.

We can see from this example that even though the centre of the LEF polygon (which is also the result of the crossing-points process between LEF and GDSII in this example) is inside the GDSII polygon, still some parts of the LEF polygon are not

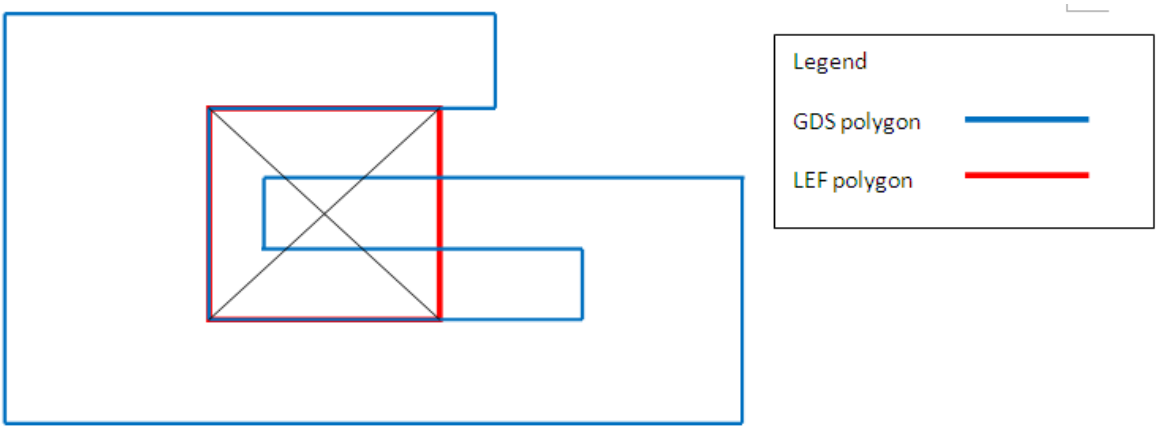


Figure 4.5: Crossing points Algorithm - Counter Example

included inside the GDSII polygon, and these will not be flagged as errors, while they really should! And this brings us to the fourth algorithm, which we actually used for implementing our layout consistency checker. This is described in the following section.

4.3 Implementation Algorithm: Grid Formation and Centre Inclusion

Keeping all the previous counter examples in mind, we reached this final algorithm, *Grid Formation and Centre Inclusion*, which succeeds in solving the problems encountered by the previous algorithms. The algorithm is best explained through an example, as follows in the next section.

4.3.1 Steps of the algorithm

Here we describe the steps we follow to implement the algorithm, illustrated with some graphical figures to make things easier to comprehend:

1. Collect all X-coordinates of LEF and GDSII polygons, as shown in Figure 4.6
2. Collect all Y-coordinates of LEF and GDSII polygons, as illustrated in Figure 4.7
3. Make a grid-like structure from the collected X and Y coordinates from LEF and GDSII polygons.
4. Get the centre of each grid element (rectangle), and check the inclusion of each of those centres inside the GDSII polygon, as shown in Figure 4.8. It is worth mentioning here that in order to save wasted time on processing grid elements that might be outside the LEF boundary, scanning in our algorithm is done only within the LEF coordinates. As we can see from Figure 4.9, the hatched grid elements will be flagged as errors, since the centre of their respective grid elements are not included inside the GDSII polygon, which is exactly what we need; however, we

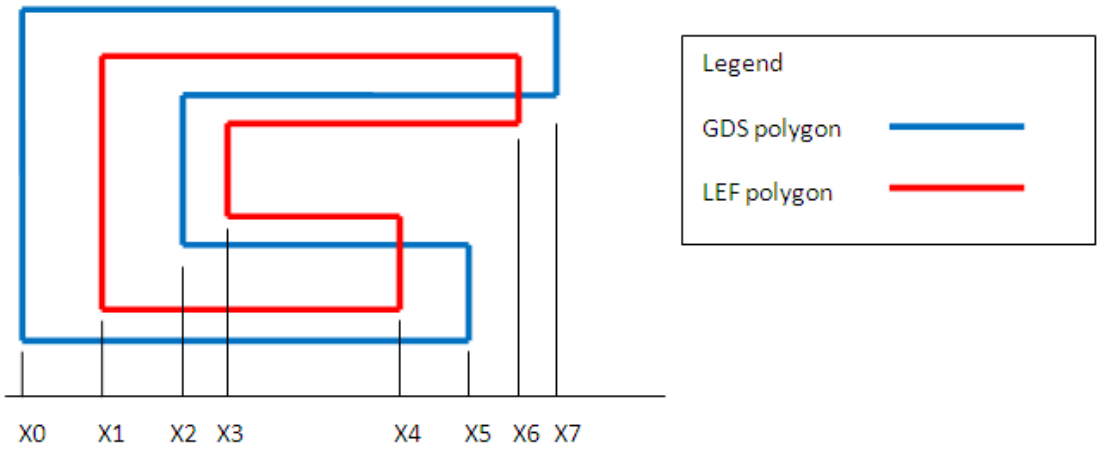


Figure 4.6: Grid Formation and Centre Inclusion Algorithm - Step 1

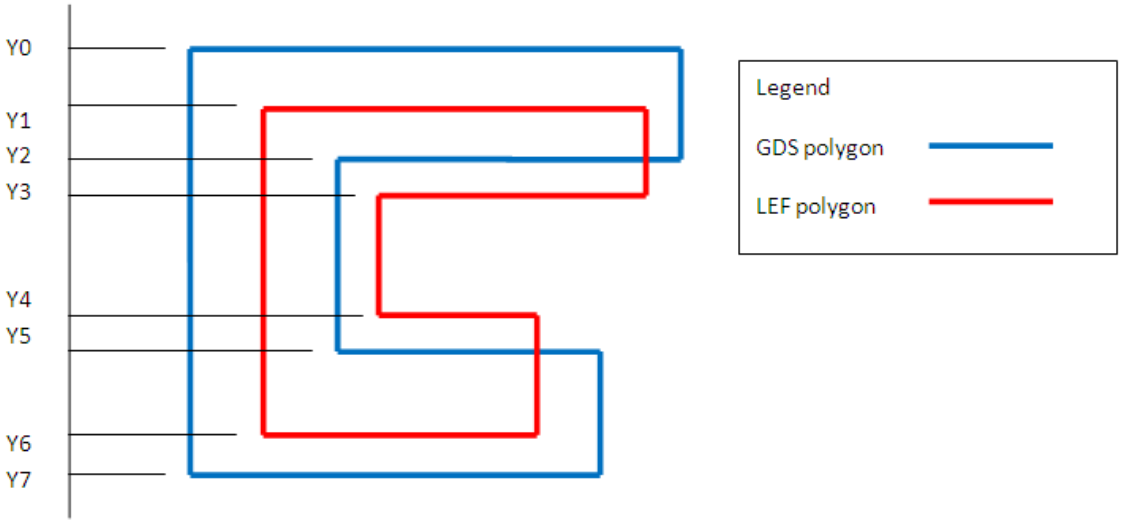


Figure 4.7: Grid Formation and Centre Inclusion Algorithm - Step 2

can see that some rectangles of the grid (the dotted ones in Figure 4.9) might be within the coordinates of LEF, but they are not actually inside the LEF polygon, which might give us incorrect errors; and thus, to solve this problem we decided to check the inclusion of grid elements inside the LEF polygon first, and then check their inclusion inside the GDSII polygon.

5. If all the centres of the grid elements are inside the GDSII polygon, no errors are reported, which implies that LEF and GDSII polygons are matching; i.e. we have matching layouts.

6. If any of the centres are not included inside the GDSII shape, we collect the coordinates of the grid elements whose centres are not inside the GDSII shape, and eventually (after we check all of the grid elements for inclusion) we flag an

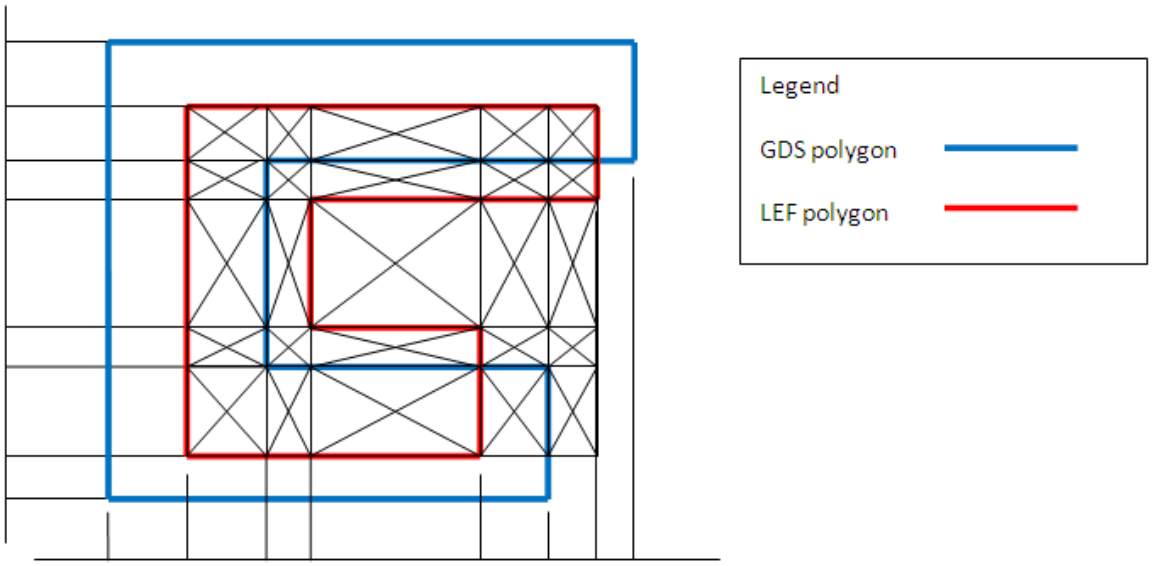


Figure 4.8: Grid Formation and Centre Inclusion Algorithm - Step 4

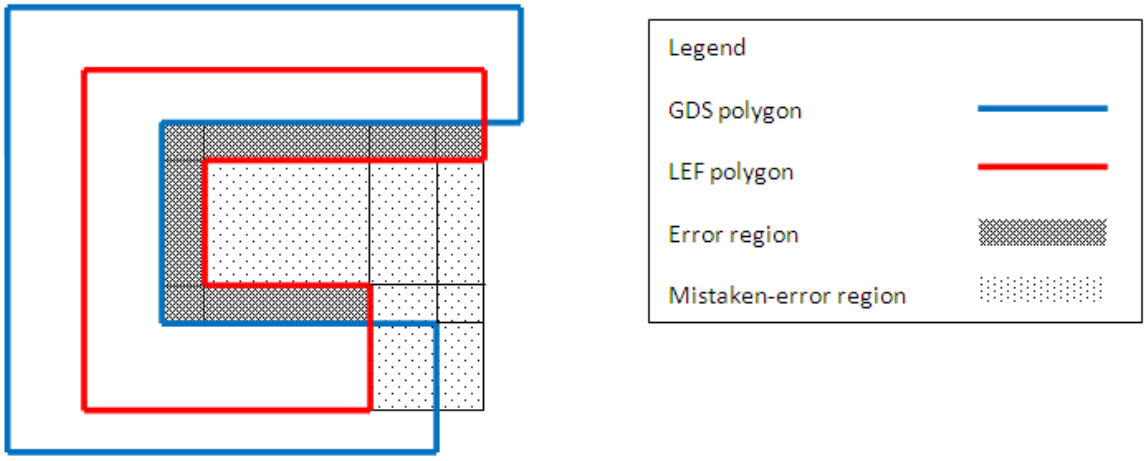


Figure 4.9: Grid Formation and Centre Inclusion Algorithm - Application results

error displaying the coordinates of the grid elements that are causing problems, to be examined more closely by the user.

4.3.2 Ray Casting Algorithm for PIP problem

From the steps of the algorithm described in the previous section, we can easily see that we need to solve the *Point In Polygon (PIP)* problem described in Section 4.2.1 in order to be able to tell if a certain point is included inside the LEF and GDSII polygons or not. In our implementation, we chose the *Ray Casting Algorithm* to solve the PIP

problem.
This algorithm tests the inclusion of a certain point (the centres of the grid-elements in our case) inside a polygon (LEF and GDSII polygons) by counting the number of times that a ray, starting from the point of interest and going in *any* fixed direction, intersects the edges of the polygon under test. An even number of intersections implies that the tested point is outside the polygon of interest, while an odd number of intersections means that the point is inside the polygon. This algorithm is also known as *crossing number algorithm* or *even-odd rule algorithm*. Figure 4.10 illustrates the application of the algorithm on a random polygon.

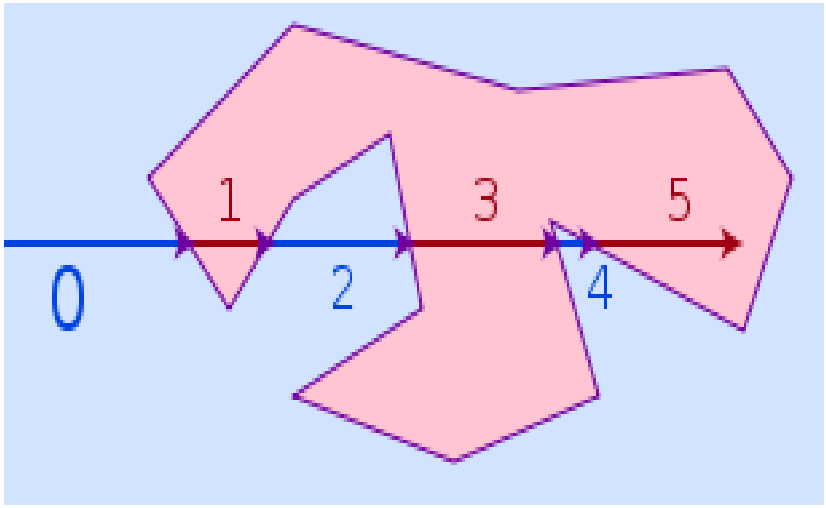


Figure 4.10: Ray Casting Algorithm - Application example. Source: Wikipedia [16]

This algorithm is based on the observation that if a point moves on a ray coming from infinity and reaching to the point of interest, then it will intersect the polygon under test a number of times before it reaches the point of interest. Considering that the travelling point initially comes from infinity, zero number of crossings (intersections with the polygon of interest) would imply that the point of interest is outside the polygon. This point travelling on the ray will then alternately go inside and outside of the polygon until it reaches the point of interest, which implies that the travelling point becomes outside the polygon after every two *border crossings*. And thus, an odd number of crossings means that the point is inside the polygon, while an even number of crossings implies that the point is outside the polygon.

