UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Power Analysis Engine Implementation for VLSI Digital Systems

Diploma Thesis

Iordanis Lilitsis

Supervisor: Christos Sotiriou

November 2023

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Power Analysis Engine Implementation for VLSI Digital Systems

Diploma Thesis

Iordanis Lilitsis

Supervisor: Christos Sotiriou

November 2023

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Υλοποίηση Εργαλείου Ανάλυσης Ισχύος για Ψηφιακά Συστήματα πολύ μεγάλης κλίμακας

Διπλωματική Εργασία

Ιορδάνης Λιλίτσης

Επιβλέπων: Χρήστος Σωτηρίου

Νοέμβριος 2023

Approved by the Examination Committee:

Supervisor **Christos Sotiriou**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Fotios Plessas**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Georgios Stamoulis**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Acknowledgements

To friends and family.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Iordanis Lilitsis

Diploma Thesis

Power Analysis Engine Implementation for VLSI Digital Systems

Iordanis Lilitsis

Abstract

As technological nodes shrink to Sub-Nanometer sizes, the need for precise Power Estimation grows so Power Analysis is becoming increasingly critical in designing and optimizing modern VLSI circuits. With the emergence of complex, portable and battery-powered devices, power consumption is a major design constraint that needs to be carefully managed to ensure optimal device performance, reliability, and longevity. Power analysis is essential to identify power-hungry components and to optimize circuit design to minimize power consumption. In this work, the implementation of a Power Analysis Engine in VLSI digital circuits is presented and compared against industry-standard tool for Power Analysis achieving an average error of -1.54%.

Keywords:

VLSI, Power Analysis, NLDM, NLPM, CCS, Static Probabilities, Digital Circuits

Διπλωματική Εργασία

Υλοποίηση Εργαλείου Ανάλυσης Ισχύος για Ψηφιακά Συστήματα πολύ μεγάλης κλίμακας

Ιορδάνης Λιλίτσης

Περίληψη

Καθώς οι τεχνολογικοί κόμβοι συρρικνώνονται σε Sub-Nanometer μεγέθη, η ανάγκη για ακριβή εκτίμηση ισχύος αυξάνεται, έτσι η Ανάλυση Ισχύος γίνεται όλο και πιο κρίσιμη για το σχεδιασμό και τη βελτιστοποίηση VLSI ψηφιακών κυκλωμάτων. Με την εμφάνιση πολύπλοκων, φορητών συσκευών και συσκευών που τροφοδοτούνται με μπαταρία, η κατανάλωση ενέργειας είναι ένας σημαντικός σχεδιαστικός περιορισμός που πρέπει να αντιμετωπιστεί προσεκτικά για να διασφαλιστεί η βέλτιστη απόδοση, αξιοπιστία και μακροζωία της συσκευής. Η Ανάλυση Ισχύος είναι απαραίτητη για τον εντοπισμό τμημάτων που απαιτούν ενέργεια και για τη βελτιστοποίηση του σχεδιασμού του κυκλώματος για την ελαχιστοποίηση της κατανάλωσης ενέργειας. Σε αυτή την εργασία, παρουσιάζεται η υλοποίηση μιας Μηχανής Ανάλυσης Ισχύος σε ψηφιακά κυκλώματα VLSI και συγκρίνεται με βιομηχανικό εργαλείο για Ανάλυση Ισχύος επιτυγχάνοντας μέσο σφάλμα -1,54%.

Λέξεις-κλειδιά:

VLSI, Ανάλυση Ισχύος, NLDM, NLPM, CCS, Στατικές Πιθανότητες, Ψηφιακά Ηλεκτρονικά Κυκλώματα

Table of contents

List of figures

List of tables

List of Algorithms

Chapter 1

Introduction

Very-Large-Scale Integration (VLSI) circuits have become a fundamental building block of modern electronic systems and have revolutionized the field of electronics. VLSI circuits are made up of millions of transistors, resistors, capacitors, and other electronic components on a single chip. These circuits have allowed us to design and implement complex systems that were once thought impossible, such as powerful computers, smartphones, and communication networks.

VLSI circuits have become a crucial component of our daily lives as they are present in almost all electronic devices around us. From home appliances, cars, and planes to medical equipment, power plants, and satellites, VLSI circuits have impacted almost every aspect of our modern society. They have enabled the development of highly efficient and cost-effective electronic systems that have led to significant improvements in our quality of life.

The development of VLSI circuits has also had a significant impact on various fields, including medicine, transportation, and entertainment. VLSI circuits have enabled the development of sophisticated medical equipment and devices that have transformed healthcare and saved countless lives. They have also revolutionized transportation, allowing for the development of smart cars and traffic control systems that have made transportation safer and more efficient.

Over the years, VLSI circuits have become increasingly complex, with the number of transistors on a single chip growing at an exponential rate. This trend is commonly known as Moore's Law, which states that the number of transistors on a chip will double every 18 months and can be observed in figure 1.1. This trend has been observed for more than four decades and has had a significant impact on the development of modern technology, although

it is no longer applicable in today's electronic industry [2].

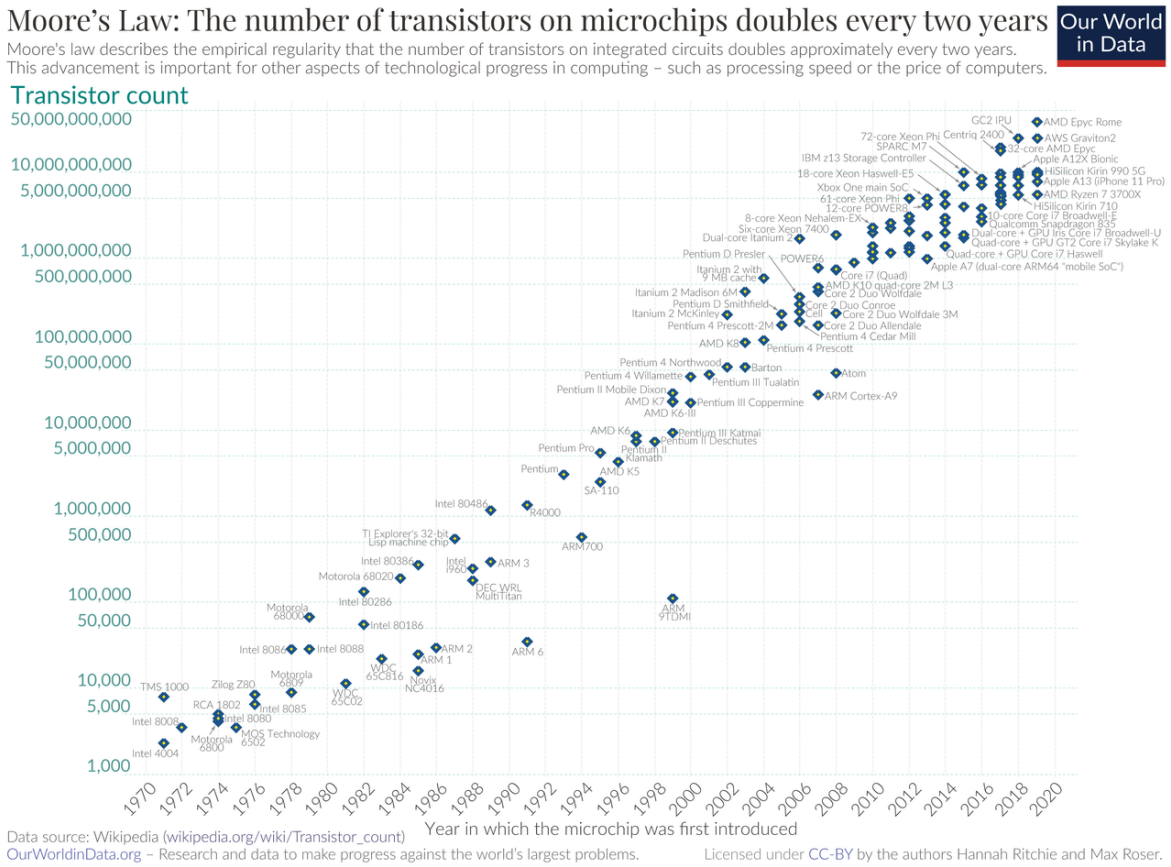

Figure 1.1: Moore's Law, source `https://en.wikipedia.org/wiki/Moore%27s_law`

Moore's Law has been a driving force behind the rapid growth of the semiconductor industry and the development of increasingly powerful and sophisticated electronic devices. As the number of transistors on a chip increases, the performance of the chip also increases, allowing for faster and more powerful electronic devices. This has led to the development of a wide range of devices, from smartphones and tablets to supercomputers and data centers.

In addition to driving innovation in the electronics industry, the complexity of VLSI circuits has also created new challenges for designers and manufacturers. As chips have become more complex, the design and manufacturing processes have become increasingly difficult and expensive. The design of VLSI circuits requires significant expertise and computational power, and manufacturing these circuits requires specialized equipment and processes that are constantly evolving.

Despite these challenges, the growth of VLSI circuits shows no signs of slowing down. As new applications for electronic devices continue to emerge, the demand for more powerful

and sophisticated VLSI circuits will continue to grow. As a result, researchers and industry professionals are constantly working to develop new design and manufacturing techniques to improve the computational and functional capabilities of digital circuits.

In addition to the increased computational and functional capabilities, power consumption is also one of the most increased factors in the use of VLSI circuits. Excessive power consumption can lead to a variety of problems such as overheating, reduced battery life, or even damage to the circuit. As VLSI circuits become more and more complex, power analysis and management have become critical aspects of the design process. Without proper power analysis and management, VLSI circuits can easily consume too much power, leading to negative consequences.

To understand the importance of power analysis, it is essential to consider that VLSI circuits operate by controlling and manipulating electrical signals. These signals are typically represented by voltage levels that can range from 0V to some maximum value, typically referred to as VDD. The power consumed by a circuit is proportional to the voltage level and the current flowing through it, and this power is dissipated as heat. In other words, the higher the voltage level, the more power the circuit will consume, and the more heat it will generate.

Therefore, to ensure the proper functioning of VLSI circuits, it is essential to analyze and manage the power consumption. Power analysis can help identify areas of the circuit that consume the most power and optimize them to minimize power consumption. This optimization can include reducing the voltage levels used in the circuit, minimizing the switching activity, or selecting components that consume less power. By minimizing power consumption, VLSI circuits can operate more efficiently, generate less heat, and potentially have a longer lifespan.

