To the Graduate Council:

I am submitting herewith a thesis written by Ashwin Balakrishnan entitled "An Experimental Study of the Accuracy of Multiple Power Estimation Methods." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Dr. Donald W. Bouldin

Major Professor

We have read this thesis and
recommend its acceptance:

Dr. Gregory D. Peterson

Dr. Chandra Tan

Accepted for the Council:

Anne Mayhew

Vice Chancellor and

Dean of Graduate Studies

(Original signatures are on file with official student records.)

An Experimental Study of the Accuracy of Multiple Power Estimation Methods

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Ashwin Balakrishnan

August, 2004

Acknowledgement

First and foremost, I would like to thank Dr. Don Bouldin for providing me an opportunity to do this project. His constant support, encouragement and super fast responses made me complete this project within the required time. I am extremely grateful to have an excellent mentor like him. Secondly, I would like to thank Dr.Gregory Peterson for all his guidance during some of the projects I have worked with him and support right from the day I landed here in the United States.

Special thanks to Dr. Chandra Tan and Dr. Fuat Karakaya for all their guidance and support not only during the course of this project but also during my entire course of study. This project would not have been completed in time without the initial background work done by them. Dr. Chandra Tan's comprehensive knowledge of scientific approaches and methodologies together with strong mathematical concepts always inspired me. I am thankful to my supervisor Shirley Moore at the Innovative Computing Laboratory of the Computer Science Department for giving me an opportunity to work with them during my course of study. I am also extremely grateful to the entire team at the University of Tennessee Telehealth Department where I was supported with a Graduate Research Assistantship position during the past year. A special thanks to my supervisors Dr. Susan Dimmick and Dr. Sam Burgiss for their invaluable guidance.

Last but not the least, I thank my family and friends for extending their support and encouragement.

Abstract

New and complex systems are being implemented using highly advanced Electronic Design Automation (EDA) tools. As the complexity increases, the dissipation of power has emerged as one of the very significant design constraints. Low power designs are not only used in small size applications like cell phones and handheld devices but also in high-performance computing applications.

Numerous tools have emerged in recent years to address this issue of power consumption and power optimization. With a vast number of these power measurement tools emerging, analyzing power consumed by digital circuits has not only become easier but also more effective methods are deployed to optimize digital circuits to dissipate less power.

This thesis involves using Synopsys power measurement tools together with the use of synthesis and extraction tools to determine power consumed by various macros at different levels of abstraction including the Register Transfer Level (RTL), the gate and the transistor level. A comparison of the power calculated using different net-lists from different extraction tools has also been done. In general, it can be concluded that as the level of abstraction goes down the accuracy of power measurement increases depending on the tool used.

Table of Contents

List of Tables

List of Figures

List of Acronyms

1. **EDA** - Electronic Design Automation
2. **VLSI** - Very Large-Scale Integration
3. **CMOS** - Complementary Metal Oxide Semiconductor
4. **ASIC** - Application-Specific Integrated Circuit
5. **FPGA** - Field Programmable Gate-Array
6. **SPICE** - Simulation Program With Integrated Circuit Emphasis
7. **RTL** - Register Transfer Level
8. **VCS** - Design Exchange Format
9. **HDL** - Hardware Design Language
10. **DEF** - Design Exchange Format
11. **LEF** - Library Exchange Format
12. **SDF** - Standard Delay Format
13. **GDS** - Graphical Data Stream
14. **DRC** - Design Rule Check
15. **LVS** - Layout Vs. Schematic
16. **SVRF** - Standard Verification Rule Format
17. **DSPF** - Delayed Standard Parasitic Format
18. **VSS** - VHDL System Simulator
19. **SAIF** - Switching Activity Interchange Format
20. **VCD** - Value Change Dump
21. **PIF** - *PrimePower* Intermediate Format

22. **DARPA** - Defense Advanced Research Projects Agency

23. **PDA** - Power Delay Area

24. **ICFB** - Integrated Circuit Front to Back

25. **TCL** - Tool Command Language

1 Introduction

1.1 Overview of the Problem

With the increasing usage of hand-held wireless devices and Internet appliances, there is a corresponding increased need for employing low-power design methodologies. One of the important requirements to know during a design process is how much power the circuit should dissipate considering its application. So after the designer writes the required code, keeping in mind all the specifications that have been given to him, a power calculation needs to be done to confirm if the design meets the required specification. This is done prior to sending the chip for fabrication. So it is extremely important to get accurate power values using power determining tools running them at certain input conditions.