For implementing the algorithm, we first did some searching on the internet to avoid redundant efforts on something that is already implemented, and we found a readily-available implementation to solve this problem. The implementation is provided by *GE Global Research* [18], and it handles the cases where the point of interest is inside, outside or on the boundary of a polygon. *Bob Stein* (who used to be a writer/developer at Galacticomm) presented a great deal of help providing us with this readily available Tcl-implementation of the solution for the PIP problem using the ray-casting algorithm.

4.3.3 Implementation: Sneak peek

In this section, we try to give a simple overview of the way we implemented the grid formation algorithm. The following *pseudo code* outlines the implementation of the algorithm.

```
GridFormationAndCentreInclusion
{
      Gather X & Y coordinates of LEF
      Gather X & Y coordinates of GDSII
      Combine, Sort & Trim(within LEF) X&Y coordinates
      for each {x1 x2} in X
     {
         for each {y1 y2} in  Y
        {
            get centre of the grid element [x1 y1 x2 y2]
            if (centre inside LEF)
                {
               if (centre NOT inside GDSII)
                   {
                   Save coordinates to "mismatched_polygon" and report
                      them as errors
               }
            }
        }
      }
}
```

4.3.4 Testing Examples

We tested the implementation of the algorithm through different scenarios to ensure the reliability of the algorithm and the implementation. Our algorithm showed a good robustness against all the scenarios that we tried in order to discover bugs or failures in the algorithm (or the implementation). We tested the algorithm using all the counter examples presented in Section 4.2, all of which correctly passed the tests, and mismatched parts of the layout between LEF and GDSII were properly reported. Beside these cases, we also tested our algorithm with other examples, as follows:

- LEF polygon is completely included inside GDSII polygon (Figure 4.11). In this case, the checker reported no errors.
- LEF polygon is completely outside the GDSII polygon (Figure 4.12). In this case, the checker reported the complete LEF polygon as a mismatched polygon, as expected.
- A single point match between LEF and GDSII polygons (Figure 4.13). In this case also, the checker reported the complete LEF polygon as a mismatched polygon, as expected.
- LEF polygon and GDSII polygon completely coincide (Figure 4.14). In this case, the checker reported no errors as expected, since the complete LEF polygon is

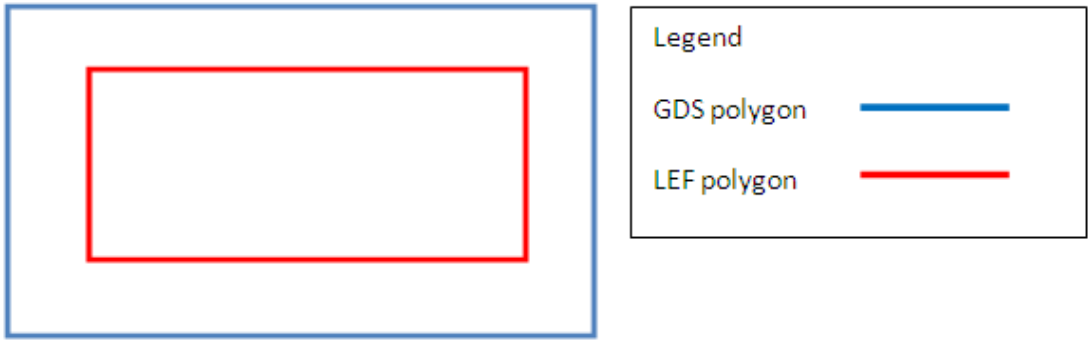


Figure 4.11: Grid Algorithm - Testing Example 1

Figure 4.12: Grid Algorithm - Testing Example 2

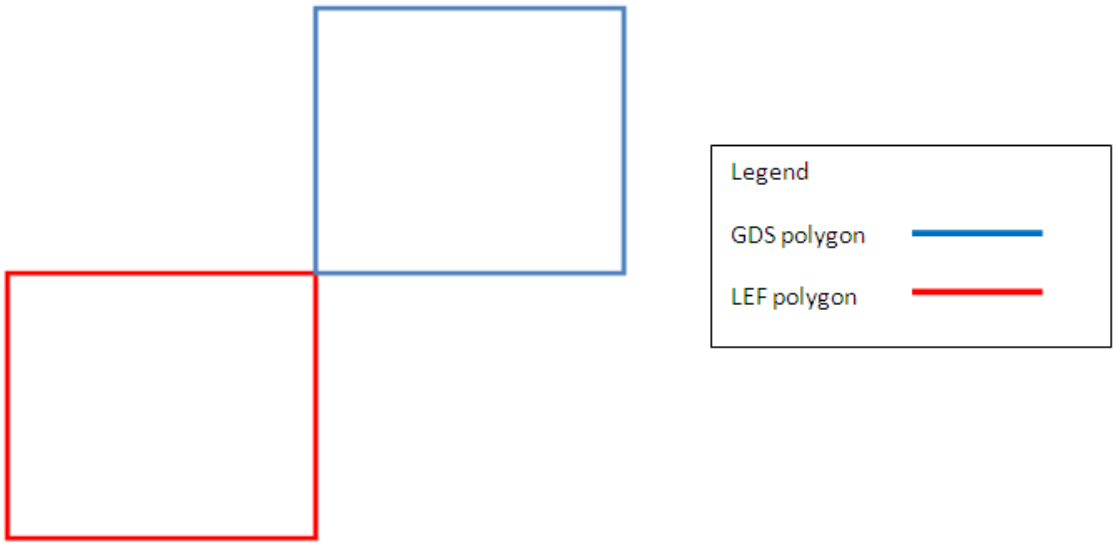


Figure 4.13: Grid Algorithm - Testing Example 3

considered to be included inside the GDSII polygon. No routing problems will occur from this example.

Figure 4.14: Grid Algorithm - Testing Example 4

4.3.5 Algorithm Limitations

The limitation of our algorithm lies mainly in its inability to handle irregular shapes. In other words, our algorithm can only handle rectangular and rectilinear shapes (mainly polygons having boundaries represented only by vertical and horizontal lines), but not shapes having inclined lines. An example of those is shown in Figure 4.15. However, we found this limitation to be acceptable for our application (Library Checker), since standard cells only have vertical and horizontal lines in both LEF and GDSII layouts. Studies also showed that using inclined lines in the layout has a lot of drawbacks that the usage of such inclined lines in layout is discouraged because they produce unpredictable impedances.

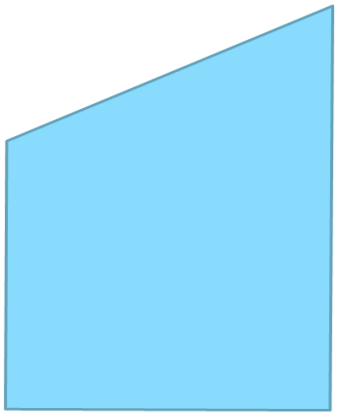


Figure 4.15: Grid Formation and Centre Inclusion Algorithm - Limitations

To increase the robustness and reliability of our checker, we also added a *Checking Condition* within our *Layout Consistency Checker* that would display a warning to the user if any irregular shapes (ones having inclined lines) were encountered, in order to

give the user a hint that these kinds of shapes are not properly handled with the current implementation of the tool.

4.4 Results

Here we present some of the results we obtained from running our layout consistency checker on the latest *GLOBALFOUNDRIES* 40nm low-Power (GF40LP) library. The checker discovered multiple mismatches in the layout of *Supply Pins* for cells. Following is a snippet of the errors displayed for some level shifter cells:

ERROR 5-1 Library package 'GF40LP_gf40npkhpdt/a05p3' Geometrics of layer 'T3' pin 'VSS' of cell 'PEH_LVLDINVE1_Y2_8' in LEF does not match GDS
Mismatched polygon is : 1020.0 400.0 1270.0 400.0 1270.0 560.0 1020.0 560.0
ERROR 5-1 Library package 'GF40LP_gf40npkhpdt/a05p3' Geometrics of layer 'T3' pin 'VSS' of cell 'PEH_LVLDINVE1_PY2_12' in LEF does not match GDS
Mismatched polygon is : 1710.0 1960.0 1960.0 1960.0 1960.0 2120.0 1710.0 2120.0
ERROR 5-1 Library package 'GF40LP_gf40npkhpdt/a05p3' Geometrics of layer 'T3' pin 'VSS' of cell 'PEH_LVLDINVE1_PY2_16' in LEF does not match GDS
Mismatched polygon is : 1020.0 1960.0 1270.0 1960.0 1270.0 2120.0 1020.0 2120.0

Here, we tag the error resulting from this checker with the tag "5-1"; this tagging helps to group similar errors in one place, which is already done by the tool, grouping layout errors and displaying them at once following their error tag. The error mentions the package (library name) and the version (*GF40LP_gf40npkhpdt/a05p3*), the layer having the layout mismatches (*T3* in the examples here), the pin of the cell that is having the layout mismatches (*VSS* in the examples above), the name of the cell having the layout discrepancy (*PEH_LVLDINVE1_Y2_8* as an example here) and finally the coordinates of the mismatched area between LEF and GDSII layouts (the ones tagged with *Mismatched polygon* in the errors above). This helps the designer to easily spot these kinds of errors, and report them to the library vendor or fix them himself as a workaround, to avoid potential hazards in later stages of the design.

The checker also discovered some mismatches in the layout of *Bias Pins*, as shown in the following snippet of the checker's output:

ERROR 5-1 Library package 'GF40LP_gf40npkhpdt/a05p3' Geometrics of layer 'T3' pin 'VBN' of cell 'PEH_CKGTPLT_PWY2_4' in LEF does not match GDS
Mismatched polygon is : 2900.0 1720.0 2910.0 1720.0 2910.0 1820.0 2900.0 1820.0
ERROR 5-1 Library package 'GF40LP_gf40npkhdst/a02p4' Geometrics of layer 'NW' pin 'VBP' of cell 'SEH_MUX2_1' in LEF does not match GDS
Mismatched polygon is : 170.0 530.0 180.0 530.0 180.0 560.0 170.0 560.0
ERROR 5-1 Library package 'GF40LP_gf40npkhdst/a02p4' Geometrics of layer 'T3' pin 'VBN' of cell 'SEH_AN2_12' in LEF does not match GDS
Mismatched polygon is : 170.0 560.0 180.0 560.0 180.0 640.0 170.0 640.0

We should also point out here that these are just some examples of the defects that

the checker found in one standard library. Multiple similar errors were also found in the GF40LP library and other standard libraries that we tested. The errors reported here are just for illustration.

4.5 Summary

In this chapter, we started by formulating the problem we have in the layout consistency checker. Next we introduced some of the thoughts we had for a proper algorithm for implementing the layout consistency checker. Multiple algorithms of the ones explored proved to be unsuitable, because they cannot handle some special cases, as illustrated by counter examples through the chapter. The failure of these algorithms brought us to the final algorithm, the *Grid Formation and Centre Inclusion Algorithm*, which we believe to be the most suitable one for implementing our checker. The chapter also presented more details about the algorithm, including steps and limitations. Finally, some results from running the layout consistency checker on the latest GF40LP library are presented to show how the tool behaves, and how it displays the different errors encountered in standard libraries.

Low-Power Design Techniques Experiment

5

As a final part of this thesis work, we present here some experiments for the actual usage of some of the low-power design techniques, which we discussed in Section 2.1, on a simple incrementer design. The application of these low-power design techniques will be done at an early stage of the design (high abstraction level, as early as system and RTL level); this gives the designer an overview about the expected power consumption in the design, which in turn enables the modification of the design at such early stages to conform with the desired or planned power budget. This early modification saves a lot of time and costs when compared to the case of applying low-power design techniques at lower abstraction levels (circuit level for example). We will also study the power consumption of exactly the same incrementer design without the application of any low-power techniques (called "*baseline design*" throughout the rest of the chapter) in order to be able to assess the effect of usage of the low-power design techniques in reducing the power consumption of the design. For implementation, we will use standard cells from one of the libraries we checked using our *Library Checker* tool. Further details are given throughout the rest of the chapter.

The organization of the chapter is as follows: low-power design techniques used in our experiment are presented in Section 5.1. The incrementer design used in this experiment is described in Section 5.2. Section 5.3 talks about the standard cell library that is used for implementing the design. Test vectors, which are used to test the design and produce the activity figures, are discussed in Section 5.4. Experimental results and power figures for both the baseline and the low-power designs are presented in Section 5.5. Comparisons between the results of the baseline and the low-power designs are given in Section 5.6. Some recommendations for an efficient usage of low-power design techniques are given in Section 5.7. At the end, a summary of the chapter is presented in Section 5.8.

5.1 Power Techniques Used

Two of the most common low-power design techniques are used in our design, namely *Power Shut-Off (PSO)* and *Multiple Supply Voltages (MSV)*. These two techniques have a huge role in reducing power consumption in SoC designs, because they either completely shut-off or reduce the supply voltage of instances, which in turn has a great effect in reducing both the leakage and active power consumption, as discussed in Section 2.1.
As mentioned in the introduction, low-power design techniques will be used in early stages of the design, starting at the RTL level of abstraction. Low-power design intent is described in *Si2 Common Power Format (CPF)* [19]. CPF is a Tcl-based format.

As mentioned by *Si2*'s CPF guide, the CPF file specifies the power characteristics of a design, describing the design and its intended use such that tools may determine what additional elements must be added to support the correct operation of the design in a variety of power conditions. The CPF files for a design complement the RTL and/or netlist description of the design, and can be used throughout different phases of the design process, such as: implementation and verification, as shown in Figure 5.1. This means that CPF acts like a wrapper around the golden RTL description of the design; this wrapper defines the power/voltage islands in the design and the power supply network to the RTL; and thus, CPF helps designers in adding low-power descriptions to an already existing design, without having to modify the RTL description to insert additional elements related to the low-power design methodology used (like isolation or level-shifter cells); this in turn saves a lot of time during the design cycle, since the golden RTL descriptions of already existing designs can be reused directly, without having to be verified again, simply because they were not changed. This is illustrated in Figure 5.2. The additional items that need to be verified are only the correct power sequences (during powering up/down different parts of the design), the correct functionality of the chip with parts of the design turned off and the proper recovery of the state of the design when powering up the switched-off parts.