1.1 Aim of this work

The aim of this work is to address the pressing challenges and opportunities within the domain of Power Analysis in Very-Large-Scale Integration (VLSI) circuits. In this work, we intend to provide algorithms for efficiently analyzing a digital circuit's Power dissipation.

Chapter 2

Static Timing Analysis Background

Static Timing Analysis is a crucial step in the design and verification of digital circuits. It is a process that determines the timing behavior of a digital circuit without electrically simulating the circuit over time. The goal of Static Timing Analysis is to ensure that a design meets timing requirements and operates correctly.

The importance of STA has increased with the increasing complexity of digital circuits and their tight timing constraints. In modern digital circuits, timing violations can lead to a range of problems, such as incorrect operation, signal integrity issues, and reduced performance. STA provides an efficient and decently accurate way to analyze the timing behavior of a design and identify potential timing violations.

In order to perform Static Timing Analysis, the circuit technology is analyzed and simulated using various parameters. The results of the electrical simulations are used to generate a timing model (such as NLDM, CCS [3]). An important note is that this model is not design-specific and the same simulation results can be used to analyze any circuit as long as it is in the same specified technology.

Next, the timing model is used to perform timing analysis on the design. The analysis involves calculating the arrival and required times for each signal in the design. The arrival time is the time at which a signal reaches a particular point in the circuit, and the required time is the time by which a signal must arrive at a particular point in the circuit to meet timing requirements. A timing violation has occurred if the arrival time is later (Setup violation) or earlier (Hold violation) than the required time.

To briefly explain the setup and hold violations, figure 2.1 is provided. Setup and Hold times are the main focus of a Static Timing Analysis tool. These times ensure that the data

read by a design flip-flop is not ambiguous. More precisely:

- Setup: the amount of time before the active clock when the data input must stay stable
- Hold: the amount of time that the data input must remain stable following the clock's active edge

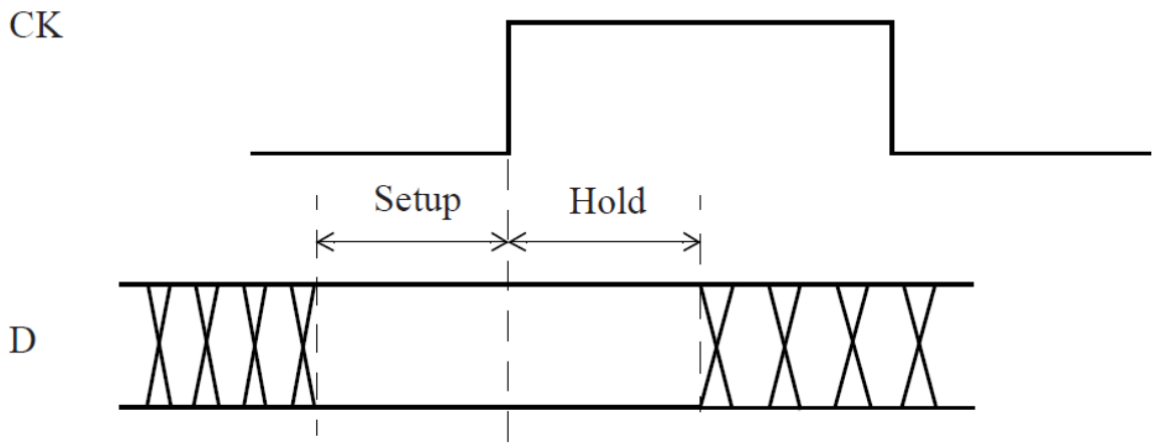

Figure 2.1: Setup and Hold Times

Finally, the Static Timing Analysis results are used to optimize and correct the design. This process involves adjusting the circuit's timing by either changing the circuit's layout or adjusting the circuit's logic to eliminate timing violations.

In summary, Static Timing Analysis is a crucial step in designing and verifying digital circuits. It helps ensure that a design meets timing requirements and operates correctly.

2.1 Gate Delay and Slew Estimation

In the field of Power Analysis, the most important component needed from the Static Timing Analysis perspective is the annotated slew in the inputs of the gate as it is a prerequisite in Internal Power Calculations. In this section, the gate delay and slew estimation will be presented as these two are closely connected.

The delay of a digital gate is determined by the time it takes for a signal to propagate through the gate, which is known as the gate delay. The gate delay is affected by several factors, such as the input slew rate, output capacitance, and the internal resistance and capacitance of the gate itself.

To calculate the delay and output slew of a gate, engineers utilize a delay/slew lookup table (delay LUT, output slew LUT), which provides a mapping between the input slew, output capacitance, and the corresponding gate delay/slew.

To calculate the delay (or slew) of a gate, the STA engine first measures the input slew, which is the rate at which the input signal transitions from low to high or high to low. The input slew affects the gate delay because it determines how quickly the gate input capacitance charges or discharges. A faster input slew will cause the input capacitance to charge or discharge more quickly, leading to a shorter gate delay.

Next, the STA engine measures the output capacitance, which is the total capacitance that the gate output must drive, including the capacitance of any loads or wires connected to the output. The output capacitance affects the gate delay/slew because it determines how quickly the gate output voltage can change. A higher output capacitance will take longer to charge or discharge, leading to a longer gate delay/slew.

The input slew is determined by the previous gate output slew while also considering the wire interconnect. Once the input slew and output capacitance have been measured, the STA tool uses the delay/slew LUT to determine the output gate delay/slew. The slew LUT contains a table of slew values for different input slew and output capacitance combinations (the same applies for the delay LUT). The STA tool uses the measured input slew and output capacitance to look up the corresponding delay value in the LUT.

Figure 2.2 provides a simple example of how a simple gate delay LUT looks like. It is indexed using the input slew and total output capacitance (index 1 and index 2 accordingly).

Finally, analyzing the gate delays of all the gates in the circuit, the STA tool can identify potential timing violations and optimize the circuit's timing to meet the required performance

```
cell_rise(template) {
  index_1 ("1, 2");
  index_2 ("0.1, 0.2");
  values ("1.1, 1.2", \
          "2.1, 2.2");
}
```

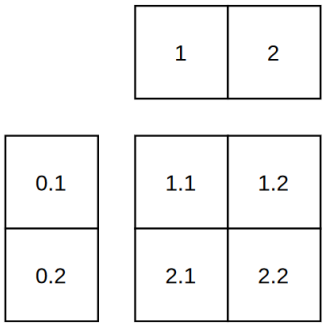

Figure 2.2: Gate Delay LUT simple example

specifications.

In conclusion, calculating the delay of a digital gate is a critical step in the design and verification of digital circuits. The delay is affected by factors such as input slew rate, output capacitance, and the internal resistance and capacitance of the gate. By using a delay lookup table, designers can accurately calculate the gate delay and perform Static Timing Analysis on the entire circuit to ensure that it meets timing requirements.

As process geometries shrink, accurate and efficient Static Timing Analysis becomes even more needed and challenging. For many decades, the most used timing model was NLDM but in advanced technology nodes (sub 45 nm) the interconnects are becoming increasingly resistive while the Miller parasitic capacitances have a higher impact on the integrity of the signal than in older technology nodes. Thus the composite current source (CCS) model [4] has replaced the NLDM model [5, 6, 7].

Chapter 3

Static Probabilities Annotation

3.1 Switching Activity

Switching activity is a crucial factor in VLSI design that impacts power consumption and signal integrity. It refers to the frequency at which digital signals in a circuit change state, from logic high to logic low or vice versa. Higher switching activity generally results in higher power consumption, as more charge is required to charge and discharge the capacitances in the circuit (these capacitances refer to the parasitic capacitances inside the logic cell or the wire capacitances introduced by the circuit interconnect). In addition, higher switching activity can also result in signal integrity issues such as noise and cross-talk. Therefore, it is essential to accurately estimate the switching activity of a circuit and optimize the design to minimize it.

Switching activity can be obtained through post-synthesis simulation in the format of SAIF, VCD, TCF and/or SAF files. By analyzing the switching activity, designers can identify the high switching activity areas in the circuit and optimize those areas to reduce power consumption and improve signal integrity.

For convenience, it can be assumed that the information extracted by the post-synthesis simulation refers to each net of the design (it can also be referring to each cell gatepin if the wire delay is taken into account).

In this Thesis, the most important variables needed by the post-synthesis simulation for the Power Analysis of each circuit net are:

- T0: Duration of time found in logic 0 state

- T1: Duration of time found in logic 1 state
- TX: Duration of time found in unknown (”X”) state
- TC: The sum of the rise (0→1) and fall (1→0) transitions captured by monitoring.

The total post-synthesis simulation time is also needed in order to compute the Static Probabilities and the toggle rate of each net.

An example of a SAIF file segment extracted from a real IC design can be found in figure 3.1 depicting the basic information analyzed above.

3.1.1 Toggle Rate and Static Probability

Static probability is the probability of a circuit net being in a specific logic state. As all probabilities, it is expressed by a floating point number between 1 and 0. In this Thesis, the notation used for the static probabilities will be as:

- P1: the static probability that a net is at logic 1 state
- P0: the static probability that a net is at logic 0 state
- PX: the static probability that a net is at unknown state (”X”)

The Static Probability of each net can be calculated as a ratio of the time duration found in each logical state (T0, T1, TX) relative to the total simulation time. For example, if the total simulation duration is $100ps$ and the time duration of a net being in the logical 1 state is $60ps$, then the P1 probability of this net is 0.7.

The Toggle Rate (TR) of each net is defined as the ratio of the rise and fall transitions sum (TC) relative to the total simulation time. For example, if the total simulation duration is $100ps$, the rise transitions of a net are 20 and the fall transitions are 21, then the toggle rate will be $(20 + 21)/100ps = 0.41 * 10^{12} Hz$.