Numerous EDA (Electronic Design Automation) tools have been developed to not only determine power but also help in power reduction. Some of these tools are targeted specifically for use in the power domain. The usage of these tools is classified depending on the layer of abstraction they are used in. The three main layers of abstraction include the RTL (Register Transfer Level), the gate and the transistor level. Though there are numerous tools that can be used at each of these levels, this thesis mainly concentrates on using Synopsys tools. The various power values that can be calculated using these tools is given in brief in the next section with detailed information following in the subsequent chapter.

1.2 Goals and contributions

The main goal of this thesis is to calculate the power of several macros which vary in complexity from a 500-transistor net-list to one containing more than 150,000 transistors. All these macros were developed by others as part of research at the Microelectronics Systems Laboratory at the University of Tennessee, Electrical and Computer Engineering Department.

For each of the macros, power will be calculated at various levels of abstraction using four EDA tools supplied by Synopsys: *Power Estimator*, *Power Compiler, PrimePower and NanoSim*. The purpose and functionality of each of tools will be discussed in the later chapters. One of the major contributions will be to calculate the power using the transistor-level simulator, *NanoSim* and compare it with the value obtained from *Prime Power* which operates at the gate level. A powerful and sophisticated extraction tool, *Calibre*, will be used to get the flat Spice-level net-list of each of the macros. Finally, a macro table will be formed indicating the power values of each macro at each level, together with the simulated time taken using each of the tools. Scripts will be developed to implement the various results. The following Figure 1.1 shows the design flow involved in the thesis in calculating the power values at different levels of abstraction. The results obtained at each level are tabulated.

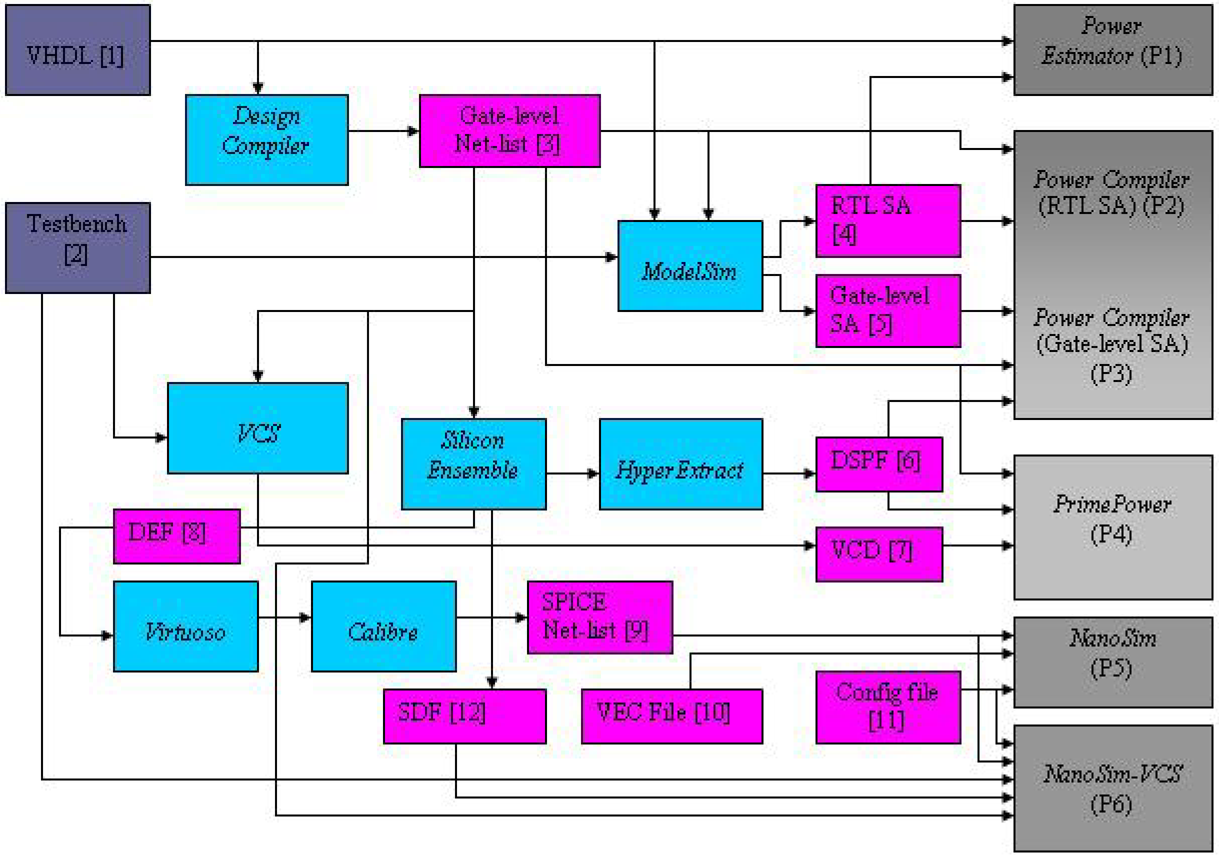


Figure 1.1 Design Methodology showing power calculation using different power tools

1.3 Outline of thesis

Chapter 2 mainly reviews the literature related to the various tools that have been used in this work. Chapter 3 discusses the implementation of the work at different levels of abstraction namely, RTL level, Gate level and Transistor level. Chapter 4 gives the results gathered from the previous chapter. Chapter 5 presents conclusions and discusses possible future work.