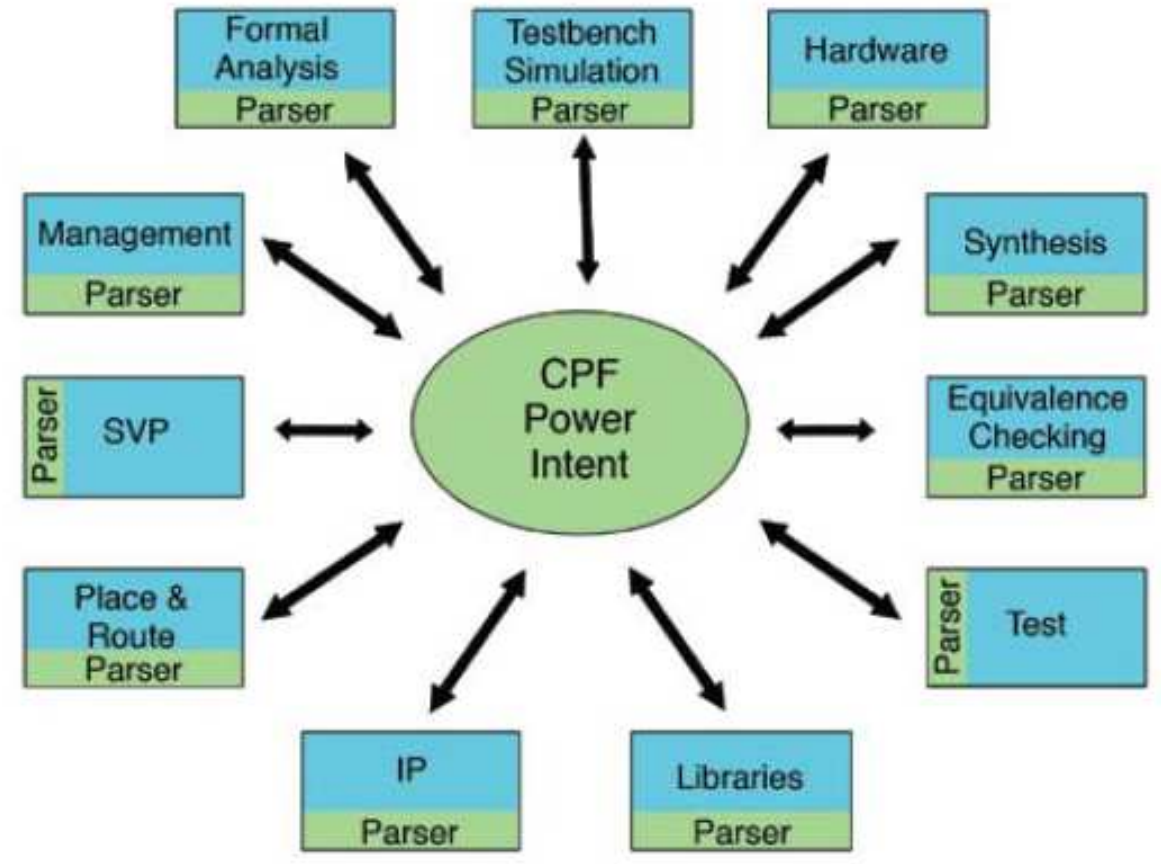


Figure 5.1: CPF-enabled flow: Power is connected in a holistic manner [19]

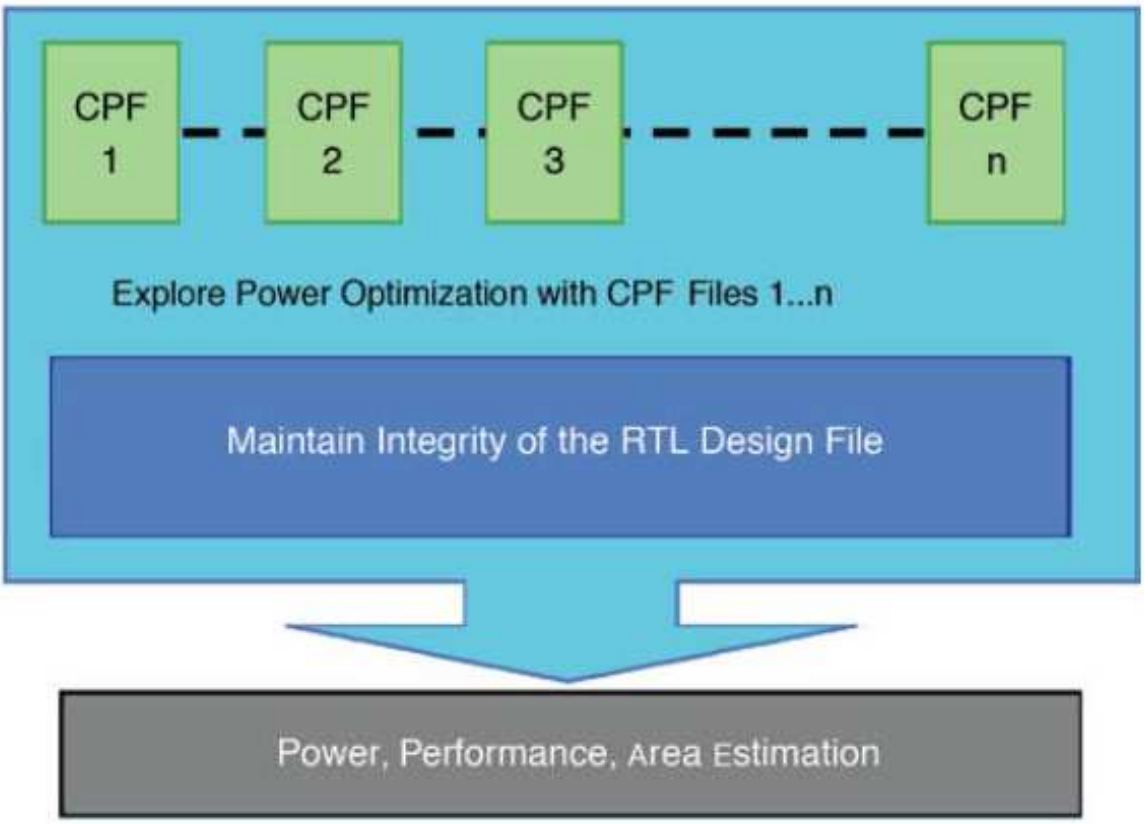


Figure 5.2: Exploring power intent with CPF while preserving RTL [19]

5.2 Design Description

In order to test the application of the aforementioned power techniques in Section 5.1, we use a simple 64-bit incrementer design with load and reset capabilities. The verilog code for this incrementer is as follows:

```
module incrementer(in, acc, clk, load, reset_in, reset_out);
    input [63:0] in;
    input clk, reset_in, load;
    output [63:0] acc;
    reg [63:0] acc;
    output reset_out;
    wire reset;

    assign reset_out = reset_in & ~load;

    always @(posedge clk)
    begin
        if (reset_out)
            acc <= 0;
        else if (load)
            acc <= in;
```

```
            else
                acc <= acc + 1;
19      end
    endmodule
```

The synchronous reset capability of the incrementer allows the user to reset the output of the incrementer "*acc*" whenever the "*reset_in*" signal is active and the "*load*" signal is inactive. The synchronous load capability enables the incrementer to be loaded with a predefined values that are present on the input port "*in*" at periods when the reset signal is not active and the "*load*" signal is active. If both "*reset_in*" and "*load*" signals are inactive, the output of the incrementer is incremented by one step at each rising edge of the clock signal "*clk*".
We then cascade a number of instances of this incrementer to make up a larger incrementer, so that we can simulate the situation of different blocks having different activity figures; and hence we can place each of those instances in a different power domain (PD), each having different power characteristics (such as: supply voltage, shut-off periods, etc.). This will allow us to test the insertion of the different types of low-power design cells (such as: retention, isolation and level-shifter cells), and observe their effect in reducing the power consumption of the design, when compared to the baseline design.

5.2.1 Baseline Design

This design contains only the functional blocks, without any low-power design techniques involved. It basically puts the whole incrementer design (with all the hierarchical instances) in a single power domain with the same power characteristics for all blocks in the design. Figure 5.3 shows the organization of the incrementer instances inside the baseline design. It consists of 3 instances, namely: "*incr1*", "*incr2*" and "*incr3*"; the top-level design containing the three instances is named "*incrementer_cascaded_3*". The whole design lies in a single power domain named "*PD_AON*" (an always-on power domain, coloured in green in the figure) which runs on a voltage of 1.2 volts.

5.2.2 Using PSO low-power technique

In this design, we used only the PSO low-power design technique. The design in this case consists of two power domains (both operating at a voltage of 1.2 volts):

- An always-on (AON) power domain: "PD_AON"
- A switchable power domain: "PD_SW"

Figure 5.4 illustrates the incrementer design using PSO. In the figure, solid lines represent the functional blocks, while dashed lines represent power domains. Table 5.1 explains the organization of the instances within the two available power domains, as illustrated by Figure 5.4. As we can observe, the two instances "*incr1*" and "*incr2*" required the insertion of retention logic due to their placement in a switchable power domain; and hence, retention cells are required to save the state of these two instances

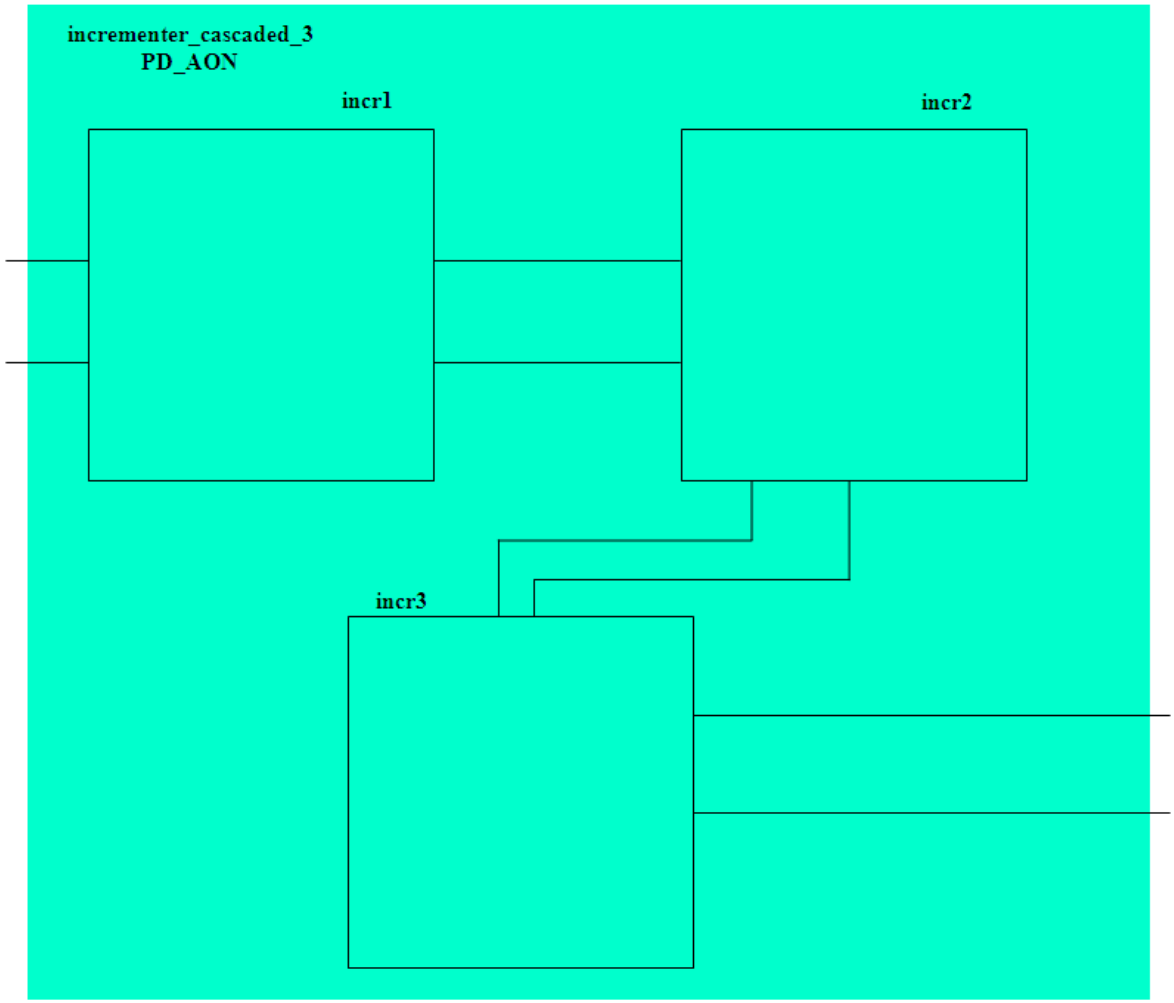


Figure 5.3: Schematic of the Baseline design

during shut-off periods. We also tried the PSO technique without the insertion of retention cells, to see the effect of their insertion on power consumption; of course in the latter case (PSO with no retention logic), a reset is required for the instances in the switchable power domain "*PD_SW*" in order to make sure we get rid of any floating values that might propagate from the switchable domain on power-up.

We can also observe the insertion of isolation logic in the instance "*incr3*" for the obvious reason of isolating the outputs of "*incr2*" during power-down periods, and protecting the inputs of "*incr3*" by keeping them at expected levels using the isolation cells. We can also see that the isolation logic is inserted in the always-on domain (the "*to*" domain), and this is just a restriction imposed by the library vendor.