During the development of the Power Analysis tool, an inconsistency among the Industrial Tools was found regarding the usage of the Unknown State Probability (”PX”). Three modes of integrating the unknown state probability were developed in order to correlate with the industrial tools namely:

- Weighted Split: Split the PX probability between the P1 probability and P0 probability

- Ignore Unknown: Ignore the PX probability and only consider the P1 and P0 probability
  - Note that with this method the P1 and P0 probabilities do not add up to 1
- Ignore Zero: Annotate the P1 probability and and calculate the P0 probability as $1-P1$

3.1.2 SAIF/TCF/VCD files

The Swithcing Activity of a circuit can be described by various file formats. The most used file formats are SAIF (Switching Activity Information File), TCF and VCD (Value Change Dump). The main difference of these files is that the VCD file format containes the entire waveform of each net during the simulation rather than just the information in 3.1. SAIF and TCF file formats store the same information (presented in 3.1) with changes only in the format syntax.

In figures 3.1 and 3.2, an example of the SAIF format and the TCF format is presented respectively. These files are generated for the c17 benchmark of the ISCAS85 suite (mentioned in the Experimental Methodology section 6). As explained before each format provides switching activity information for each pin of the design and the two formats are almost identical.

3.2 Static Probability Propagation

Static probability propagation is a technique used in most Power Analysis tools. It involves calculating the probabilities of each signal in the circuit being in a high (”1”) or low (”0”) logic state. These probabilities are calculated statically, based on the circuit’s topology and the logical values of the inputs. The calculated probabilities are then used to estimate both the static and the dynamic power consumption of the circuit.

A logical question may rise to he reader as to why is the Static Probability Propagation needed since the circuit has its static probabilities already annotated by SAIF/TCF/VCD files. The reason lies in 2 main factors. Firstly, the execution time of the Static Probabilities Propagation is orders of magnitude lower than the execution time of the Circuit Simulation so many designers choose the speed-up in terms of execution time versus accuracy. Second, even if the designer has already performed Circuit Simulation, the circuit netlist may change

```
(INSTANCE "c17" c_tb.dut_c17_INSTANCE
   (PORT
      (N1
         (T0 10430000000) (T1 9570000000) (TX 0)
         (TZ 0) (TB 0) (TC 999)
      )
      (N2
         (T0 10170000000) (T1 9830000000) (TX 0)
         (TZ 0) (TB 0) (TC 985)
      )
      (N3
         (T0 10140000000) (T1 9860000000) (TX 0)
         (TZ 0) (TB 0) (TC 1006)
      )
      (N6
         (T0 10180000000) (T1 9820000000) (TX 0)
         (TZ 0) (TB 0) (TC 1006)
      )
      (N7
         (T0 10070000000) (T1 9930000000) (TX 0)
         (TZ 0) (TB 0) (TC 980)
      )
      (N22
         (T0 9210000000) (T1 10790000000) (TX 0)
         (TZ 0) (TB 0) (TC 1211)
      )
      (N23
         (T0 8780000000) (T1 11220000000) (TX 0)
         (TZ 0) (TB 0) (TC 1116)
      )
      (clk
         (T0 10000000000) (T1 10000000000) (TX 0)
         (TZ 0) (TB 0) (TC 3999)
      )
   )
)
```

Figure 3.1: Example of the SAIF format

```
instance("dut_c17_INSTANCE") {
    pin() {
        "N1"  :  "0.478500  999";
        "N2"  :  "0.491500  985";
        "N3"  :  "0.493000  1006";
        "N6"  :  "0.491000  1006";
        "N7"  :  "0.496500  980";
        "clk"  :  "0.500000  3999";
        "N22"  :  "0.539500  1211";
        "N23"  :  "0.561000  1116";
        "n_0"  :  "0.771500  701";
        "n_1"  :  "0.764000  718";
        "n_2"  :  "0.616500  1104";
        "n_3"  :  "0.625000  1106";
    }
```

Figure 3.2: Example of the TCF format

after optimizations or ECO moves. If that happens, then the annotation file will not cover the entire design and some parts will be left unannotated. In this case Static Probabilities must be propagated i order to 100% annotate the design.

Static probability propagation can be performed using different algorithms, such as Boolean satisfiability (SAT) solvers or probabilistic model checking. These algorithms analyze the circuit's logic gates and their connections to determine the Static Probabilities of each signal in the circuit.

While static probability propagation provides a quick and efficient method to estimate power consumption, it does have limitations. For instance, it does not account for dynamic behavior and timing effects in the circuit. As a result, the estimates obtained through static probability propagation may not be entirely accurate. Nonetheless, this technique is still widely used in power analysis of VLSI circuits due to its efficiency and ability to quickly analyze the circuit graph compared to full simulation.

In this work, an algorithm based on Boolean Cubes is used in order to propagate the Static Probabilities in the circuit and annotate each net of the design.

3.2.1 Introduction to Boolean Cubes

Boolean cubes, also known as hypercubes or n-cubes, are a mathematical construct used in VLSI (Very Large Scale Integration) design to represent and analyze digital circuits. Boolean cubes provide a visual and intuitive representation of the Boolean functions that underlie digital circuits, making them a useful tool for VLSI designers.

In a Boolean cube, each dimension represents a single input variable to the circuit, with each vertex representing a possible combination of input values. The value at each vertex represents the output of the Boolean function for that combination of input values. By analyzing the structure of the Boolean cube, designers can gain insights into the behavior of the circuit and identify potential optimization opportunities. (include Boolean cube graph)

Across literature Boolean cubes can be used for a variety of tasks in VLSI design, such as circuit verification, optimization, and testing. By analyzing the Boolean cube of a circuit, designers can quickly identify potential design flaws or areas for improvement. Additionally, Boolean cubes can be used to generate test patterns for circuit testing, making them an essential tool in the manufacturing process.

In this work, Boolean cubes are used to to extract the Boolean function of each gate (with the specified information found in the library file of the technology) and compute the Static Probabilities (”P1” and ”P0”) of the cell output pin based on the Static Probabilities of the cell input pins.

Overall, Boolean cubes provide a powerful tool for VLSI designers to understand and analyze digital circuits. By leveraging the insights provided by Boolean cubes, designers can optimize circuit performance, improve energy efficiency, and minimize the risk of errors or faults in the final product.

In order to explain theusage of Boolean cubes in the developement of this Power Analysis Engine, the cubes representation must be introduced. Each cube is an array of values, one for each literal. Each Boolean function is an array of cubes that describe the function. The possible values for each literal in every term of the function are:

- INVALID
  - Note: If any of the literals has this value, then the entire cube is considered invalid and discarded
- ”1”: The literal is positive
- ”0”: The literal is negative
- ”X”-not care: The literal is absent from the term

In the following example, a cube representation of a Boolean function is presented to clarify the meaning described above:

$f = a'd' + a'b + ab' + ac'd$

The corresponding cube representation of f is presented in the following table:

	a	b	c	d	Term
cube 1	0	X	X	0	$a'd'$
cube 2	0	1	X	X	$a'b$
cube 3	1	0	X	X	ab'
cube 4	1	X	0	1	$ac'd$

Table 3.1: Boolean Cubes Representation of function f

The detailed theory behind Boolean Cubes will not be explained in this work as it is a vast field and it is not the focus of this Thesis. During this work, the Boolean Cubes were used merely as a tool to provide the Power Analyzer with the extra feature of Static Probability Propagation.

3.2.2 Static Probabilities Propagation Methodology

Two algorithms were developed during this Thesis and are explained below. One aims towards the Probability Annotation of a single gate output 2 and the other one is responsible for the Static Probability Propagation in the entire design 1.

Algorithm 2 is designed as a step in order to propagate static probabilities in a circuit in the context of power analysis. It takes as input a gate output, the corresponding annotated gate input pins, and the boolean cubes output representation, and annotates the gate output with its probability of outputting a logical one and a logical zero.

The algorithm starts by expanding the boolean cube representation of the gate output to its corresponding minterms. It then loops over each minterm and calculates the probability of it outputting a logical one. This is done by looping over each input pin and multiplying the probability of that input pin being a logical one or a logical zero based on the value of the input pin in the minterm. The probabilities of each input pin are obtained from their annotated values as an input. The algorithm first calculates the probability of the output being at logic one.

After calculating the probability of the gate outputting a logical one, the algorithm continues by calculating the probability of the gate outputting a logical zero. This is done by first expanding the boolean cube representation of the gate output to its offset minterms, that were previously calculated. The offset minetrms are essentially the minterms that satisfy the output boolean function evaluating to logic zero.. These offset minterms are then used in the same process as before, where the probability of each minterm is calculated based on the input probabilities of the corresponding input pins of the gate.

Finally, the probabilities of the gate outputting a logical one and a logical zero are stored in the Gate Output data structure for further use. These probabilities are later used in order to determine the power consumption of the gate under different input patterns.

The algorithm ”Static Probabilities Propagation” 1 is used to propagate the static probabilities through a circuit design. The algorithm takes in a levelized design and an initial gate pin list as input, and its goal is to annotate the gate outputs with their respective static probabilities and propagate these probabilities through the circuit to each input pin.

The algorithm starts by initializing a levels data structure, which is used to store the gate pins in each level of the circuit. It then iterates over the gate pins in the initial gate pin list and adds them to the first level in the levels data structure. In case of a full circuit probability

propagation, the initial gate pin list is a list with the Primary Inputs of the design.

Next, the algorithm iterates over each level in the levels data structure, and for each gate pin in the level, it annotates the gate output static probability using the ”annotate_gate_output_static_probability” subroutine. This subroutine is algorithm 2

After annotating the gate output, the algorithm traces the successors of the gate pin using the ”trace_successors_gate_pins” subroutine and iterates over them. For each successor, if it is an input pin, the algorithm sets the zero and one probability of the successor to the zero and one probability of the gate pin, respectively. This is performed because the output pin of a net (i.e. the driver) and the input pin/pins of a net (i.e. the receiver). It then adds the successor to the next level in the level's data structure.

Finally, the algorithm repeats this process for each level in the levels data structure until all gate pins have been annotated with their respective static probabilities. The output of the algorithm is a design with each gate pin annotated with its coresponding static probability.

3.2.3 Application on SET/SEU related methodologies

The aforementioned Static Probabilities Propagation Algorithm can be utilized in Single Event Transient (SET) analysis in VLSI circuits. An SET is a temporary change in the logic state of a circuit caused by a high-energy particle strike, such as a cosmic ray. These transient faults can cause serious reliability issues and can lead to system failure.