2 Background

2.1 Need for Low Power Design

In the early 1970's designing digital circuits for high speed and minimum area were the main design constraints. Most of the EDA tools were designed specifically to meet these criteria. Power consumption was also a part of the design process but not very visible. The reduction of area of digital circuits is not as big issue today because with new IC production techniques, many millions of transistors can be fit in a single IC. However, shrinking sizes of circuits have paved the way for reduced power consumption in order to have an extended battery life. Also in submicron technologies, there is a limitation on the proper functioning of circuits due to heat generated by power dissipation. Market forces are demanding low power for not only better life but also reliability, portability, performance, cost and time to market. This is very true in the field of personal computing devices, wireless communications systems, home entertainment systems, which are becoming popular now-a-days. Devices that are also used for high-performance computing particularly need to dissipate less power to function correctly and for a long period of time [1].

Keeping all these in mind, low power design has become one of the most important design parameters for VLSI (Very Large Scale Integration) systems.

2.1.1 Design Flow with and without Power

A top-down ordinary VLSI design approach is illustrated in Figure 2.1. The figure summarizes the flow of steps that are required to follow from a system level specification to the physical design. The approach was aimed at performance optimization and area minimization. However, introducing the third parameter of power dissipation made the designers to change the flow as you shown in the right-hand side of the Figure 2.1.

In each of the design levels are two important power factors, namely power optimization and power estimation. Power optimization is defined as the process of obtaining the best design knowing the design constraints and without violating design specifications. In order to meet the design and required goal, a power optimization technique unique to that level should be employed. Power estimation is defined as the process of calculating power and energy dissipated with a certain percentage of accuracy and at different phases of the design process. Power estimation techniques evaluate the effect of various optimizations and design modifications on power at different abstraction levels.

Generally a design performs a power optimization step first and then a power estimation step, but within a certain design level there is no specific design procedure. Each design level includes a large collection of low power techniques. Each may result in a significant reduction of power dissipation. However, a certain combination of low power techniques may lead to better results than another series of techniques.