5.2.3 Using MSV low-power technique

In this design, we used only the MSV low-power design technique. The design in this case consists of two power domains (both are always-on domains):

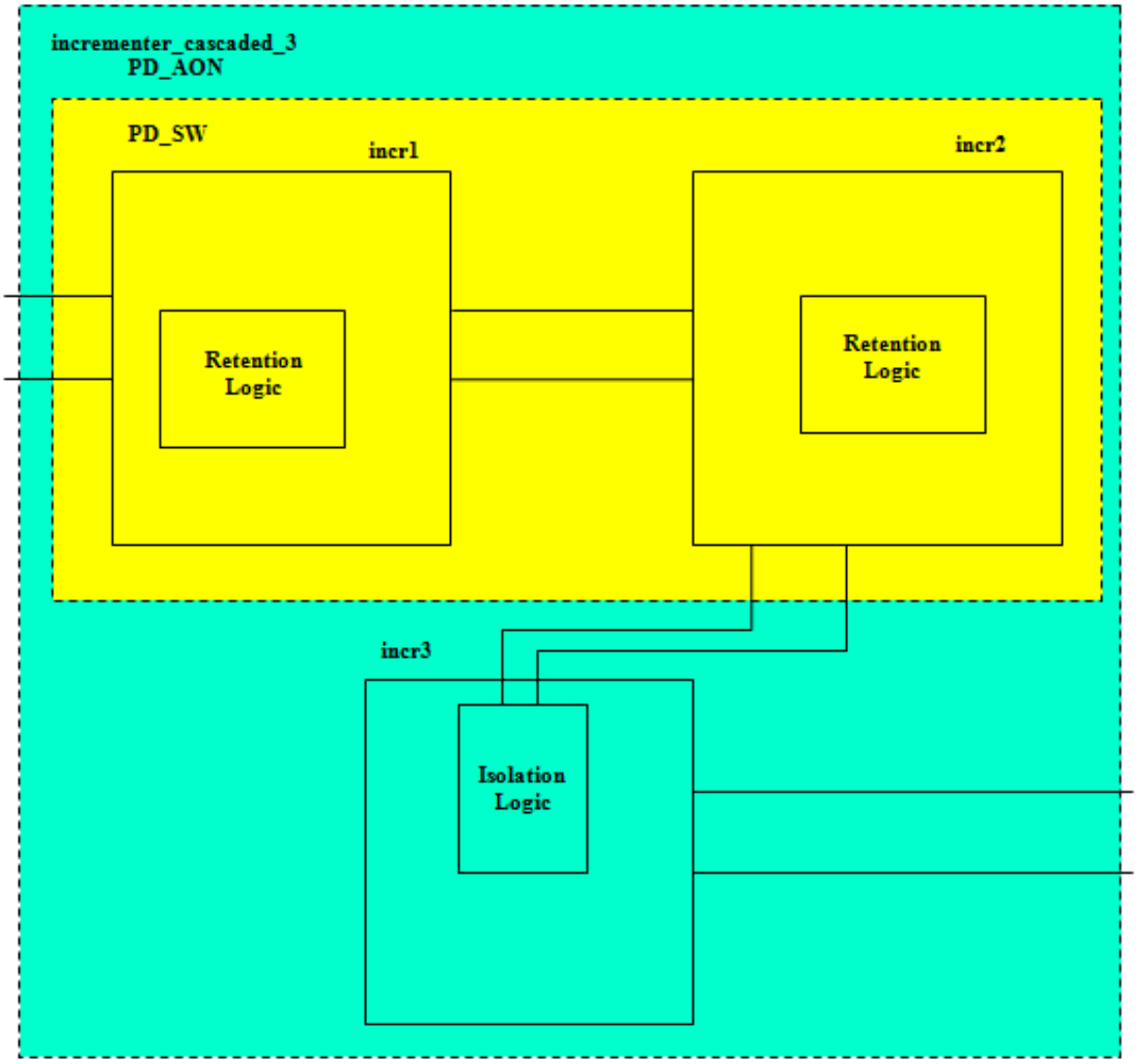


Figure 5.4: Schematic of the incrementer design using PSO

Table 5.1: Organization of PDs and instances in the incrementer design using PSO

Comparison Point	**Power Domains**	
	PD_AON	PD_SW
Instances	incrementer_cascaded_3+incr3	incr1+incr2
Power State	Always-on	Switchable
Schematic Colour	Green	Yellow
Operating Voltage	1.2v	1.2v

- A high voltage power domain: "PD_HIGH_AON", operating at a voltage of 1.2 volts.

- A low voltage power domain: "PD_LOW_AON", operating at a voltage of 1.1 volts.

Note that the usage of the two operating voltages is also imposed by the library vendor, since the characterization of the standard cell library is only done at these two voltages. Figure 5.5 illustrates the incrementer design using MSV. In the figure, solid lines represent the functional blocks, while dashed lines represent power domains. Table 5.2 explains the organization of the instances within the two available power domains, as illustrated by Figure 5.5. As we can observe, the insertion of level-shifters is essential in this case, due to the existence of different voltage islands in the design, and hence we need to preserve the logic value of signals as they travel through different voltage islands, to prevent any undesired change in these logic values of signals due to the different voltage levels between the two power domains. Red level-shifters in the figure are up (low-to-high) level-shifters, while blue ones are down (high-to-low) level-shifter cells.

We can also observe that the level-shifter cells are inserted in the destination (the "*to*") domain, and this is also a restriction imposed by the library vendor.

Table 5.2: Organization of PDs and instances in the incrementer design using MSV

Comparison Point	**Power Domains**	
	PD_LOW_AON	PD_HIGH_AON
Instances	incrementer_cascaded_3+incr1	incr2+incr3
Power State	Always-on	Always-on
Schematic Colour	Green	Yellow
Operating Voltage	1.1v	1.2v

5.2.4 Using PSO and MSV low-power techniques

In this design, we used both PSO and MSV low-power design techniques combined. The design in this case consists of four power domains:

- A low-voltage always-on power domain: "PD_LOW_AON", operating at a voltage of 1.1 volts.
- A low-voltage switchable power domain: "PD_LOW_SW", operating at a voltage of 1.1 volts.
- A high-voltage switchable power domain: "PD_HIGH_SW", operating at a voltage of 1.2 volts.
- A high-voltage always-on power domain: "PD_HIGH_AON", operating at a voltage of 1.2 volts.

Figure 5.6 illustrates the incrementer design using both PSO and MSV. In the figure, solid lines represent the functional blocks, while dashed lines represent power domains. Table 5.3 explains the organization of the instances within the four available power domains, as illustrated by Figure 5.6. As we can observe, in this case we require the

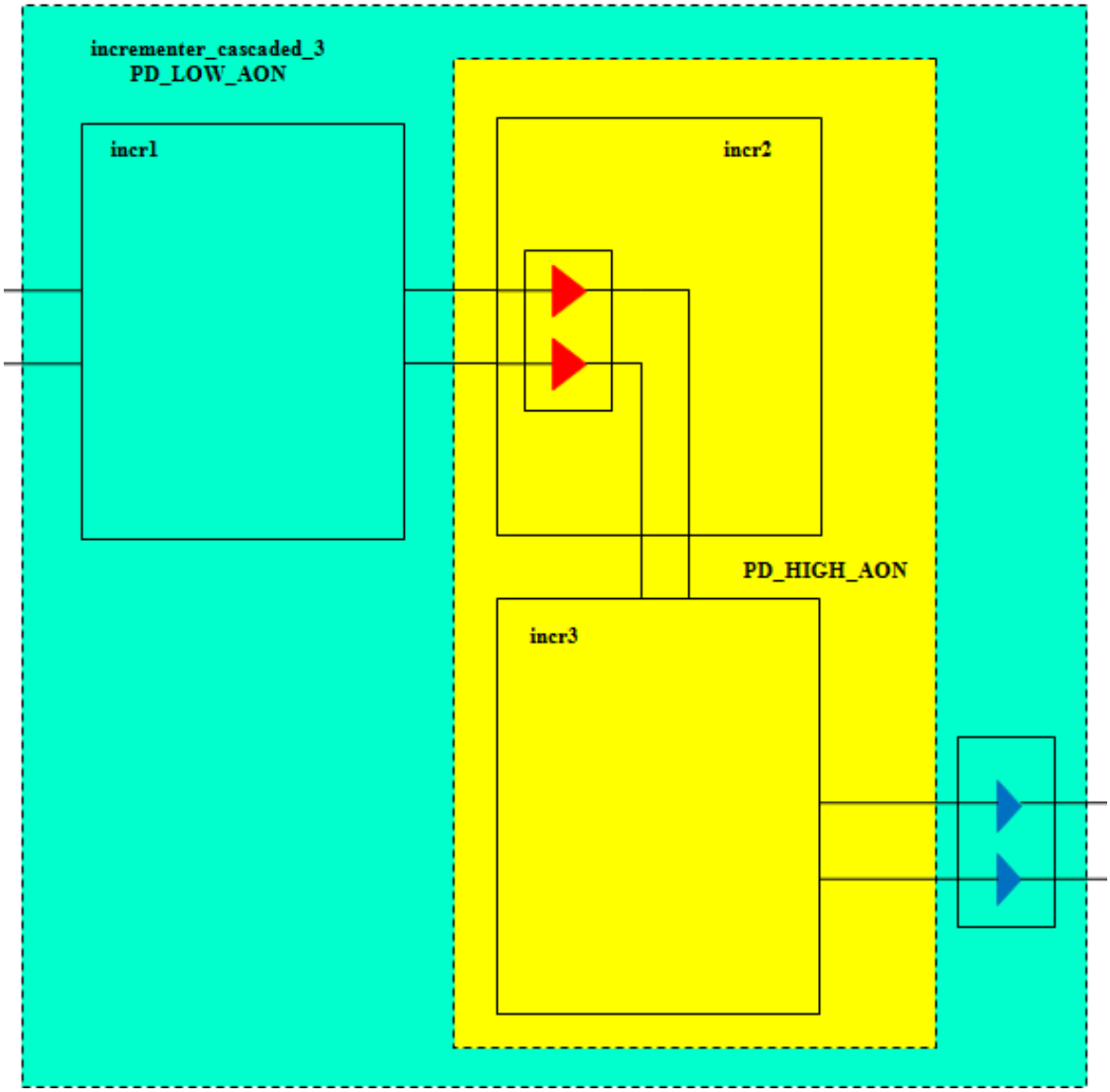


Figure 5.5: Schematic of the incrementer design using MSV

usage of retention, isolation and level-shifter cells; and consequently, this organization has the highest overhead due to the big amount of extra cells that needs to be inserted in order to preserve the correct functionality of the design.

5.3 Libraries used for Implementation

In our experiment, we used one of the libraries we checked for consistency using our *Library Checker* tool. The library package we used is the *GLOBALFOUNDRIES* 40nm low-power library. The library provides the designers with the needed functionality (extra cells) to add power management features to their design. This library provides

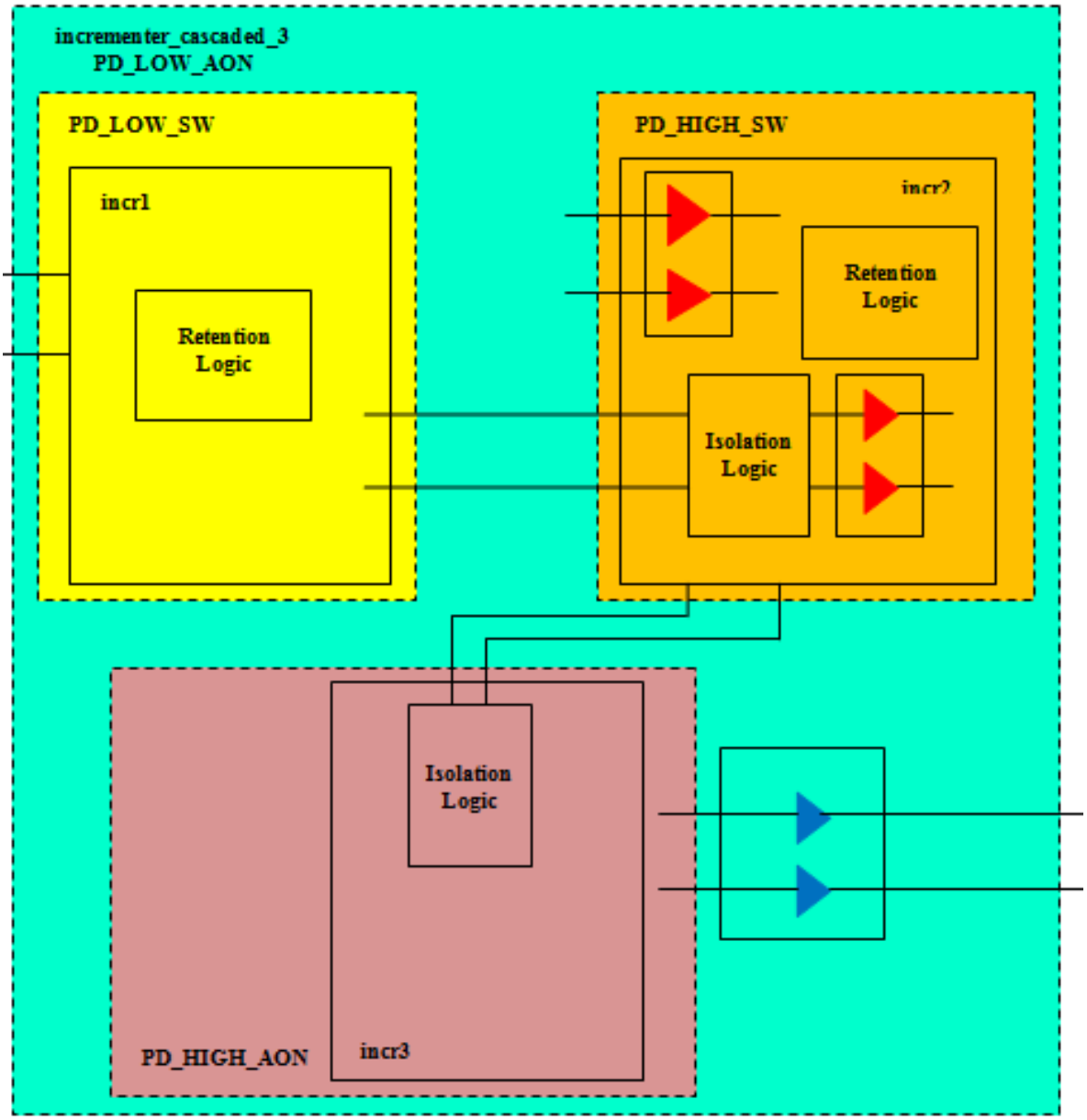


Figure 5.6: Schematic of the incrementer design using PSO and MSV

Table 5.3: Organization of PDs and instances in the incrementer design using PSO and MSV

Comparison Point	**Power Domains**			
	PD_LOW_AON	PD_LOW_SW	PD_HIGH_SW	PD_HIGH_AON
Instances	incrementer_cascaded_3	incr1	incr2	incr3
Power State	Always-on	Switchable	Switchable	Always-on
Schematic Colour	Green	Yellow	Orange	Brown
Operating Voltage	1.1v	1.1v	1.2v	1.2v

new standard cells, and also updated old standard cells, to support the different activities required by low-power design techniques, such as: the creation of different voltage islands, power gating of switchable blocks and state retention of powered-down blocks during shut-off periods. The library contains more than 200 new standard cells serving the low-power design techniques, including:

1. Power Switch Cells: Power gates to supply power to switchable blocks.
2. Isolation Cells.
3. Level Shifters.
    - Up Level Shifters.
    - Down Level Shifters.
4. Always-On Cells.
5. Retention Flops.