Algorithm 1 can be used to estimate the gate pins static probabilities of a circuit by analyzing the circuit's structure and the probability of each gate being in a high or low state. This information can later be used in order to properly propagate an SET throughout the circuit and estimate whether this glitch will be captured as valid data by a flip-flop. This information can be used to design more reliable and fault-tolerant circuits.

Algorithm 1 Static Probabilities Propagation

Input: Levelized Design, Initial Gate Pin list

Output: Static Probabilities are Propagated in the design

```
1: levels ← initialize_levels_structure();
2: for GatePin in InitialGatePinList do
3:     levels ← add_to_level(levels, Gatepin);
4: end for
5: for level in levels do
6:     for GatePin in level do
7:         if GatePin is output then
8:             annotate_gate_output_static_probability(GatePin);
9:         end if
10:        successors ← trace_successors_gate_pins(GatePin);
11:        for successor in successors do
12:            if successor is input then
13:                successor.zero_probabality ← GatePin.zero_probability
14:                successor.one_probabality ← GatePin.one_probability
15:            end if
16:            levels ← add_to_level(levels, successor);
17:        end for
18:    end for
19: end for
```

Algorithm 2 Gate Output Static Probability Annotation

Input: Gate Output, Corresponding annotated Gate Input pins, Boolean Cubes Output Representation

Output: Gate Output is annotated

1: $cubes\_matrix \leftarrow Gate\_Output.Boolean\_Cubes\_Representation$;
2: $expanded_cubes_matrix \leftarrow$EXPAND_TO_MINTERMS($cubes\_matrix$) ;
3: $logic\_one\_probability \leftarrow 0$;
4: **for** $minterm$ in $expanded\_cubes\_matrix$ **do**
5: $cube\_probability \leftarrow 1$;
6: **for** $literal = 0$ **to** $Total\_Input\_Pins$ **do**
7: **if** $minterm[literal] == 0$ **then**
8: $cube\_probability * = GateInputs[literal].Probabilities(0)$;
9: **else if** $mintern[literal] == 1$ **then**
10: $cube\_probability * = GateInputs[literal].Probabilities(1)$;
11: **end if**
12: **end for**
13: $logic\_one\_probability \leftarrow logic\_one\_probability + cube\_probability$;
14: **end for**
15: $off\_set\_cubes \leftarrow$ EXPAND_TO_OFFSET_MINTERMS($cubes\_matrix$);
16: $logic\_zero\_probability \leftarrow 0$;
17: **for** $minterm$ in off_set_cubes **do**
18: $cube\_probability \leftarrow 1$;
19: **for** $literal = 0$ **to** $Total\_Input\_Pins$ **do**
20: **if** $minterm[literal] == 0$ **then**
21: $cube\_probability * = GateInputs[literal].Probabilities(0)$;
22: **else if** $mintern[literal] == 1$ **then**
23: $cube\_probability * = GateInputs[literal].Probabilities(1)$;
24: **end if**
25: **end for**
26: $logic\_zero\_probability \leftarrow logic\_zero\_probability + cube\_probability$;
27: **end for**
28: $Gate\_Output.logic\_zero\_probability \leftarrow logic\_zero\_probability$;
29: $Gate\_Output.logic\_one\_probability \leftarrow logic\_one\_probability$;

Chapter 4

Power Modelling

In the following chapter the Power Modelling in VLSI Digital Systems will be discussed. In order to understand the modelling of the various Power Categories, it is important to state the basic terms that will be referenced later on. The basic theory underlying the various power dissipation sources stems from the definition of *Instantaneous Power* and *Average Power*. The *Instantaneous Power* is defined as the product of *Voltage* and *Current*

$$P(t) = V(t)I(t) \tag{4.1}$$

While the *Average Power* is defined as the Energy over Time

$$P_{average} = \frac{Energy}{Time} \tag{4.2}$$

Taking into account that *Energy* is defined as the interval of *Power*, equation 4.2 can be expressed as:

$$P_{average} = \frac{1}{T}\int_0^T P(t)dt \tag{4.3}$$

where *T* is Time and *P* is expressed in Watts (W) as Energy is defined in Joules (J) and Time is defined in seconds (s). Note that $1W = 1J/s$.

Figure 4.1 shows a CMOS inverter driving a capacitive load. When the input changes from high to low, the pMOS transistor activates and charges the load to VDD, while when the input changes from low to high the nMOS transistor activates and the load is discharged from VDD to 0. According to Neil Weste and David Harris [8], only half of the supply energy is stored in the capacitor while the other half is dissipated in the transistors (i.e. converted to heat).

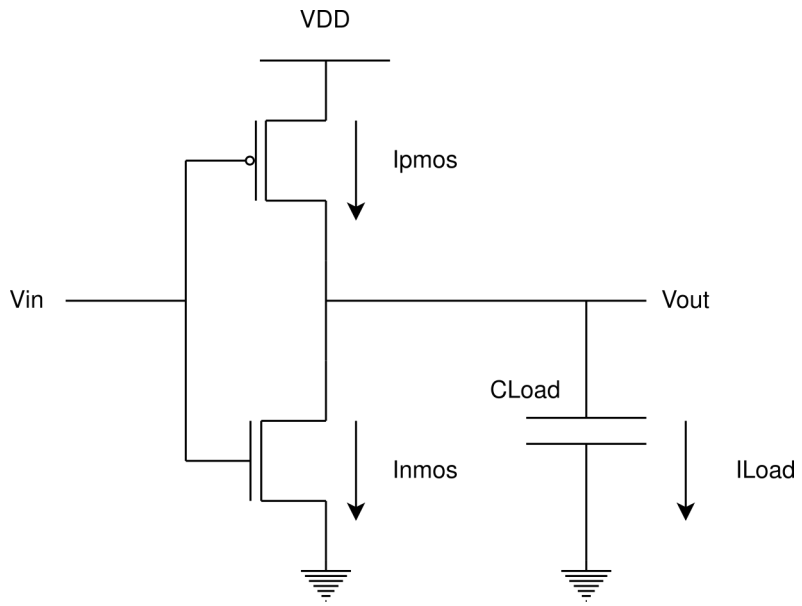

Figure 4.1: Example of Power dissipation on a CMOS inverter

4.1 Power Categories

Power dissipation can be separated in two main factors [8], namely Dynamic Power and Static Leakage Power. The power dissipation sources for each of the two are:

- Dynamic Power dissipation:
  - charging and discharging of output loads
    - referenced to as **Switching Power**
  - current flowing through PMOS and NMOS transistors when simultaneously switching
    - referenced to as **Internal Power**
- **Static Leakage Power** Dissipation:
  - sub-threshold leakage current
  - gate dielectric leakage
  - junction leakage from source/drain

Adding the aforementioned Power Categories into one, gives the total Power dissipation of the integrated circuit.

$$P_{total} = P_{dynamic} + P_{leakage}$$
$$P_{dynamic} = P_{switching} + P_{internal} \tag{4.4}$$

In the following sections, the modeling of each of the Power Categories (**Static Leakage Power**, **Internal Power** and **Switching Power**) will be explained in depth.

4.2 Static Leakage Power

In digital VLSI systems, static leakage power is a major concern due to its continuous power consumption even when the circuit is in idle mode. As process technology continues to scale down, the static leakage power has become a significant portion of the total power consumption, and therefore, accurate modeling of static leakage power is essential for power optimization in digital VLSI systems.

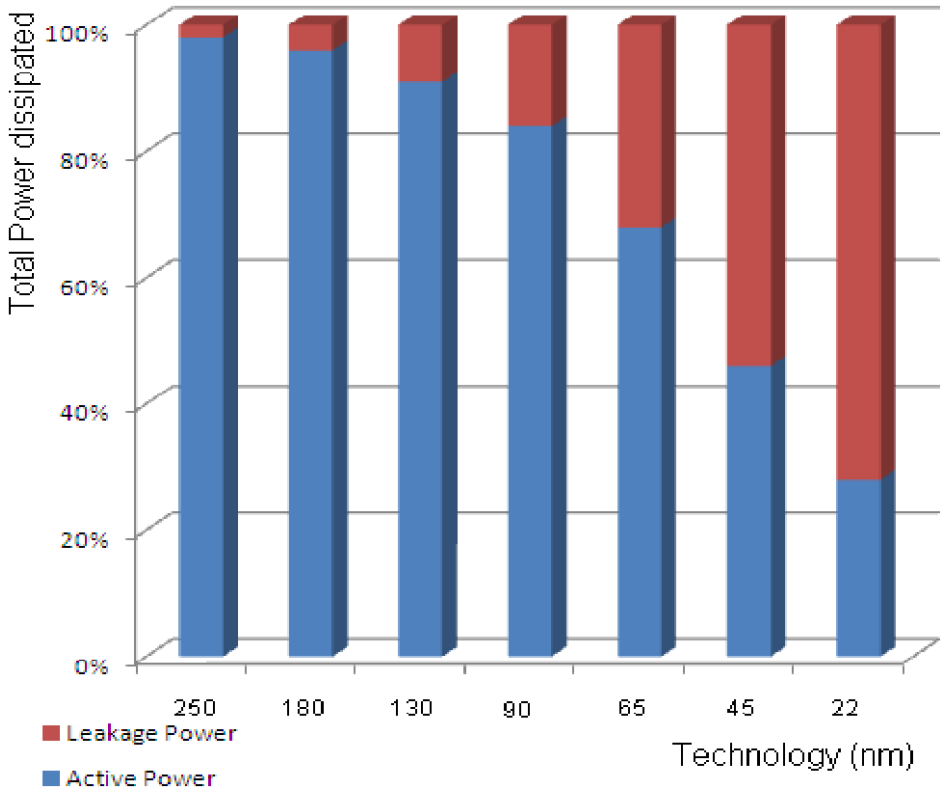

Figure 4.2: Leakage Power in different technologies [1]

As analyzed in 4.1 the Static Leakage Power consists of several factors and it is not practical to evaluate the Static Leakage Power of each circuit gate by calculating using analytical

models. This approach would be extremely time-consuming and computationally heavy. Instead, the Technology foundry has pre-characterized each technology gate providing the static leakage power (pre-characterized inside the .lib file) [9]. Depending on the technology and the technology foundry, there could be only one static leakage power value or an array of values, each based on the input gate pins state [10].