It is worth mentioning that studying the characteristics of the library is very important in order to be able to use the low-power design cells included in the library in an easy manner, and avoid difficulties that may arise when inserting these kinds of low-power design cells. An example for these difficulties is a problem that we faced in our design during Gate Level Simulation (GLS); we had a problem with replacing normal registers in the design with retention flops from the low-power library because the tool (mentioned in Section 5.5) did some optimizations for the registers so as to include more functionality, and hence the tool would be able to pick more optimized cells (for area purposes) from the available standard cells in the library. However, when we incorporated the CPF file with the original RTL files, the tool started complaining about its inability to replace registers in the switchable domains with retention flops. After some debugging and tracing, we found out that the retention cells provided by the library are only simple flops, and cannot replace a combined block (of a register and a multiplexer for example) that is produced by the tool's optimization process during synthesis. Turning off the optimization of the tool would be a complete waste, because other parts of the design (other than registers inside switchable domains) still need to be optimized; and thus, we forced the tool to use simple registers (turn off optimization for registers) in those switchable domains, in order to be able to use retention flops afterwards during the insertion of low-power design cells, to properly implement the low-power intent described in CPF. This reveals the importance of studying the library characteristics, and the cautious usage of the tool's optimization capabilities.

It is worth mentioning also that we have been faced with some problems in the modelling of retention cells at the phase of powering-up switchable domains, which also ate a lot of time for debugging the problem, to find out that some primitives inside the retention flops do not restore their values correctly when activating the restore signal; this is illustrated by a simple schematic in Figure 5.7. In this figure, the signal "*int_res_b2*" represents the "restore" signal, while the signal "*int_res_lq*" represents the

restoration value that should appear on the output port "*int_res_iq*" at power up. The schematic clearly shows that activating the restore signal (0 to 1 transition) has no effect in restoring the correct value ('1' in this case) at the output port. We had to modify the description of the retention cell in the library in order to get correct behaviour. Our *Library Checker* tool does not catch this type of modelling problems, and this would be something for verification engineers to catch; which again shows the importance of aggressive verification for low-power designs, as discussed in Section 2.1.1.

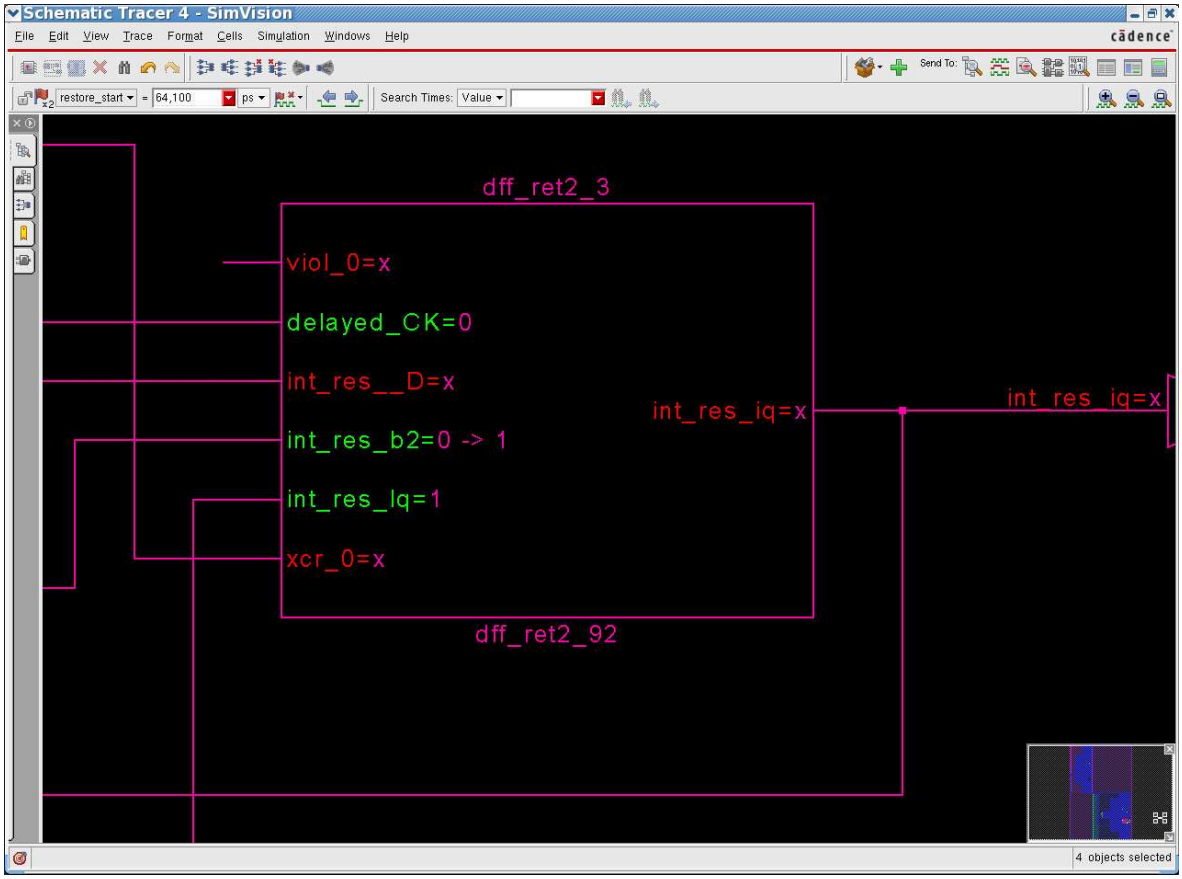


Figure 5.7: Schematic of the problem with retention flops

5.4 Test Vectors

For testing the design, a testbench is constructed; this testbench instantiates the top-level design "*incrementer_cascaded_3*", and drives the test vectors for testing the design. Test vector stimuli have been manually generated in order to test the correct functionality of the design without applying any low-power design techniques (for the case of the baseline design), and also with applying the different low-power design techniques mentioned in Section 5.1, in order to verify the correct power-sequences in case of switchable domains, and guarantee the correct functionality of the design after applying those low-power design techniques. These test vectors are also used to

generate "activity-figures" files, which are used by design tools in order to analyse the power consumption of different design cases.

Test vectors first tests the functionality of the incrementer in the different modes: reset, load and simple increment. Afterwards, control signals (as specified by the CPF file) for different low-power design activities (such as: turning off/on switchable power domains and activating & deactivating isolation and retention cells) are used to drive the correct power sequences (described in Section 2.1.1.5) to test the correct operation of the design when applying the low-power design techniques.
The incrementer has been designed (and tested) with a 100 MHz clock frequency as a constraint; this frequency is useful for low-power applications, such as some embedded processors and wireless sensor networks. The verilog code for the testbench module used for analysing and testing the incrementer design is included in Chapter A in the appendix.

A snap-shot of the simulation waveform for the baseline (and the similar MSV) design case is shown in Figure 5.8. Also a snap-shot of the simulation waveform for the PSO (and the similar PSO+MSV) design case is shown in Figure 5.9. Shut-off period is coloured red in the waveforms, to indicate the "*corruption*" of data during this period.

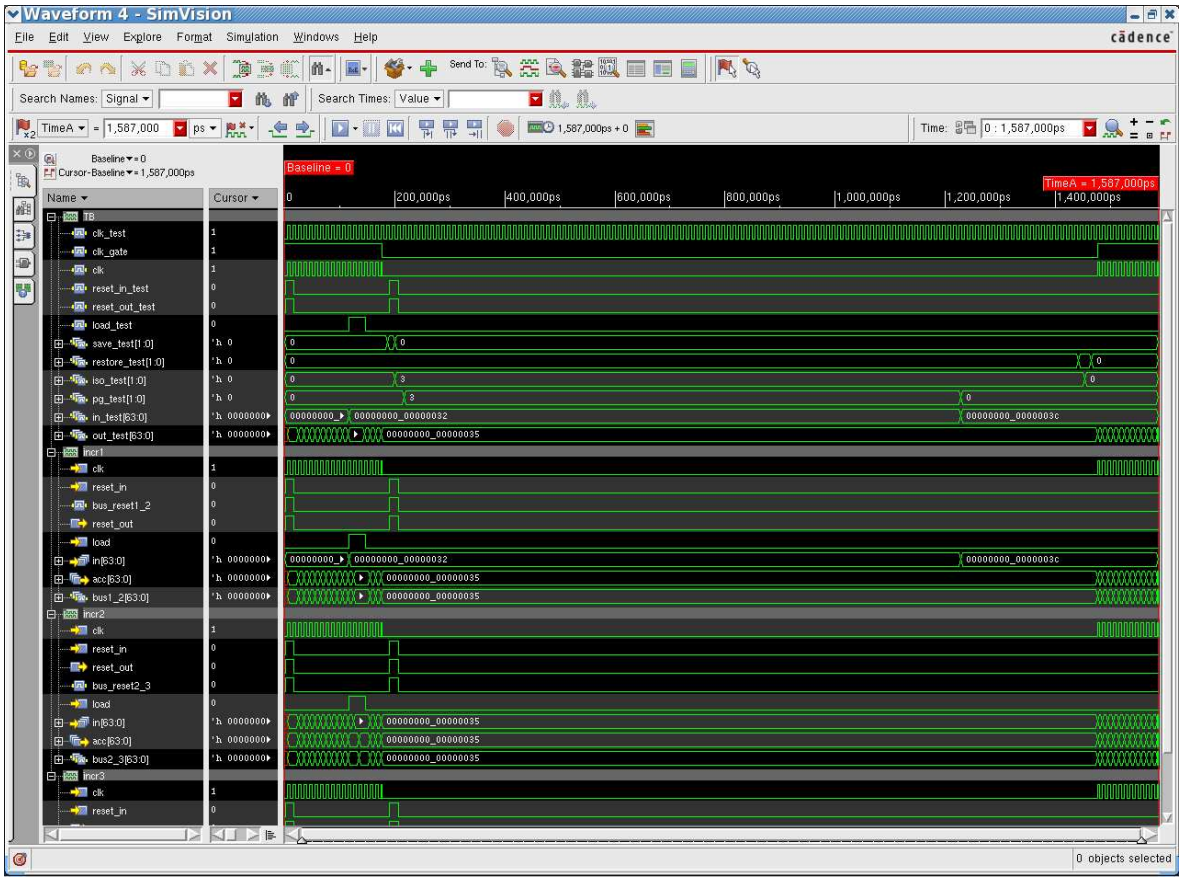


Figure 5.8: Simulation waveform for Baseline design

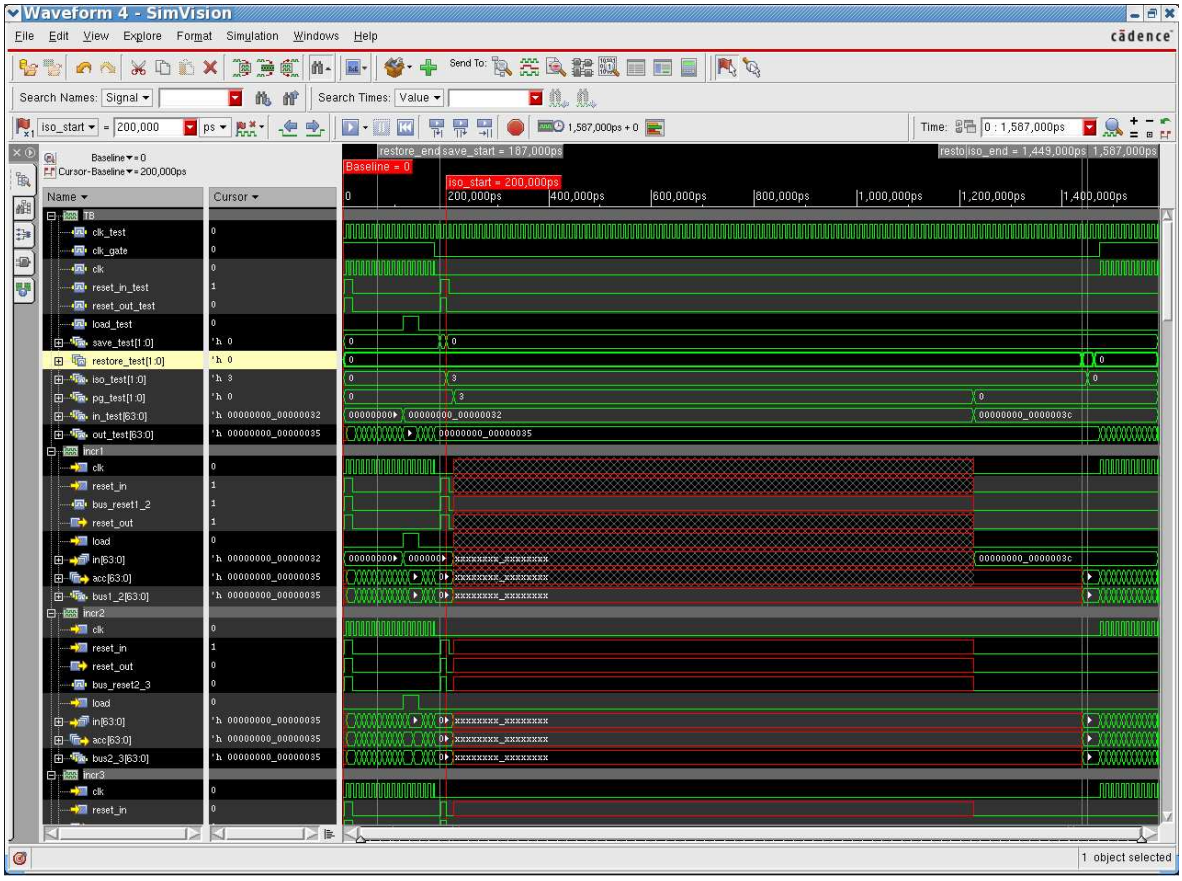


Figure 5.9: Simulation waveform for PSO design

5.5 Experimental Results

In this section, we present the simulation results for running the incrementer designs described in Section 5.2. Results include different metrics, namely: power and energy consumption figures, cell-count, total area, critical path delay and maximum frequency. We also present here the theoretical expected reduction percentage in power consumption, along with the actual reduction percentage, which is obtained from the experimental results; this will be further explained in Section 5.6.

We first start off by presenting the results of the baseline incrementer design. Next we introduce the results when applying only the PSO low-power technique. Afterwards, results with the application of only the MSV low-power technique are introduced. Finally the results of applying both PSO and MSV low-power techniques are presented.

Note that for the case of PSO we have two different cases; the first includes adding retention cells to the design, and the second does not add any retention cells to the design (and consequently will need a reset on power-up). For each of these two PSO designs, we shut-off the power for different periods of time during simulation (specifically for $\sim$ 25% and $\sim$ 65% of the simulation time) in order to have more insight about the effect of turning off the power for different periods of time. And

thus, for the fairness of comparison, we run the design in all cases for two different simulation periods; a short one for 787ns (in which we shut-off the power for $\sim 25\%$ of the time in switchable domains), and a long one for 1587ns (in which we shut-off the power for $\sim 65\%$ of the time in switchable domains).

Cadence Encounter Rtl Compiler [20] is used for RTL design synthesis and power analysis, while Cadence Incisive Design Team Simulator [21] is used for simulation and generation of activity figures. All measurements are done at the gate-level.
Wire-load models are used to estimate the effect of wire length and fanout on the resistance, capacitance, and area of nets. As explained by the Digital Electronics Blog [22]: "A wire load model attempts to predict the capacitance and resistance of nets in the absence of placement and routing information. The estimated net capacitance and resistance are used for delay calculation. Technology library vendors supply statistical wire load models to support estimation of wire loads based on the number of fanout pins on a net". Default wire load model is used during synthesis, and thus RTL Compiler extracts the required information from the technology library.

5.5.1 Baseline Results

Table 5.4 shows the simulation results in the baseline design.

Table 5.4: Baseline Power Results

Power Figure	**Simulation time**	
	787ns	1587ns
Leakage Power (nW)	327.586	327.987
Dynamic Power (μW)	64.896	32.304
Total Power (μW)	65.224	32.632
Total Energy (pJ)	51.33	51.787
Cell Count	739	739
Total Area (μm^2)	1528.153	1528.153
Critical path delay (ns)	8.974	8.974
Maximum frequency (MHz)	111.433	111.433

5.5.2 PSO Results

In this case, we have two simulation cases:

1. Using retention cells; results of which are shown in Table 5.5.
2. Using no retention cells; results of which are shown in Table 5.6.

5.5.3 MSV Results

Table 5.7 shows the simulation results in MSV design.

Table 5.5: PSO (using retention) Power Results

Power Figure	**Simulation time**	
	787ns	1587ns
Leakage Power (nW)	359.802	236.245
Dynamic Power (μW)	59.989	25.852
Total Power (μW)	60.349	26.089
Power Reduction (actual)	7.5%	20%
Power Reduction (theoretical)	16.25%	42.25%
Total Energy (pJ)	47.495	41.403
Cell Count	805	805
Total Area (μm^2)	2163.193	2163.193
Critical path delay (ns)	9.271	9.271
Maximum frequency (MHz)	107.863	107.863

Table 5.6: PSO (using no retention) Power Results

Power Figure	**Simulation time**	
	787ns	1587ns
Leakage Power (nW)	298.413	211.281
Dynamic Power (μW)	62.157	27.674
Total Power (μW)	62.455	27.885
Power Reduction (actual)	4.2%	14.5%
Power Reduction (theoretical)	16.25%	42.25%
Total Energy (pJ)	49.153	44.255
Cell Count	805	805
Total Area (μm^2)	1598.008	1598.008
Critical path delay (ns)	8.974	8.974
Maximum frequency (MHz)	111.433	111.433

Table 5.7: MSV Power Results

Power Figure	**Simulation time**	
	787ns	1587ns
Leakage Power (nW)	364.005	364.329
Dynamic Power (μW)	63.664	31.685
Total Power (μW)	64.028	32.049
Power Reduction (actual)	1.8%	1.8%
Power Reduction (theoretical)	6.6%	6.6%
Total Energy (pJ)	50.39	50.862
Cell Count	902	902
Total Area (μm^2)	1816.920	1816.920
Critical path delay (ns)	9.99	9.99
Maximum frequency (MHz)	100.1	100.1

5.5.4 PSO+MSV Results

Table 5.8 shows the simulation results when using PSO+MSV low-power techniques combined.

Table 5.8: PSO+MSV Power Results

Power Figure	**Simulation time**	
	787ns	1587ns
Leakage Power (nW)	399.320	318.463
Dynamic Power (μW)	64.781	31.706
Total Power (μW)	65.180	32.024
Power Reduction (actual)	0.067%	1.86%
Power Reduction (theoretical)	48.1%	48.6%
Total Energy (pJ)	51.297	50.823
Cell Count	977	977
Total Area (μm^2)	2504.351	2504.351
Critical path delay (ns)	10	10
Maximum frequency (MHz)	100	100

The previously mentioned results are graphically represented as follows:

1. Leakage power consumption in Figure 5.10
2. Dynamic power consumption in Figure 5.11
3. Total power consumption in Figure 5.12
4. Total energy consumption in Figure 5.13

All of the charts (except for the leakage power chart) shows that using PSO with retention saves the most on power and energy consumption; however, this comes at the expense of a huge overhead in area usage (as shown in Figure 5.14, and as will be emphasized in the next section). PSO with no retention saves the most on leakage power consumption, with a much smaller area overhead than the case of PSO with retention cells, and no degradation in speed (as shown in Figure 5.14). This shows that the individual usage of the PSO low-power design technique is the most effective in reducing both components of power consumption, leakage and dynamic.

5.6 Comparison with Baseline

In this section, we will try to verify the effectiveness of low-power design techniques by comparing the power results of designs using those techniques to the power results of the baseline design.

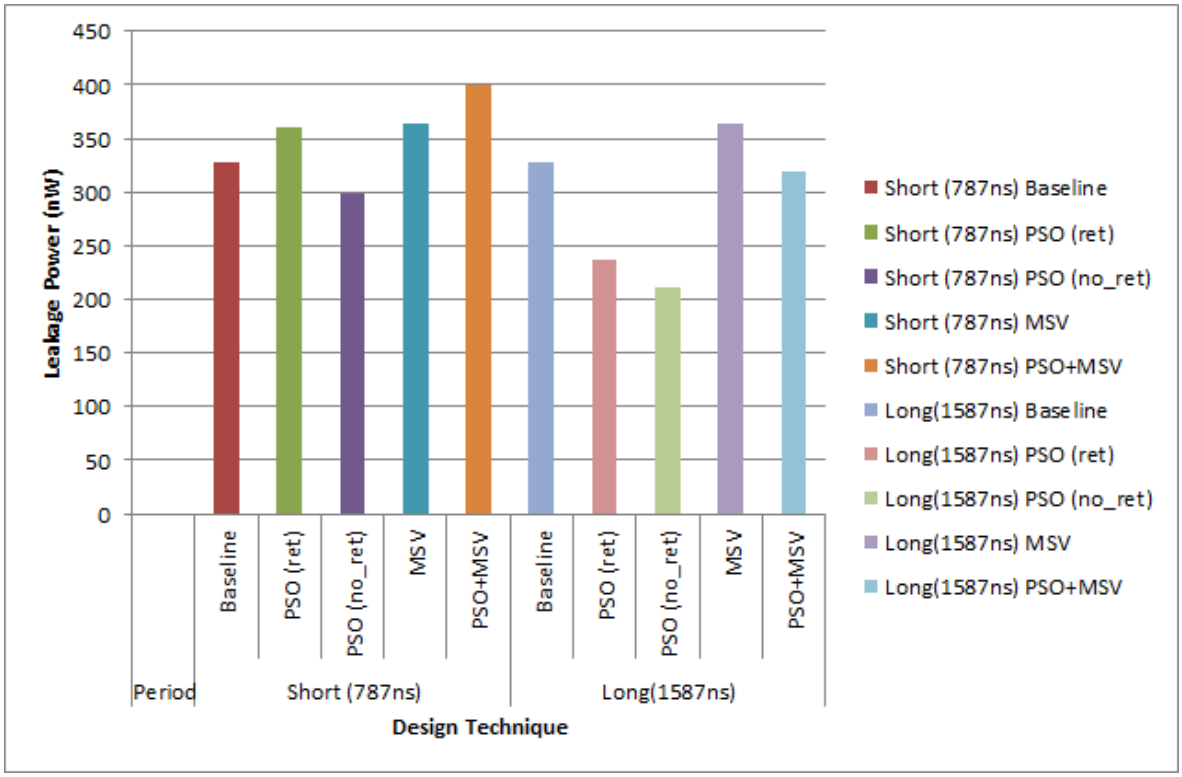


Figure 5.10: Leakage power consumption in different designs

5.6.1 Baseline VS PSO

For the PSO design case, we expect a theoretical reduction in power consumption that amounts to the fraction of time during which the power is shut off (0.25 for the short simulation period, and 0.65 for the long simulation period) multiplied by the fraction of power consumed by the blocks inside the switchable power domains. In our design, we have the two instances "*incr1*" and "*incr2*" inside switchable power domains, and these represents $\sim 65\%$ of the cells of the design (245 cells for the first and 245 for the second); consequently, these two blocks should consume $\sim 65\%$ of the power of the design.
Thus, for the short simulation period, we expect a power reduction of $0.25 \times 0.65 = 16.25\%$; while for the long simulation period, we expect a power reduction of $0.65 \times 0.65 = 42.25\%$.

In this case we have two comparison cases; PSO with retention logic and PSO with no retention logic.

5.6.1.1 With retention logic

For the short-period simulation, we find that the leakage power is not reduced (as expected), but rather increased by $\sim 9\%$ (normalized to baseline); this happened because of the insertion of around 66 extra isolation cells ($\sim 9\%$ of the total cell

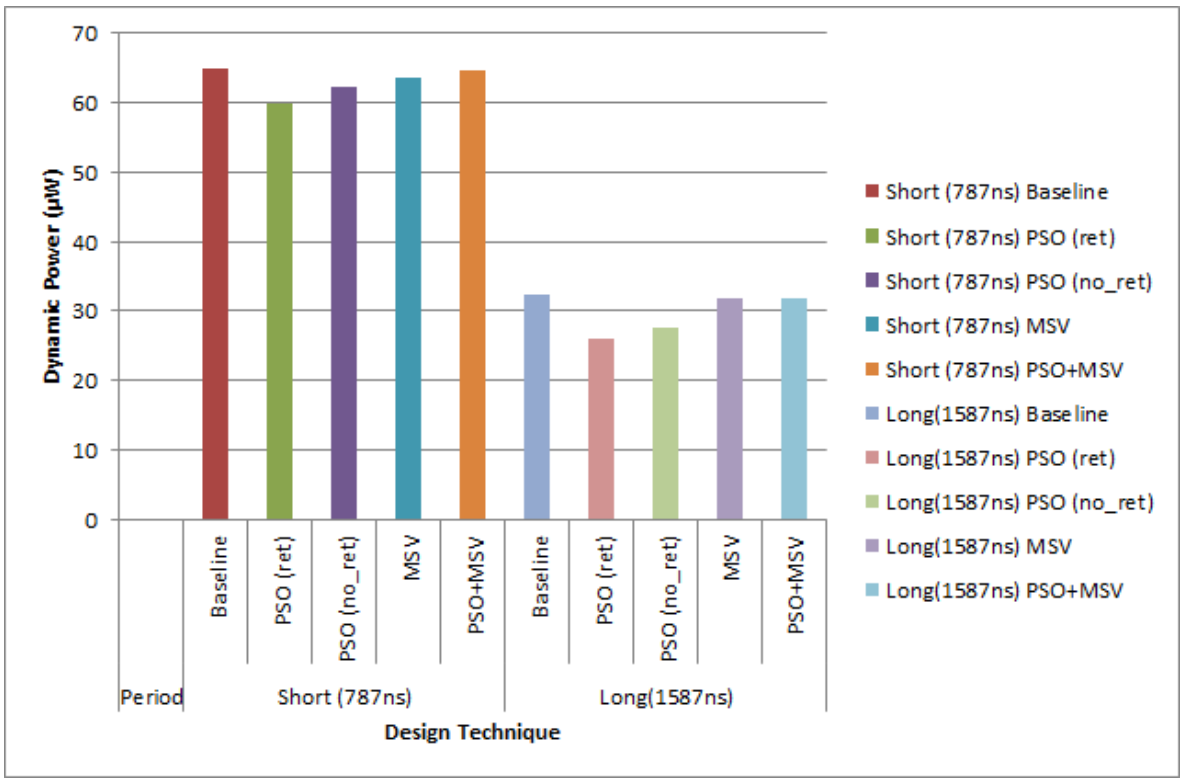


Figure 5.11: Dynamic power consumption in different designs

count of the baseline design), and the usage of retention cells in the design, which leak more than the normal flip-flops (FFs) used in the baseline design. The result of the usage of these two types of cells adds around 32.2nW to the leakage power consumption; however, if we only looked at the combinatorial parts of the design (basically the adders) we will find that the leakage power is reduced from 101.72nW to 86.664nW, that is $\sim$ 15% reduction in leakage power of combinatorial logic, which is approximately our theoretical expectation for power consumption in this design case. As for the dynamic power, we find that it is reduced by $\sim$ 7.5% compared to baseline; obviously the reduction in power consumption is smaller than the expected theoretical reduction, the reason for which is again attributed to the usage of extra isolation cells, and also the usage of retention cells, both of which needs to be always-on (even during shut-off periods), and thus they consume a fraction of the power that should be reduced using the PSO technique. For the energy consumption, we find that the total energy has been reduced by $\sim$ 7.5% compared to baseline.