In advanced technology nodes, it is almost certain that the second option will be given to the engineer meaning the array of static leakage values. The computation of the actual gate leakage power based on these values will be analyzed in depth in 5.1

In 4.1, the factors for Static Power Dissipation were namely presented. First, the subthreshold leakage current will be explained. Subthreshold leakage current flows when a transistor is supposed to be OFF [8]. A simplified equation for the subthreshold leakage current is presented in 4.5

$$I_{sub} = I_{off} 10^{\frac{V_{gs}+\eta(V_{ds}-VDD)-\kappa_\gamma V_{sb}}{S}} \tag{4.5}$$

where Ioff is the subthreshold current at Vgs = 0 and Vds = VDD, and S is the subthreshold slope which indicates how much the V_{gate} must drop to decrease the leakage current. The body coefficient κ_γ describes how the body effect modulates the threshold voltage.

There are many techniques for reducing the sub-threshold leakage current. The two most used are reducing the VDD voltage of the circuit and applying a negative body voltage [8]. Another way to reduce the sub-threshold leakage current is to implement the ASIC in a Silicon-on-Insulator (SOI) technology. For many years, silicon-on-insulator (SOI) has been an area of study, but its significance has been increased commercially since IBM started using SOI for PowerPC microprocessors in 1998. The main difference between SOI and traditional bulk CMOS technology is that SOI uses an insulating oxide material to surround the transistor source, drain, and body instead of the conductive substrate or well, which is commonly referred to as the bulk [11].

Another source of Static Leakage Power dissipation is the Gate Dielectric Leakage. Gate leakage is the phenomenon in which carriers tunnel through a thin gate dielectric when a voltage is applied across the gate, usually when the gate is switched on. The circuit technology process (i.e. the foundry) often specifies the gate leakage current I_G in either nA/mm^2 for a minimum-length gate or in A/mm^2 of the transistor gate. The magnitude of gate leakage is heavily dependent on the thickness of the dielectric material. Typically, the dielectric thick-

ness is chosen to limit gate leakage to acceptable levels during the manufacturing process. In the case of pMOS gates with ordinary SiO_2, gate leakage is usually an order of magnitude lower and may not be significant, but it can become significant for other types of gate dielectrics. Gate leakage also depends on the voltage across the gate which means that the higher the V_{gs} voltage the more gate leakage will be observed in the transistor [12].

Finally, Junction leakage is the result of a source or drain diffusion region being at a different potential than the substrate. Although the usual leakage from reverse-biased diodes is generally negligible, mechanisms such as band-to-band tunneling (BTBT) and gate-induced drain leakage (GIDL) can cause leakage currents that approach subthreshold leakage levels in high-Vt transistors. BTBT is most significant when a strong reverse bias is applied between the drain and body (such as when $Vdb = -VDD$ for an nMOS transistor). GIDL, on the other hand, is most significant when the transistor is turned off and a strong bias is applied to the drain (such as when $Vgd = -VDD$ for an nMOS transistor). While junction leakage is typically less significant than other types of leakages, it can still be expressed in nA/μm of transistor width when necessary to consider.

4.3 Dynamic Power

As stated in equation 4.4, Dynamic Power consists of the Switching Power and the Internal Power [8]. In figure 4.3 a comparison between the Static Leakage Power and the Dynamic Power can be observed across multiple technology nodes. It is clear that the Dynamic Power has a steady increase as technologies shrink. It is therefore essential, that the ASIC engineers have an accurate estimation of the Dynamic Power dissipated in their circuit.

4.3.1 Internal Power

Internal power is the power dissipated due to the short-circuit current of the transistors while both PULL-UP and PULL-DOWN networks are partially ON while the input switches. Naturally, this power is increased if the input slews are higher because the transistors remain simultaneously ON for a longer time [13]. Internal Power LUTs [9] are extracted by the characterization of the desired technology. LUT-based modeling of internal power is a fast and efficient approach that can provide accurate results, especially for circuits with large fan-out or complex logic. A simple example of a LUT is presented in 2.2.

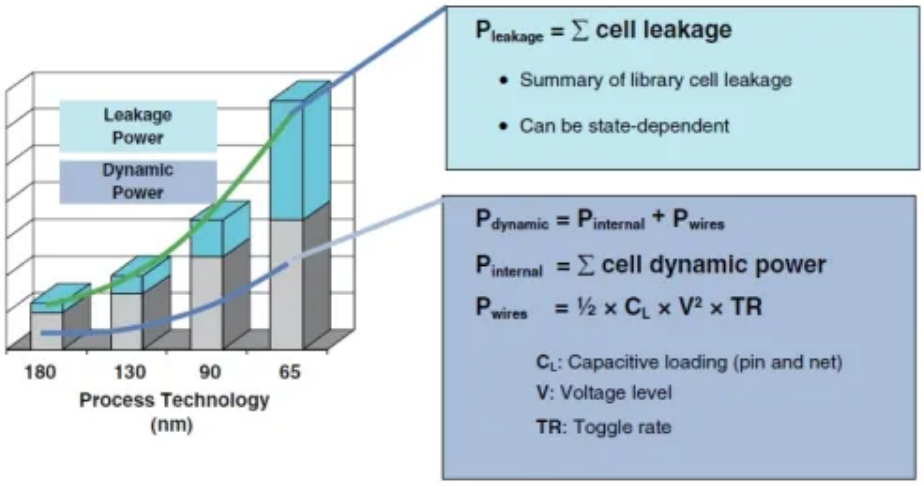

Figure 4.3: Comparison between the Static Leakage Power dissipation and the Dynamic Power dissipation over the years `https://semiengineering.com/knowledge_centers/low-power/low-power-design/power-consumption/`

Along with the value interpolation of LUTs, power factors must be calculated in order to provide a weight for the internal power dissipation of each output pin. These power factors are a function of the Static Probabilities and the transitions of each gate [10].

4.3.2 Switching Power

Switching power, on the other hand, is the power dissipated due to the charging and discharging of the load capacitance during output switches. It is directly proportional to the load capacitance, voltage, and frequency of the switching activity based on the equation 4.6.

$$P_{switching} = \frac{1}{2} C V_{dd}^2 \alpha \tag{4.6}$$

where C is the output load capacitance, Vdd is the supply voltage and α is the net switching activity.

Output load C refers to the capacitance seen by the output of a logic gate in a circuit. It represents the total capacitance that the output driver needs to charge and discharge during each switching cycle, depending on the transition (rise or fall). The output load capacitance is composed of two main components: the intrinsic capacitance of the gate and the parasitic capacitance of the interconnects, which can be significant and highly dependent on the layout of the circuit. There has been a lot of research progress in the field of parasitic extraction and the computation of the capacitance of an interconnect as it is a field that sparks interest in

many researchers.

Switching Power has a quadratic dependence on the supply voltage. This means that a lower supply voltage significantly reduces the switching power consumption of a chip. The naive approach to tackle the Switching Power dissipation problem would be to lower the supply voltage throughout the chip. This cannot be implemented in real chip applications because the chip speed is also proportional to the supply voltage. One approach is to divide the circuit into blocks and apply different $voltage\ domains$ [14]. A different method for approaching this problem is known as Clustered Voltage Scaling (CVS) [15], which involves the use of two supply voltages in a single block. In this approach, gates at the beginning of the path use V_{DDH}, while noncritical gates further along the path use V_{DDL}. The voltages are arranged in such a way that the path never crosses from a V_{DDL} gate to a V_{DDH} gate within a combinational logic block. This means that level converters are only necessary at the registers endpoints and not throughout the circuit.

Switching activity α has already been deeply explained in section 3.1 of this Thesis.

Chapter 5

Power Calculation

In this chapter, the Power Calculation will be explained. It is divided into three parts namely:

- Static Leakage Power
- Switching Power
- Internal Power

The ”When Probability Calculation” algorithm 0 is used to evaluate the probability of a given when condition. The algorithm takes two inputs: the library pin and the when condition. The library pin refers to a specific input or output pin of a digital circuit. The when condition is a Boolean expression that specifies the conditions under which the library pin is expected to be in a particular state. The algorithm works by iterating through each input pin in the library, finding the position of the pin in the when condition string, and evaluating whether the pin is positive or negative in the expression. If the pin is positive, the algorithm retrieves the probability of the pin being in a logic one state, and multiplies it with the total probability calculated so far. Similarly, if the pin is negative, the algorithm retrieves the probability of the pin being in a logic zero state and multiplies it with the total probability calculated so far. The final output of the algorithm is the total probability calculated.

5.1 Static Leakage Power

The algorithm ”Static Leakage Power Calculation” 4 is designed in order to calculate the total static leakage power dissipated in a VLSI design based on a Static Probability Annotated

Algorithm 3 When Probability Calculation

Input: Library pin, when condition

Output: Probability of the when condition

1: $TotalProbability \leftarrow 1$;
2: **for** $InputPin$ in $LibraryPin$ **do**
3: $RelatedPosition \leftarrow find\_pin\_string\_related\_position(InputPin, When)$;
4: **if** $When[RelatedPosition - 1] \neq$! **then**
5: /* The Input Pin is positive in the when condition string */
6: $LogicOne \leftarrow$ get_logic_one_probabilty($InputPin$);
7: $TotalProbability \leftarrow TotalProbability * LogicOne$;
8: **end if**
9: **if** $When[RelatedPosition - 1] ==$! **then**
10: /* The Input Pin is negative in the when condition string */
11: $LogicZero \leftarrow$ get_logic_zero_probabilty($InputPin$);
12: $TotalProbability \leftarrow TotalProbability * LogicZero$;
13: **end if**
14: **end for**
15: **return** $TotalProbability$;

Gate-Level Netlist. The input to the algorithm is the Static Probability Annotated Gate-Level Netlist, which is a netlist annotated with static probabilities for each gate.

The algorithm starts by resetting any previously calculated leakage power. It then initializes the variable ”TotalLeakagePower” to 0. This variable will be used to accumulate the total leakage power dissipated in the design.

The algorithm then enters a loop that iterates over each gate in the design. For each gate, it retrieves the library cell associated with that gate. It initializes the variable ”DefaultWhenProbability” to 1, which represents the probability that none of the leakage power tables for this library cell apply.