For the long-period simulation, we find that the leakage power is reduced by $\sim$ 28% (normalized to baseline); this happened because of the long period of power-down, which compensated for and outweighed the overhead in leakage power caused by adding isolation and retention cells. This reduction is again smaller than the expected theoretical power reduction; however, looking again at only the combinatorial power consumption of the incrementer, we find that the leakage power consumption was

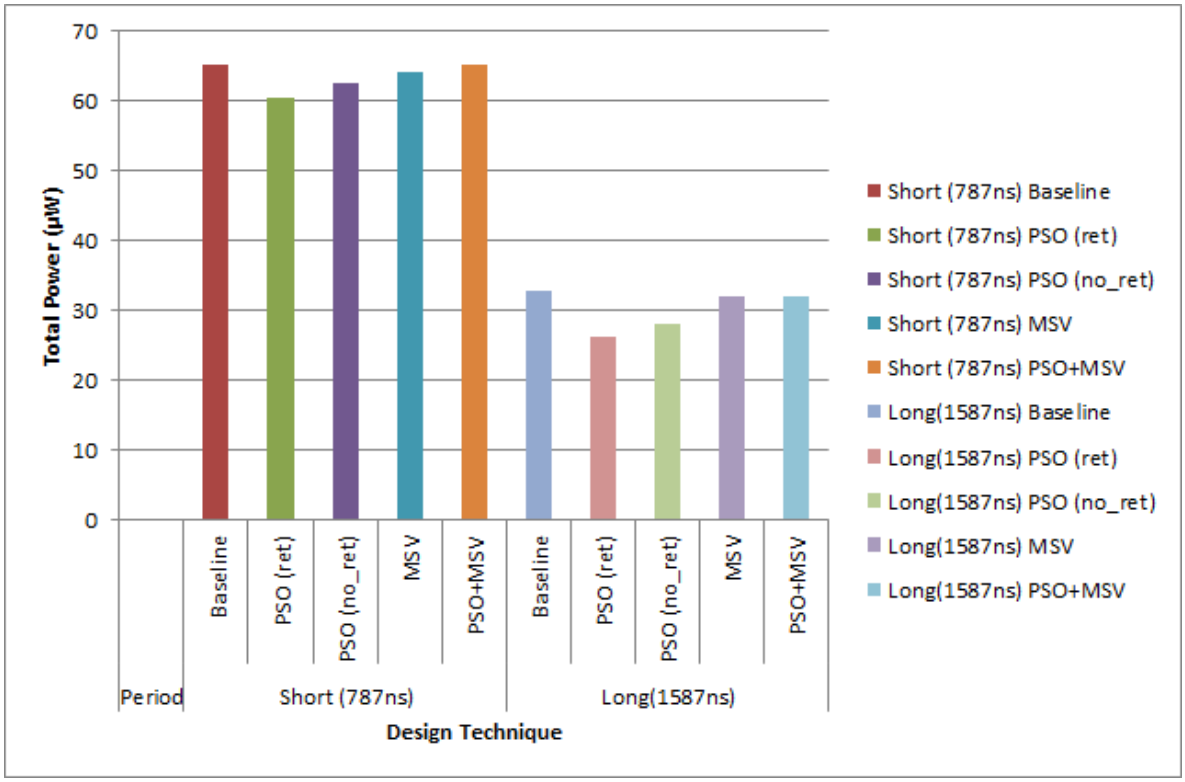


Figure 5.12: Total power consumption in different designs

reduced from 101.717nW to 64.157nW, that is $\sim$ 37% reduction in leakage power of combinatorial logic, which is closer to our theoretical expectation for power reduction in this design case. As for the dynamic power, we find that is has been reduced by around 20% compared to baseline, because of the long shut-off period of switchable domains, which is smaller than the theoretical expectations, again for the same reasons mentioned for the short simulation period. Also the energy consumption has been reduced by 20% over baseline.

This reduction in power consumption comes at the expense of an increase in area of about 41.5%, and a reduction in speed of around 3.5 MHz (3% speed degradation). And this shows that this low-power design technique should only be used with big designs, in which the reduction in power consumption will outweigh the overhead in area usage and speed degradation.

5.6.1.2 Without retention logic

For the short-period simulation, we find that the leakage power is reduced by $\sim$ 9% (normalized to baseline); this is because of the obvious reason of not using retention cells in this design case. Examining the combinatorial parts of the design, we find that the leakage power is reduced from 101.72nW to 86.59nW, that is $\sim$ 15% reduction in leakage power of combinatorial logic, which is approximately our theoretical

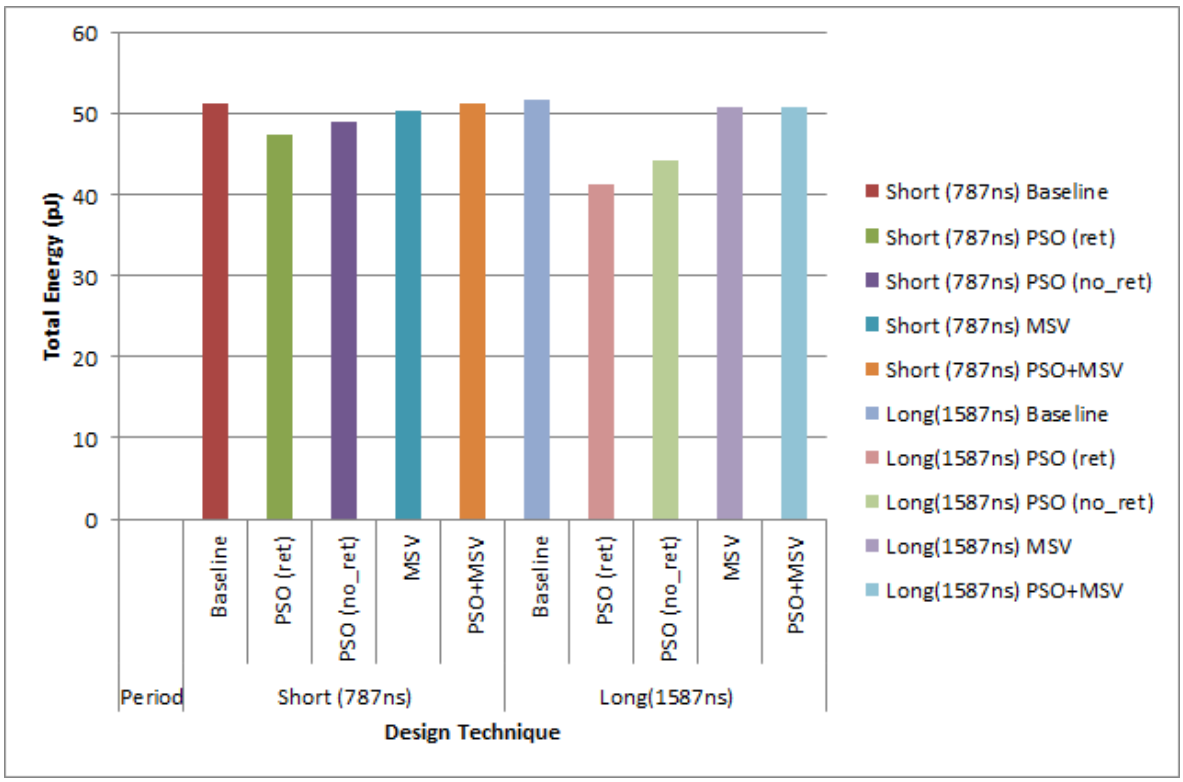
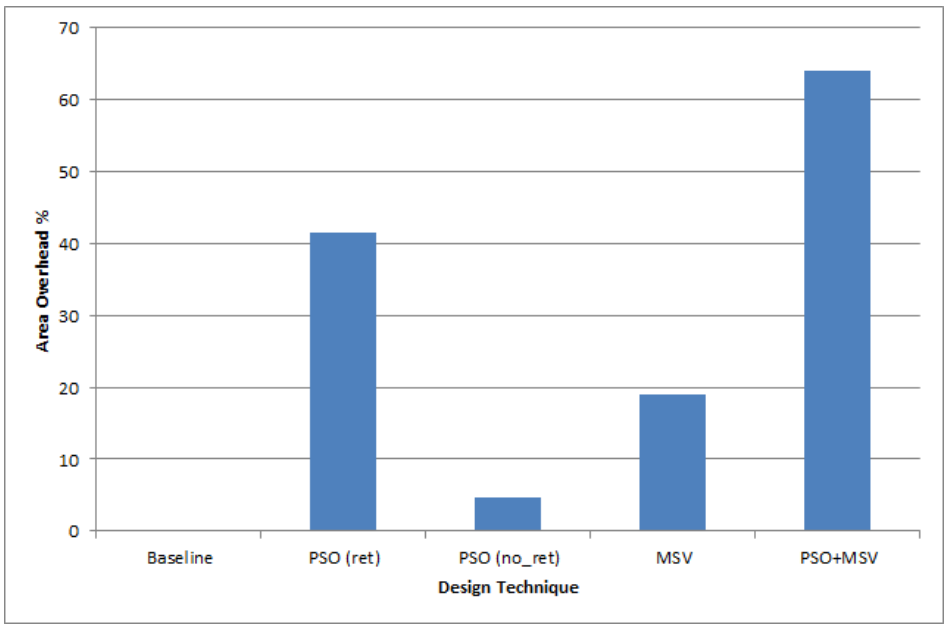
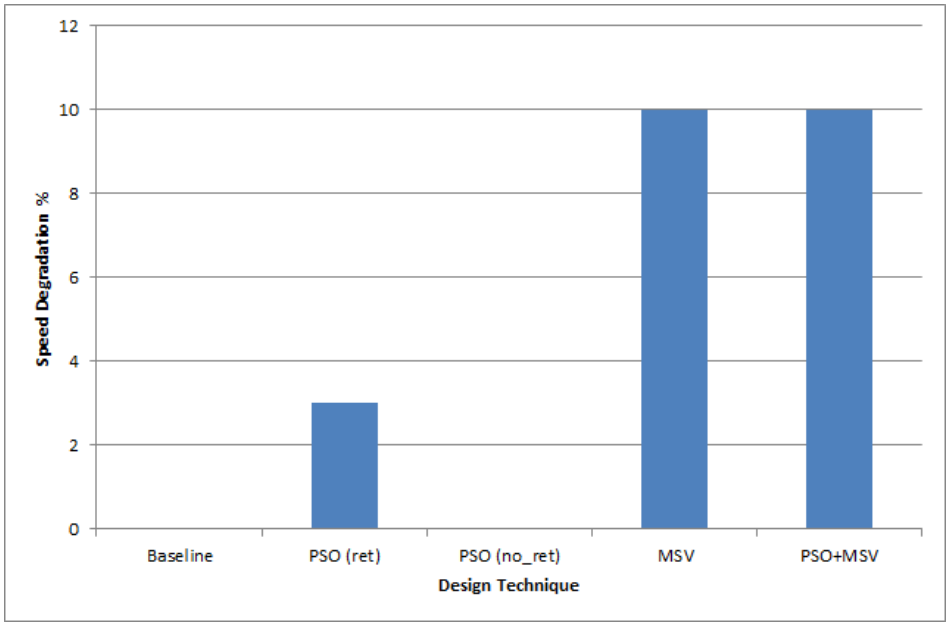


Figure 5.13: Total energy consumption in different designs

expectation for power consumption in this design case.
As for the dynamic power, we find that it is reduced by $\sim 3\%$ compared to baseline, which is odd since we expected a bigger reduction in dynamic power compared to the case of using PSO with retention cells in the design. However, this can be explained by the fact that we had to change the input stimuli for this specific case, because we need a reset signal before deactivating the isolation signal, so that we won't have floating values on the nets after getting out of the power down period. So, in an attempt to see the effect of the reset signal here, we tried simulating the baseline design and the PSO design with retention cells, using the same stimuli as that in PSO without retention cells; we found that the dynamic power consumption in the PSO design with retention cells case became 72.917 μW, and the dynamic power consumption for the baseline became 70.291 μW; which means that we had an increase in dynamic power consumption of around 3.7% over baseline for the case using retention cells, but $\sim 14.8\%$ reduction over baseline in the case when not using retention cells; which explains the fact that dynamic power consumption did not get much reduced when using PSO with no retention cells; that is because of the need for a reset signal at power up.
For the energy consumption, we find that it has been reduced by 2.75% over baseline, for the same reason of the extra reset stimuli; this reduction becomes $\sim 11\%$ when using the same stimuli with the baseline design.



(a) Area overhead in different designs



(b) Speed degradation in different designs

Figure 5.14: Overhead due to the usage of low-power design techniques

For the long-period simulation, we find that the leakage power is reduced by $\sim 36\%$ (normalized to baseline); this happened because of the long period of power-down, which compensated for and outweighed the overhead in leakage power caused by adding isolation cells. It also becomes clear how much leakage power is consumed by retention cells, since the reduction in leakage power for the current case is much more

than the case using retention cells.
As for the dynamic power, we find that is has been reduced by around $\sim$ 14% compared to baseline; the reason for this small reduction (compared to the theoretical expectation) is attributed to the usage of a reset signal at power up and the usage of extra isolation cells. We also found a similar reduction of $\sim$ 14% for energy consumption.

This reduction in power consumption comes at the expense of an increase in area usage of about 4.6%, and no reduction in speed. Which shows that this technique has a lower overhead on both area usage and speed degradation, but it has a smaller reduction of power consumption (due to the need of an extra reset at power up) and it also cannot be used except for the cases where preserving the state of the design during shut-off periods is not necessary.

5.6.2 Baseline VS MSV

For the MSV design case, we try to calculate the theoretical reduction in dynamic power consumption using the equations (2.2) and (2.3) given in Chapter 2. We can see that the dynamic power consumption has a quadratic dependence on the supply voltage, and consequently we get the first order derivative to evaluate the effect of the difference in supply voltage on the dynamic power consumption, which gives us a linear dependence of $[2 \times \Delta V_{dd}]$ (assuming that all other factors contributing to the dynamic power consumption are kept constant), where ΔV_{dd} is the difference in supply voltage. In our design, we have a 100mV reduction in the supply voltage of the instance "*incr1*"; and since this instance represents $\sim$ 33% of the cells of the design (245 cells out of 739 cells), this block should consume $\sim$ 33% of the power of the design. Thus, we expect a dynamic power reduction of $2 \times 100mV \times 33\% = 2 \times 0.1 \times 0.33 = 6.6\%$.
As for the leakage power consumption, we use equation (2.4) in order to predict the power savings. Here we have a mixed linear and exponential dependence on the supply voltage that might roughly be approximated as: $V_{dd} \times I_{SUBTH} \approx V_{dd} \times [1 - e^{\frac{-V_{dd}}{V_t}}]$; differentiating this formula with respect to the supply voltage, we get: $[V_{dd} \times e^{\frac{-V_{dd}}{V_t}}] + [1 - e^{\frac{-V_{dd}}{V_t}}]$. Substituting by $V_{dd} = \Delta V_{dd} = 0.1$, and multiplying by 0.33 which represents the fraction of cell count of the instance inside the low-voltage power domain, we get a leakage power reduction expectation of $\sim$ 6%.

For the short-period simulation, we find that the leakage power is not reduced (as expected), but rather increased by $\sim$ 11% (normalized to baseline); this happened because of the insertion of around 134 extra level-shifter cells ($\sim$ 18% of the total cell count of the baseline design); the result of the usage of this type of cells adds around 58.536nW to the leakage power ($\sim$ 18% of the total baseline leakage power consumption). However, if we only looked at the combinatorial parts of the design (basically the adders) we will find that the leakage power is reduced from 101.72nW to 95.196nW, that is $\sim$ 6.5% reduction in leakage power of combinatorial logic, which meets our expectation of leakage power reduction in the design. As for the dynamic power, we find that it is reduced by $\sim$ 2% compared to baseline (which shows that

PSO low-power technique saved more than MSV). Looking at the dynamic power consumption of the adders, we find that it went from 1.9516 μW in baseline, to 1.8651 μW here in MSV, which means we had $\sim$ 4.5% dynamic power reduction; which is close to our power reduction expectation, but does not match it exactly due to the multiple approximations we did for the expression of I_{SUBTH}. Comparing the total energy reduction for the MSV design, we find that it is around $\sim$ 2% reduction over baseline.
The same results were obtained for the long-period simulation, since we did not change anything that would affect the power consumption in this case.

This reduction in power consumption comes at the expense of an increase in area of about 19%, and a reduction in speed of around 11.3MHz (10% speed degradation). From these results we conclude that the MSV low-power technique is only suitable for application with large designs, because the overhead of insertion of level-shifter cells outweighs the benefit in power reduction that we might get from the reduction of supply voltages, not to mention the area overhead and the performance degradation that the application of this technique causes (as explained earlier).
We can also see that power-savings using the PSO technique is much more than those using MSV (at the expense of bigger area overhead for the PSO case); on the other hand, MSV has an advantage over PSO in that it does not completely shut down some power domains (as in PSO), but only reduces the supply voltage of some domains that does not require high performance, and the whole design remains completely functional the whole time.

5.6.3 Baseline VS PSO+MSV

For this design case, the theoretical reduction in power consumption should be the combined expectations of reduction in the Section 5.6.1 and Section 5.6.2. Thus, for the leakage power consumption, we expect a reduction of $16.25 + 6 = 22.25\%$ for the short simulation period, and a reduction of $42.25 + 6 = 48.25\%$ for the long simulation period.
As for the dynamic power consumption, we expect a reduction of $16.25 + 6.6 = 22.85\%$ for the short simulation period, and a reduction of $42.25 + 6 = 48.85\%$ for the long simulation period.

For the short-period simulation, we find that the leakage power is not reduced (as expected), but rather increased by $\sim$ 22% (normalized to baseline); this happened because of the insertion of all extra logic for low power; retention cells + isolation cells + level shifter cells. However, if we only looked at the combinatorial parts of the design (basically the adders) we will find that the leakage power is reduced from 101.72nW to 81.272nW, that is $\sim$ 20% reduction in leakage power, which is within the range of our expectations for leakage power reduction (exact matching is not achieved again because of the approximations we made in calculating the theoretical leakage power reduction). As for the dynamic power, we find that it is reduced by $\sim 0.18\%$ compared to baseline; which shows that PSO and MSV low-power techniques saved more when applied separately, better than PSO and MSV combined. This mainly happened

due to the insertion of a huge amount of extra standard cells (238 extra standard cell, with a percentage of 32.2% of the total cell count of the baseline design) to preserve the correct functionality of the design when using these low-power design techniques combined. As for the total energy consumption, it has been reduced by $\sim 0.17\%$.

For the long-period simulation, we find that the leakage power is reduced by $\sim 3\%$ (normalized to baseline), and the dynamic power has been reduced by around $\sim 2\%$ compared to baseline. As for the energy consumption, it has been reduced by $\sim 2\%$ over baseline.

The previous analysis shows that PSO and MSV low-power design techniques saved more when applied individually than when applied combined. This case is not the general case, but it only happened in our case because we are examining a really small design; and thus, we should not just use all the power techniques we know about in the design without careful analysis, because as we can see from our incrementer design example, using combined low-power design techniques can sometimes have worse results than those using individual low-power design techniques.

The overhead of the combined application of these two techniques is the area increase of $\sim 64\%$, and the speed reduction of 11.43MHz ($\sim 10\%$ speed degradation).

5.7 Recommendations for the Efficient Usage of Low-Power Techniques

As we can observe from the comparisons in Section 5.6, the usage of low-power design techniques has to be done wisely. Those techniques cannot be used for small designs, because then the overhead power consumption added due to the insertion of low-power design cells (retention, isolation and level-shifters) will outweigh the benefits that we might have in reducing the power consumption of the chip due to the usage of these low-power design techniques, and in some cases, the usage of the these techniques may even increase the power consumption. Consequently, these low-power design techniques are best suited for big designs, in which the added low-power design cells represent a small fraction of the complete design cell-count.
Also attention has to be made to the overhead in area usage, and the performance degradation due to the usage of these low-power design techniques. Careful trade-offs analysis has to be made in order to meet the power budget of the design, without violating the area and performance constraints.

5.8 Summary

In this chapter we tried to exercise the application of different low-power design techniques on an incrementer design. PSO and MSV low-power design techniques were used in this experiment, since they play a very important role in reducing both leakage and active power consumption. Low-power standard cells were used from the GF40LP

library, which is one of the libraries we verified using our *Library Checker* tool. Descriptions and schematics of the different design experiments have been introduced, along with the results of simulation showing different power, area and speed figures for each case. Comparisons between the baseline design and designs using low-power design techniques were presented. Results showed that the usage of low-power design techniques is best suited for big designs; also the insertion of retention logic must be done only when needed. This way, we can make sure that the reduction in power consumption of the chip will outweigh the increase in power consumption caused by the overhead of low-power design cells (such as: retention and level-shifter cells) that have to be inserted into the design.

6 Conclusion

After almost one year of hard and dedicated work, a great deal of progress has been made in this thesis project. Summary and future work related to this project are discussed in this chapter.

6.1 Summary

This thesis work presents a complete checker for the consistency of the different formats of standard cells in standard cell libraries. Different checkers implemented in the tool are specified, along with their function. Preparations needed for building the checkers have been discussed, such as building parsers for the different library formats in order to capture the pieces of information required by the checkers to do their job.
A complete algorithm (which we named *Grid Formation & Centre Inclusion*) for the verification of the layout consistency of standard cells between the LEF and the GDSII library formats has been proposed and discussed (along with its implementation) to avoid any routing problems that might occur in chip design due to inconsistencies in layouts of standard cells between these two library formats.
The checker has been tested using small testcases created to test the implementation statically. The checker has also been used to verify the consistency of different standard cell libraries (such as CMOS 140nm, 90nm and 40nm libraries), and some results from running the checkers on the latest *GLOBALFOUNDRIES* low-power 40nm (GF40LP) standard cell library have been presented in the thesis.

A final chapter of this thesis discusses the effectiveness of the application of common low-power design techniques, using the GF40LP standard cell library that we verified using our *Library Checker* tool, on a simple incrementer design.
Two of the most common low-power design techniques are used in this design experiment, namely: PSO (power shut-off) and MSV (Multiple Supply Voltage) techniques. Simulation results about area usage, speed and power & energy consumption for different design cases are presented. Furthermore, design cases using low-power design techniques are compared to the baseline design (that does not use any low-power design techniques) in order to calculate the actual reduction in power consumption; also comparisons between the actual power reduction obtained from simulation and the expected theoretical power reduction have been conducted.

6.2 Future Work

The algorithm used for layout consistency checking has a limitation that lies mainly in its inability to handle irregular shapes, such as shapes with inclined lines. We had

some ideas about solving this limitation, but time was not really on our side. We believe that fixing this limitation would allow our layout consistency checker to be used to verify the layout of complete designs, instead of being used only for standard cells.
Another improvement that we thought of for our tool is building a simple Graphical-User-Interface (GUI) for using our tool, instead of running it only from the Command Line Interface (CLI).
A further improvement also is to draw the mismatches in layout between LEF and GDSII in a simple graphical form (using Tk), in order to help the user better visualize the errors in the layout, and help the user to fix them easily. All of these improvements are left as a future work for our *Library Checker* tool.

Further work on the experimentation of the low-power design techniques is also needed. Trying more low-power design techniques such as: Dynamic Voltage, Frequency Scaling (DVFS) and Body-biasing and more low-power techniques is needed in order to further reduce the power consumption in SoC designs, and study the trade-offs for area usage and performance.
Also large designs need to be used since it proved from our experiments in this thesis work that those low-power design techniques are not effective for small designs.
High frequencies also need to be tested (around 1 to 2GHz frequency range) with the usage of these low-power design techniques, since these frequencies represent the average operating frequency of processors in mobile and hand-held devices.

Testbench Code

A

```
// Testbench for module "incrementer_cascaded_3"

`timescale 100ps / 1ps

module tb_incrementer_3_PSO_MSV;

reg [1:0] pg_test;
reg [1:0] iso_test;
reg [1:0] save_test;
reg [1:0] restore_test;
reg [63:0] in_test;
reg clk_test, load_test, reset_in_test;
wire [63:0] out_test;
wire reset_out_test;

reg clk_gate,clk;

// Instantiation
incrementer_cascaded_3 incr_test (pg_test, iso_test, save_test,
   restore_test, in_test, out_test, clk, load_test,
reset_in_test, reset_out_test);


`ifdef dump_vcd
    initial
      begin
        $dumpfile("tb_activity.vcd");
        $dumpvars;
      end
`endif

// Generating clock
  always
    #5  clk_test = ~clk_test;

// Clock gating procedure
always @ (clk_gate,clk_test)
begin
    clk = clk_test & clk_gate;
end

initial  // Stimulus
begin
```

```
    pg_test = 0;       // Shutoff condition de-asserted
    iso_test = 0;
    save_test = 0;
    restore_test = 0;
    in_test = 0;
    clk_test = 0;

    clk_gate = 1;
    load_test = 0;
    reset_in_test = 0;

    // Checking reset
    #1 reset_in_test = 1;
    @(posedge clk);   // Waiting for the next rising_edge of the clock (
       since reset is synchronous)
    #2;
    @(posedge clk);
    #2;
    // Checking normal counting
    reset_in_test = 0;
    @(posedge clk);   // Each rising_edge of the clock increments the
       output count by one
    #2;
    @(posedge clk);
    #2;
    @(posedge clk);
    #2;
    @(posedge clk);
    #2;
    @(posedge clk);
    #2;
    @(posedge clk);
    #2;
    @(posedge clk);
    #2;
    @(posedge clk);
    #2;

    // Checking load
    load_test = 1;
    in_test = 64'd50;
    @(posedge clk);   // Waiting for the next rising_edge of the clock (
       since load is synchronous)
    #2;
    @(posedge clk);
    #2;
    @(posedge clk);
    #2;
    // Checking normal counting
```

```
    load_test = 0;
95  @(posedge clk);
    #2;
    @(posedge clk);
    #2
    @(posedge clk);
100 #2;
    // Checking power features
    // Gating the clock
    clk_gate = 0;
    // Saving using retention signals
105 #10 save_test = 2'b11;
    #2;
    // Isolating the powered down outputs
    #10 iso_test = 2'b11;
    // De-asserting the save signals
110    save_test = 0;
    #2;
    // Powering down switchable domains
    #10 pg_test = 2'b11;
    #2;
115 #100; // Waiting for some period, to simulate a period of inactivity in
        switchables domains.
    #100;
    // Powering up switchable domains
    #10 pg_test = 2'b00;
        in_test = 64'd60;
120 #2;
    #100;
    #100;
    // De-asserting retention (Restoring)
    #10 restore_test = 2'b11;
125 #2;
    // De-asserting isolation
    #10 iso_test = 2'b00;
    #2;
    // De-asserting restore
130 #10 restore_test = 2'b0;
    #2;
    // Ungating the clock
    #10 clk_gate = 1;
    @(posedge clk);   // Checking normal counting after powering up
       switchable domains
135 #2;
    @(posedge clk);
    #2;

    // Ending Simulation
140 #100 $finish;
end

endmodule
```

Bibliography

[1] Jan Rabaey, *Low Power Design Essentials*, Springer, 2009, ISBN: 978-0-387-71712-8.

[2] Fred Pollack, "KeyNote MICRO-32," in *32nd Annual International Symposium on Microarchitecture*. IEEE, 16-18 Nov. 1999.

[3] Aveek Sarkar, "An RTL to GDSII approach for low power design: A design for power methodology," *EE Times*, December 2011.

[4] Si2, *Practical Guide to Low-Power Design*, Silicon Integration Initiative, http://www.si2.org/?page=1061, June 2009.

[5] "Standard cell," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Standard_cell, Retrieved 14 August, 2012.

[6] Paulo Francisco Butzen and Renato Perez Ribas, "Leakage Current in Sub-Micrometer CMOS Gates," *Master thesis*, September 2007.

[7] Christian Piguet, *Low-Power Electronics Design*, CRC Press, 2005, ISBN: 0-8493-1941-2.

[8] "Inrush current," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Inrush_current, Retrieved 16 September, 2012.

[9] Cadence, *Virtuoso Abstract Generator*, Cadence Design Systems, Inc., http://www.cadence.com/products/cic/layout_suite, product version 6.1 edition, January 2007.

[10] "Tcl - Tool Command Language," http://tcl.sourceforge.net.

[11] "GDS Utilities 1.3," GB Research, http://www.gbresearch.com/gdsutilities/Default.aspx.

[12] "OwlVision GDSII Viewer," OwlVision, http://www.owlvision.org/.

[13] "Glade (GDS, LEF and DEF editor)," Glade, http://www.peardrop.co.uk/glade/.

[14] "LayoutEditor," LayoutEditor, http://www.layouteditor.net/.

[15] "KLayout - High Performance Layout Viewer And Editor," Klayout, http://www.klayout.de/.

[16] "Point in polygon," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Point_in_polygon, Retrieved 30 July, 2012.

[17] "Convex and concave polygons," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Convex_polygon, Retrieved 30 July, 2012.

[18] "INPOLYG - Determine if an image map click is in a polygon," GE Global Research, http://ge.geglobalresearch.com/.

[19] Silicon Integration Initiative, *Si2 Common Power Format Specification*, Silicon Integration Initiative Organization, 2008, ISBN: 1-882750-34-9.

[20] Cadence, *Encounter RTL Compiler*, Cadence Design Systems, Inc., http://www.cadence.com/products/ld/rtl_compiler, product version 10.1 edition, December 2010.

[21] Cadence, *Incisive Design Team Simulator*, Cadence Design Systems, Inc., http://www.cadence.com/products/ld/design_team_simulator, product version 10.2 edition, November 2010.

[22] "Wire load models," The Digital Electronics Blog, http://digitalelectronics.blogspot.com/2005/02/wire-load-models.html, Retrieved 24 August, 2012.