It then enters another loop that iterates over each leakage power table associated with the library cell. For each leakage power table, the algorithm calls the ”calculate_when_probability()” function (described in 0) to calculate the when probability for that leakage power table based on the gate’s static probability annotation and the library cell’s when condition. It then retrieves the actual leakage power value from the Static Leakage

LUT. The algorithm then updates the ”TotalLeakagePower” variable by adding the product of the leakage power value and the when probability.

After iterating over all the leakage power tables for the library cell, the algorithm checks whether the sum of all the when probabilities calculated in the previous loop is less than 1. If so, it means that there is a probability that none of the leakage power tables apply. In that case, the algorithm uses the remaining probability to calculate the default leakage power for the library cell. If the default leakage power is non-zero, the algorithm updates the ”TotalLeakagePower” variable by adding the product of the default leakage power and the remaining probability.

Once the algorithm has processed all the gates in the design, it returns the total leakage power dissipated in the design, which is stored in the ”TotalLeakagePower” variable.

In summary, the algorithm calculates the total leakage power dissipated in a VLSI design based on a Static Probability Annotated Gate-Level Netlist by iterating over each gate in the design and, for each gate, iterating over each leakage power table associated with the library cell of the gate. It calculates the when probability for each leakage power table and updates the total leakage power accordingly. If the sum of all the when probabilities is less than 1, it uses the remaining probability to calculate the default leakage power for the library cell.

5.2 Switching Power

The purpose of this algorithm is to calculate the total switching power of a design. It takes in a gate-level netlist design as input and returns the total switching power as output.

The algorithm starts by resetting any previously calculated switching power.The next step is to get the operating conditions of the library. This is required to calculate the switching power of each gate. Variable ’TotalSwitchingPower’ is initialized to 0 to accumulate the total switching power of the design.

The algorithm then enters a loop that iterates through each gate in the design. For each gate, it first gets the output pin of the gate. It then calculates the switching activity of the output pin by dividing the number of times the pin toggles during the simulation duration.

Next, the algorithm gets the total rise and fall output capacitance of the output pin. It then calculates the average output capacitance as the sum of the rise and fall capacitances divided by two. This is done because half of the transitions will be rise ($arrowup$) transitions and the

Algorithm 4 Static Leakage Power Calculation

Input: Static Probability Annotated Gate-Level Netlist

Output: Total Static Leakage Power dissipated

```
1: reset_previously_calculated_leakage_power();
2: TotalLeakagePower ← 0;
3: for Gate in Design do
4:     LibraryCell ← get_library_cell(Gate);
5:     DefaultWhenProbability ← 1;
6:     for LeakagePowerTable in LibraryCell do
7:         WhenProbability ← calculate_when_probability(LibraryCell.WhenCondition);
8:         LeakagePowerValue ← get_leakage_value(LeakagePowerTable);
9:         TotalLeakagePower ← TotalLeakagePower + LeakagePowerValue∗
10:                                                 WhenProbability;
11:        DefaultWhenProbability ← DefaultWhenProbability−
12:                                                 WhenProbability;
13:    end for
14:    /* If when probabilities do not add up to 1, */
15:    /* use the remaining probability in order to calculate the default leakage power */
16:    if DefaultWhenProbability ≥ 0 then
17:        DefaultLekeageValue ← get_default_leakage(LibraryCell);
18:        if DefaultLekeageValue ≠ 0 then
19:            TotalLeakagePower ← TotalLeakagePower+DefaultLekeageValue∗
20:                                                 DefaultWhenProbability;
21:        end if
22:    end if
23: end for
24: return TotalLeakagePower;
```

other half will be fall ($arrowdown$) transitions.

Using the operating conditions, the output capacitance, and the switching activity, the algorithm calculates the switching power of the gate using the equation:

$$P_{Switching} = \frac{VDD^2 * C * \alpha}{2} \tag{5.1}$$

Finally, the calculated switching power of the gate is added to the 'TotalSwitchingPower' variable. After all gates have been processed, the algorithm returns the total switching power as output.

Algorithm 5 Switching Power Calculation

Input: Gate-Level Netlist

Output: Total Switching Power

```
1: reset_previously_calculated_switching_power();
2: OperatingConditions ← get_library_operating_conditions();
3: TotalSwitchingPower ← 0;
4: for Gate in Design do
5:     OutputPin ← Gate.OutputPin;
6:     SwitchingActivity ← Gate.OutputPin.ToggleCount/SimulationDuration;
7:     RiseCapacitance ← get_total_rise_output_capacitance(Gate.OutputPin);
8:     FallCapacitance ← get_total_fall_output_capacitance(Gate.OutputPin);
9:     OutputCapacitance ← (RiseCapacitance + FallCapacitance)/2;
10:    SwitchingPower ← (OperatingConditions.VDD)^2 *
11:                                  OutputCapacitance * SwitchingAcitivity/2;
12:    TotalSwitchingPower ← TotalSwitchingPower + SwitchingPower;
13: end for
14: return TotalSwitchingPower;
```

5.3 Internal Power

The "Output Pin Internal Power Calculation" algorithm 6 calculates the internal power dissipation of an output pin. It takes three inputs: Rise Internal Power Arcs, Fall Internal Power Arcs, and Power Factor. The output of this algorithm is the total internal power dissipated by the output pin.

The algorithm first initializes a variable "PinPower" to 0. Then, it iterates through the Rise Internal Power Arcs using a for-loop. For each Rise Internal Power Arc, the algorithm obtains the related input slews and calculates the Rise Power using the interpolate2D function with the corresponding input parameters. It then multiplies the Rise Power by the Power Factor and adds it to the PinPower variable.

After completing the Rise Arcs loop, the algorithm iterates through the Fall Internal Power Arcs using another for-loop. For each Fall Internal Power Arc, the algorithm obtains the related input slews and calculates the Fall Power using the interpolate2D function with the corresponding input parameters. It then multiplies the Fall Power by the Power Factor and adds it to the PinPower variable.

Finally, the algorithm returns the PinPower variable, which contains the total internal power dissipated by the output pin.

Algorithm 6 Output Pin Internal Power Calculation

Input: Rise internal Power Arcs, Fall Internal Power Arcs, Power Factor

Output: Output Pin Internal Power Dissipated

1: $PinPower \leftarrow 0$;
2: **for** Arc in $RiseArcs$ **do**
3: $RelatedInputSlews \leftarrow Arc.RelatedInput.Slews$;
4: $RisePower \leftarrow$ interpolate2D(Arc, $Pin.OutputRiseCap$,
5: $RelatedInputSlews.Rise$, $Pin.OutputFallCap$,$RelatedInputSlews.Fall$);
6: $PinPower \leftarrow RisePower * PowerFactor$;
7: **end for**
8: **for** Arc in $FallArcs$ **do**
9: $RelatedInputSlews \leftarrow Arc.RelatedInput.Slews$;
10: $FallPower \leftarrow$ interpolate2D(Arc, $Pin.OutputRiseCap$,
11: $RelatedInputSlews.Rise$, $Pin.OutputFallCap$,$RelatedInputSlews.Fall$);
12: $PinPower \leftarrow PinPower + FallPower * PowerFactor$;
13: **end for**
14: **return** $PinPower$;

The ”Input Pin Internal Power Calculation” algorithm 7 takes as input the Rise Internal Power Arcs and Fall Internal Power Arcs, as well as a Power Factor, and calculates the amount of power dissipated by an input pin. The algorithm initializes a variable, $PinPower$, to zero. It then iterates over all of the RiseArcs and FallArcs.

For each RiseArc, it retrieves the related input slews and uses them to interpolate the RisePower using a one-dimensional interpolation function. It then multiplies the RisePower by the Power Factor and adds it to the $PinPower$ variable. For each FallArc, the algorithm performs a similar process, retrieving the related input slews and using them to interpolate the

FallPower. The FallPower is then multiplied by the Power Factor and added to the $PinPower$ variable.

Finally, the algorithm returns the $PinPower$ variable, which represents the total internal power dissipated by the input pin.

Compared to the ”Output Pin Internal Power Calculation” algorithm 6, the main difference in this algorithm lies in the interpolation stage. While the previous algorithm uses a two-dimensional interpolation function, this algorithm uses a one-dimensional interpolation function for both the RisePower and FallPower calculations.

Algorithm 7 Input Pin Internal Power Calculation

Input: Rise internal Power Arcs, Fall Internal Power Arcs, Power Factor

Output: Input Pin Internal Power Dissipated

1: $PinPower \leftarrow 0$;
2: **for** Arc in $RiseArcs$ **do**
3: $RelatedInputSlews \leftarrow Arc.RelatedInput.Slews$;
4: $RisePower \leftarrow$ interpolate1D(Arc,$RelatedInputSlews.Rise$,
5: $RelatedInputSlews.Fall$);
6: $PinPower \leftarrow RisePower * PowerFactor$;
7: **end for**
8: **for** Arc in $FallArcs$ **do**
9: $RelatedInputSlews \leftarrow Arc.RelatedInput.Slews$;
10: $FallPower \leftarrow$ interpolate1D(Arc,$RelatedInputSlews.Rise$,
11: $RelatedInputSlews.Fall$);
12: $PinPower \leftarrow PinPower + FallPower * PowerFactor$;
13: **end for**
14: **return** $PinPower$;

Algorithm 8 takes as input a netlist that is timing annotated and static probability annotated. The objective is to calculate the total internal power dissipated by the entire design. The algorithm begins by resetting any previously calculated internal power.

Next, the algorithm iterates through each gate in the design and for each gate, it iterates through each pin. For each pin, the algorithm calculates the switching activity, which is the number of toggles divided by the simulation duration.

If the pin is an output pin, the algorithm calculates the when probabilities and the power factor. The when probabilities represent the probability distribution of the timing of the output transition, while the power factor represents the portion of the internal power dissipated by the output pin. The algorithm then retrieves the rise and fall power arcs associated with the output pin and calculates the output pin's internal power dissipation using the ”calculate_output_internal_power” algorithm 6. Finally, the internal power dissipation of the output pin is added to the total internal power.

If the pin is an input pin, the algorithm calculates the when probabilities and the power factor. The algorithm then retrieves the rise and fall power arcs associated with the input pin and calculates the input pin's internal power dissipation using the ”calculate_input_internal_power” algorithm 7. Finally, the internal power dissipation of the input pin is added to the total internal power.

Once all pins for all gates have been processed, the algorithm returns the total internal power dissipated by the entire design.

Algorithm 8 Internal Power Calculation

Input: Annotated Netlist (Timing Annotated and Static Probability Annotated)

Output: Total Internal Power Dissipated

```
1: reset_previously_calculated_internal_power();
2: TotalInternalPower ← 0;
3: for Gate in Design do
4:     for Pin in Gate do
5:         PinPower ← 0;
6:         SwitchingActivity ← Pin.ToggleCount/SimulationDuration;
7:         if Pin is output then
8:             WhenProbabilities ← calculate_when_probability(Pin, Gate);
9:             PowerFactor ← calculate_power_factor(Pin, WhenProbabilities,
10:                                                  SwitchingActivity);
11:            RiseArcs ← get_rise_power_arcs(Pin);
12:            FallArcs ← get_fall_power_arcs(Pin);
13:            PinPower ← calculate_output_internal_power(Pin, RiseArcs, FallArcs,
14:                                                       PowerFactor);
15:            TotalInternalPower ← TotalInternalPower + PinPower;
16:        end if
17:        if Pin is input then
18:            WhenProbabilities ← calculate_when_probability(Pin, Gate);
19:            PowerFactor ← calculate_power_factor(Pin, WhenProbabilities,
20:                                                 SwitchingActivity);
21:            RiseArcs ← get_rise_power_arcs(Pin);
22:            FallArcs ← get_fall_power_arcs(Pin);
23:            PinPower ← calculate_input_internal_power(Pin, RiseArcs, FallArcs,
24:                                                      PowerFactor);
25:            TotalInternalPower ← TotalInternalPower + PinPower;
26:        end if
27:    end for
28: end for
29: return TotalInternalPower;
```

Chapter 6

Experimental Methodology

In this part of the Thesis, the Experimental Methodology used to evaluate the Power Analysis engine developed will be examined. In figure 6.1, the Top Level Flow is presented. First, the necessary files are gathered and modified in order to begin the flow. Along with the initial necessary files, further files and information must be generated for each design in order to perform the Power Analysis such as the SAIF/TCF file. Next, Power Analysis is performed using both a state-of-the-art Industry Tool and the Power Analysis Engine developed. Finally, the reports from both tools are gathered and the Error is calculated.

6.1 Experimental Flow

In the following section, the Experimental Flow used to evaluate this Thesis work will be analyzed. The complete flow is presented in figure 6.2.

First, each design under test is synthesized using the Design RTL and the timing constraints. Synthesis refers to the process of generating a gate-level netlist from a higher-level hardware description language (HDL) code [16]. This netlist describes the integrated circuit (IC) being designed in the form of logic gates. The synthesis step involves a number of important tasks, such as logic optimization to minimize power consumption and area, technology mapping to map the design onto specific physical components, and timing analysis to ensure that the circuit meets its performance specifications. Ultimately, the output of the synthesis step is a gate-level netlist that can be used as the input for the next steps in the design process, such as layout and verification.

Another necessary Synthesis output needed for this work is the Standard Delay Format

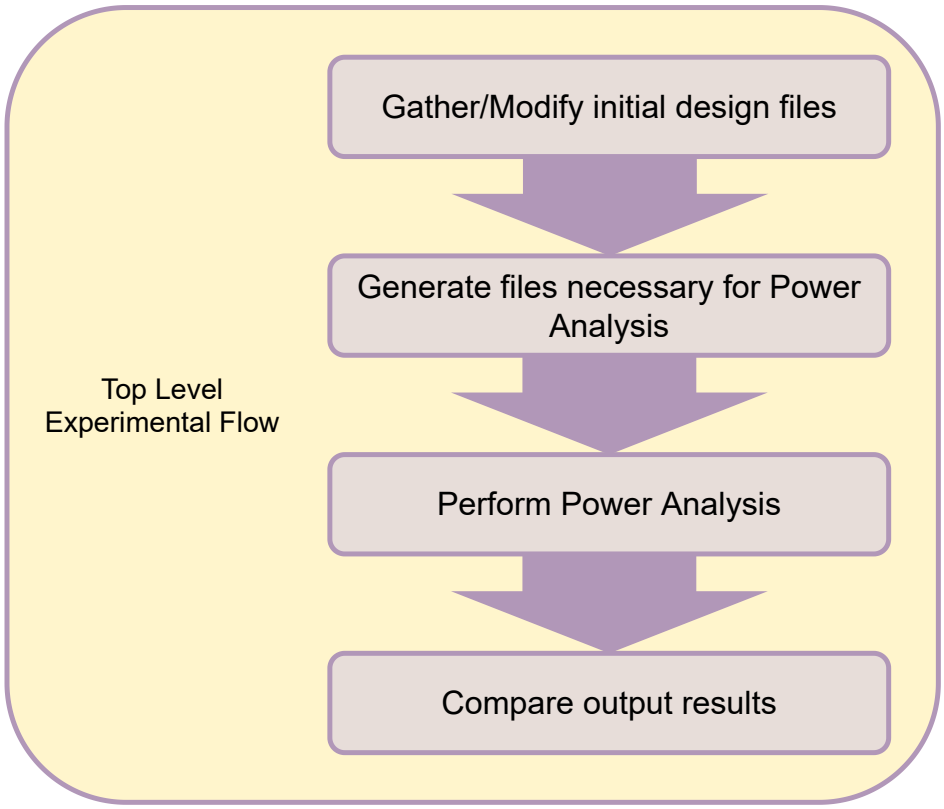

Figure 6.1: Top Level Flow Diagram

(SDF) [17] file. The Standard Delay Format is a file format used to specify the timing information of a digital circuit. The timing information is represented in terms of delays associated with various circuit elements such as wires, gates, and pins. It is an ASCII format file which includes path delays, timing constraint values, and interconnect delays.

Next step in this Experimental Flow is the circuit simulation. The simulation step is an important part of the VLSI SPAR (Synthesis, Place, and Route) flow, as it allows designers to verify the functionality and timing of their design before it is fabricated. During the simulation step, the digital circuit is tested by applying input signals to its inputs and observing the resulting output signals. This is done using specialized software tools known as simulation engines or simulators.

Apart from checking the functionality of the gate-level netlist compared to the Design RTL, the necessary output from the simulator is the SAIF/TCF file which indicates the switching activity of the circuit. The simulation tool monitors the switching activity of the circuit by tracking the number of transitions that occur in each signal over time.

Having the SAIF/TCF file, the Power Analysis step is performed. The same inputs are used in both the Industrial Power Analysis tool and the Power Analysis Engine developed in

this current work. These inputs are the Design Netlist, the Timing constraints of the design and the SAIF/TCF file containing the switching activity information.

Finally, the last step of the Experimental Flow is to compare the results extracted from both tools and measure the computation error. This flow is executed iteratively for each of the benchmarks used in this work. The benchmarks selected to evaluate the work are a subset of the ISCAS85 benchmarks [18], a benchmark suite commonly used in the field of Power Analysis.

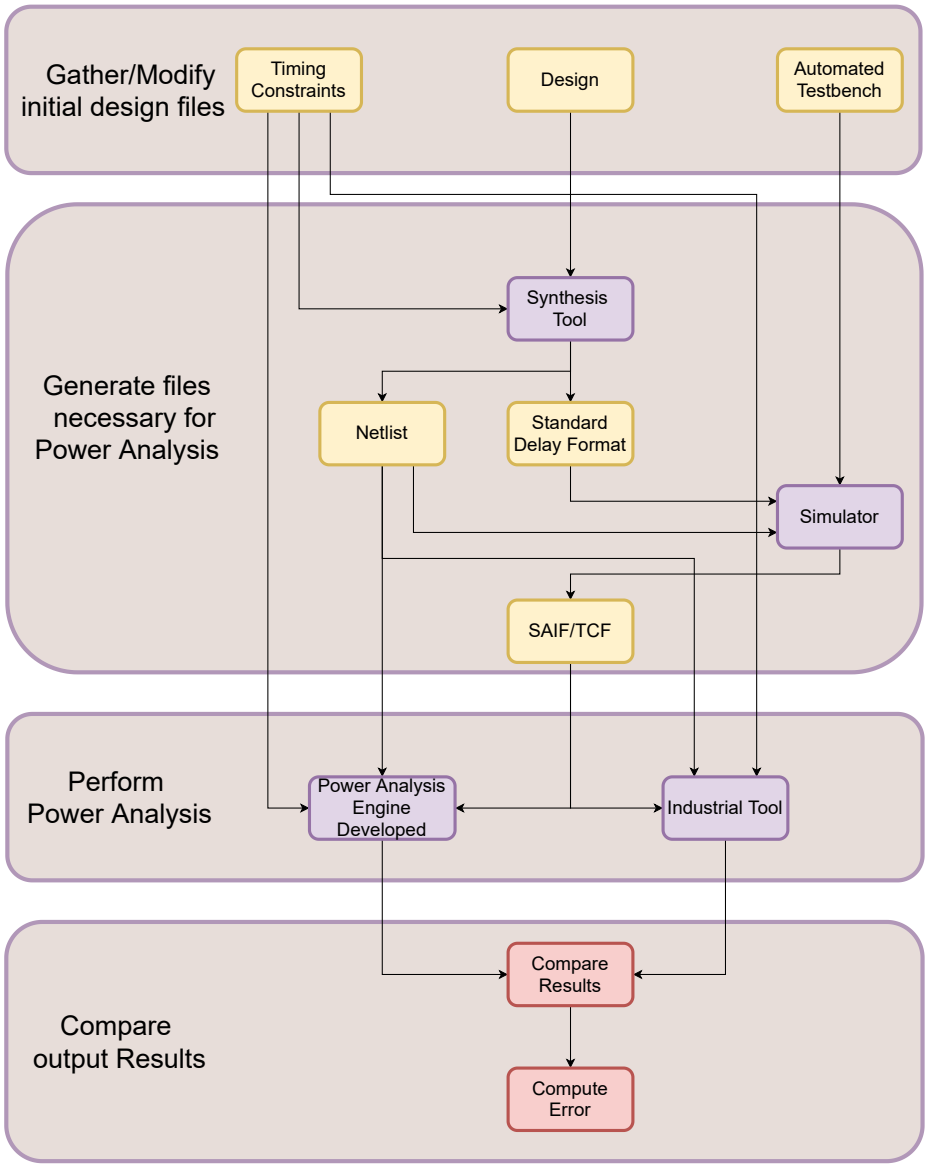

Figure 6.2: Experimental Flow Diagram

One note that has to be taken into account, is that each Design RTL used was written in Verilog Primitive format. The PDK used to evaluate the Power Analysis Engine is found in [19] which contains complex gates besides the standard (AND, OR, XOR, NOT). This

results in the Synthesis tool algorithm finding more efficient mappings in order to produce the same boolean function other than the Verilog Primitives found in ISCAS85 benchmarks. This results in a slightly different configuration of logic gates in the gate-level netlists compared to the Verilog Primitives RTL but the overall Primary Output functions remain the same.

Chapter 7

Experimental Results

In the following chapter, the Experimental Results will be presented. Overall, the Power Analysis Engine developed achieved an average error in Total Power of **-1.54%** out of which a significant percentage is due to Internal Power. The underlying reason is the difference in Timing Calculations between the two engines and not in the Power Calculations as it will be explained below.

(Unit: mW)	Industrial Tool				Power Analysis Engine			
Benchmarks	Internal	Switching	Leakage	Total Power	Internal	Switching	Leakage	Total Power
c17	0.002864	0.00131	5.87E-07	0.004175	0.002872	0.00131	5.87E-07	4.18E-03
c499	0.1908	0.1746	2.95E-05	0.3654	0.194113	0.174568	2.94582E-05	3.69E-01
c880	0.1644	0.1029	2.40E-05	0.2673	0.170281	0.102887	2.40426E-05	2.73E-01
c1908	0.2897	0.2016	2.91E-05	0.4914	0.294238	0.201638	2.90837E-05	4.96E-01
c3540	0.8392	0.6388	7.01E-05	1.478	0.866843	0.638816	7.00659E-05	1.51E+00
c6288	13.95	12.83	0.0001698	26.78	14.805708	12.825649	0.000169781	2.76E+01

Table 7.1: Result Comparison between Industrial Tool and Power Analysis Engine developed

Table 7.2 presented shows the comparison of error percentages between a state-of-the-art Industrial Tool and the Power Analysis Engine developed in this work. The error percentages for each benchmark are presented separately for internal power, switching power, leakage power, and total power. Note that the error was calculated using the equation 7.1:

$$Error = \frac{Golden - Measurement}{Golden} * 100\% \tag{7.1}$$

where $Golden$ is the result obtained from the Industrial Tool and $Measurement$ is the result obtained from the Power Analysis Engine developed. Note that a negative $Error$ shows

that the Power Analysis Engine is more pessimistic than the Industry Tool while a positive $Error$ indicates that the Power Analysis Engine is optimistic.

It can be observed that the errors in the power analysis engine are negative for most of the benchmarks, indicating that the power analysis engine is overestimating the Power Measurement. The total Power Error for all benchmarks is negative, indicating that the power analysis engine consistently overestimates the total Power dissipation.

(%)	Error Comparison			
Benchmarks	Internal Power	Switching Power	Leakage Power	Total Power
c17	-0.28%	0.00%	0.00%	-0.18%
c499	-1.74%	0.02%	0.01%	-0.91%
c880	-3.58%	0.01%	-0.01%	-2.20%
c1908	-1.57%	-0.02%	-0.01%	-0.92%
c3540	-3.29%	0.00%	0.01%	-1.88%
c6288	-6.13%	0.03%	0.01%	-3.18%
Average	**-2.76%**	**0.01 %**	**0.00 %**	**-1.54 %**

Table 7.2: Error Comparison between Industrial Tool and Power Analysis Engine developed

Regarding Switching Power and Static Leakage Power the Errors are insignificant and can also be present because of rounding errors because of different rounding accuracy between the two engines. The comparison in Internal Power results suggests that there are differences in the underlying models and algorithms used by the Industrial Tool and the Power Analysis Engine.

Upon further investigation, it was revealed that the source of error in Internal Power (and hence the error of the Total Power Dissipation) originated from the Timing Analysis Engine which is required to run to perform a full Static Timing Annotation of the entire chip. This is a prerequisite as a piece of essential information in the calculation of the Internal Power dissipation is the gate input transition times.

As stated in 5.1, the input transition time is used in order to perform a 2D interpolation. If the input transition time annotated by the 2 timing engines inside the Power Analysis engines is not the same, then the resulting interpolation will be different. Even if the transition times are the same, they may be outside of the LUT index values and then the extrapolating function

of each tool would provide a different result if the functions are not identical.

Percentage (%)	Percentage of Power Dissipation		
Benchmarks	Internal Power	Switching Power	Leakage Power
c17	68.666%	31.320%	0.014%
c499	52.646%	47.346%	0.008%
c880	62.330%	37.661%	0.009%
c1908	59.334%	40.661%	0.006%
c3540	57.570%	42.426%	0.005%
c6288	53.583%	46.417%	0.001%
Average	**59.021%**	**40.972%**	**0.007%**

Table 7.3: Percent part of Total Power Dissipation for each of the three types of Power calculated (Internal, Switching, Leakage)

Table 7.3 provides the percentage breakdown of total power dissipation for different benchmarks in a 130nm VLSI technology. Power dissipation is categorized into three types: internal power, switching power, and leakage power.

For each benchmark, the table shows the percentage distribution of power among these three components.

- Internal Power: The percentages range from 52.646% to 68.666% across the benchmarks, with an average of 59.021%. This indicates that a significant portion of the total power is consumed by the internal power sources in these circuits.

- Switching Power: The percentages range from 31.320% to 47.346%, with an average of 40.972%. It shows that switching power substantially contributes to these circuits' overall power consumption. It also shows that as the design gets bigger and more complex, the switching power will tend to be more dominant than the internal power.

- Leakage Power: The percentages for leakage power are very low, ranging from 0.001% to 0.014%, with an average of 0.007%. This indicates that the leakage power component has minimal impact on the total power dissipation in these benchmarks.

Overall, the table provides insights into the power distribution in VLSI circuits using

a 130nm technology. It highlights the dominance of internal power and switching power components, while the leakage power component remains relatively small.

Chapter 8

Conclusion and Future Work

The algorithms presented in this Thesis utilize a combination of gate-level and pin-level calculations to compute the **Internal Power**, **Switching Power** and **Static Leakage Power** dissipation of the design.

In the field of Power Analysis Engine Implementation for VLSI Digital Systems, there are several potential areas for future work. One area of focus could be on improving the accuracy and efficiency of the existing power analysis engine. This could involve exploring new power modeling techniques or optimizing the existing algorithms to reduce computational overhead and increase accuracy.

Another area for future work could be to extend the Power Analysis algorithms to handle more complex digital designs, such as those with asynchronous circuits or those that incorporate dynamic voltage and frequency scaling. These designs may require more advanced modeling techniques than the ones provided. Also, machine learning-based models could be integrated in order to accurately estimate the Power dissipation in various digital designs [20, 21].

In summary, the algorithms and techniques presented provide an accurate and efficient method for calculating the Power Dissipation of digital designs. Future work in this area can further improve the accuracy and applicability of these methods to even more complex designs.

Bibliography

[1] D. Sudhakar and R. Bhavani. Various power dissipation mechanisms and leakage current reduction techniques in deep-submicron technology. 2013.

[2] Ilkka Tuomi. The lives and death of moore's law. *First Monday*, 7, 04 2002.

[3] J. Bhasker and Rakesh Chadha. *Static Timing Analysis for Nanometer Designs: A Practical Approach*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[4] Synopsys Inc. Ccs timing. technical white paper, version 2.0,. 2006.

[5] Dimitrios Garyfallou, Stavros Simoglou, Nikolaos Sketopoulos, Charalampos Antoniadis, Christos P Sotiriou, Nestor Evmorfopoulos, and George Stamoulis. Gate delay estimation with library compatible current source models and effective capacitance. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(5):962–972, 2021.

[6] Tariq El Motassadeq. Ccs vs nldm comparison based on a complete automated correlation flow between primetime and hspice. In *2011 Saudi International Electronics, Communications and Photonics Conference (SIECPC)*, pages 1–5, 2011.

[7] Synopsis G. Mekhtarian. Composite current source (ccs) modeling technology backgrounder. 2005.

[8] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publishing Company, USA, 4th edition, 2010.

[9] Synopsys. Liberty user guides and reference manual suite. 2017.

[10] Cadence. Voltus ic power integrity solution user guide. 2021.

[11] G.G. Shahidi. Soi technology for the ghz era. In *2001 International Symposium on VLSI Technology, Systems, and Applications. Proceedings of Technical Papers (Cat. No.01TH8517)*, pages 11–14, 2001.

[12] D. Lee, W. Kwong, D. Blaauw, and D. Sylvester. Analysis and minimization techniques for total leakage considering gate oxide leakage. In *Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451)*, pages 175–180, 2003.

[13] H.J.M. Veendrick. Short-circuit dissipation of static cmos circuitry and its impact on the design of buffer circuits. *IEEE Journal of Solid-State Circuits*, 19(4):468–473, 1984.

[14] Sherif A. Tawfik and Volkan Kursun. Low power and high speed multi threshold voltage interface circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(5):638–645, 2009.

[15] Kimiyoshi Usami and Mark Horowitz. Clustered voltage scaling technique for low-power design. In *International Symposium on Low Power Electronics and Design*, 1995.

[16] Richard L Rudell. *Logic synthesis for VLSI design*. University of California, Berkeley, 1989.

[17] Ieee standard for standard delay format (sdf) for the electronic design process. *IEEE Std 1497-2001*, pages 1–80, 2001.

[18] M.C. Hansen, H. Yalcin, and J.P. Hayes. Unveiling the iscas-85 benchmarks: a case study in reverse engineering. *IEEE Design Test of Computers*, 16(3):72–80, 1999.

[19] IHP Leibniz-Institut für innovative Mikroelektronik. Ihp-open-pdk. `https://github.com/IHP-GmbH/IHP-Open-PDK`, 2023.

[20] Govindaraj Vellingiri and Arunadevi B. Machine learning based power estimation for cmos vlsi circuits. 07 2021.

[21] E. Poovannan and S. Karthik. Power prediction of vlsi circuits using machine learning. pages vol. 74, no.1, pp. 2161–2177, 2023.