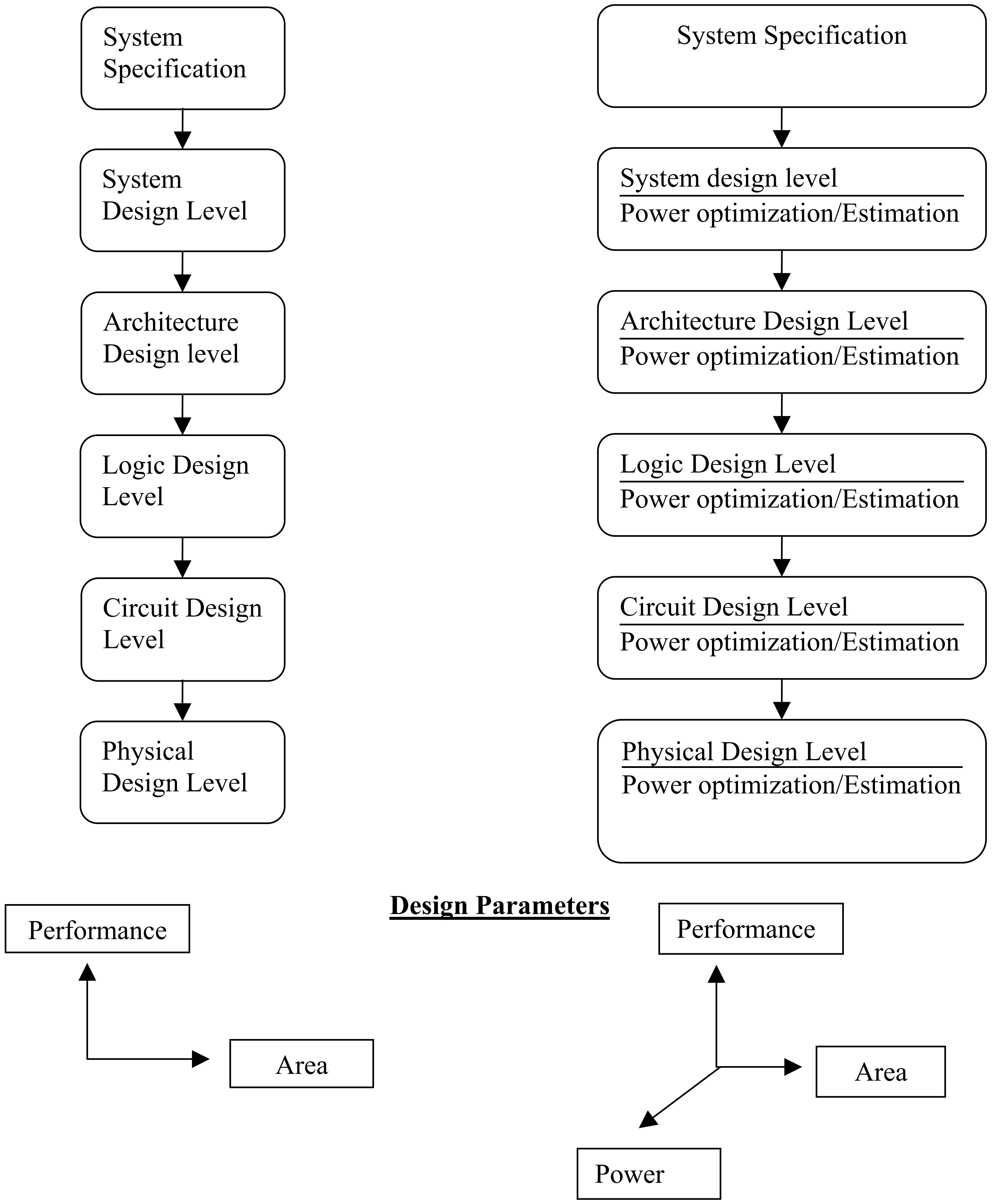


Figure 2.1 VLSI design flows

Generally, power is consumed when capacitors in the circuits are either charged or discharged due to switching activities. So at higher levels of a system this power dissipation is conserved by reducing the switching activities which is done by shutting down portions of the system when they are not needed. Large VLSI circuits contain different components like a processor, a functional unit and controllers. The idea of power reduction is to stop any of the components of the processor when they are not needed so that less power will be dissipated when the processor is operating [2].

2.2 Basic Concepts for Power

The power dissipation of digital CMOS circuits can be described by

$$\mathbf{P_{avg} = P_{dynamic} + P_{short\text{-}circuit} + P_{leakage} + P_{static}}$$

$\mathbf{P_{avg}}$ is the average power dissipation, $\mathbf{P_{dynamic}}$ is the dynamic power dissipation due to switching of transistors**,** $\mathbf{P_{short\text{-}circuit}}$ is the short-circuit current power dissipation when there is a direct current path from power supply down to ground , $\mathbf{P_{leakage}}$ is the power dissipation due to leakage currents, $\mathbf{P_{static}}$ and is the static power dissipation [2].

2.2.1 Static Power

Static power is the power dissipated by a gate when it is not switching that is, when it is inactive or static. Ideally, CMOS (Complementary Metal Oxide Semiconductor) circuits dissipate no static (DC) power since in the steady state there is no direct path from V_{dd} to ground. This scenario can never be realized in practice, since in reality the MOS transistor is not a perfect switch. There will always be leakage currents, subthreshold currents, and substrate injection currents, which give rise to the static component of power dissipation. The largest percentage of static power results from source-to-drain subthreshold voltage, which is caused by reduced threshold voltages that prevent the gate from completely turning off [2].

2.2.2 Dynamic Power

Dynamic power is the power dissipated when the circuit is active. A circuit is active anytime the voltage on net changes due to some stimulus applied to the circuit. In other words, dynamic power dissipation is caused by the charging. Because voltage on an input net can change without necessarily resulting in logic transition in the output, dynamic power can be dissipated even when an output net doesn't change its logic state. This component of dynamic power dissipation is the result of charging and discharging parasitic capacitances in the circuit [2].

Dynamic power of a circuit is composed of

a) Switching power

b) Internal power

2.2.2.1 Switching power

The switching power of a driving cell is the power dissipated by the charging and discharging of the load capacitance at the output of the cell. The total load capacitance at the output of a driving cell is the sum of the net and gate capacitances on the driving output. The charging and discharging are result of logic transitions. Switching power increases as logic transitions increase. Therefore, the switching power of a cell is a function of both the total load capacitance at the cell output and the rate of logic transitions. Switching power comprises 70-90 percent of the power dissipation of an active CMOS circuit [2].

2.2.2.2 Internal power

Internal power is any power dissipated within the boundary of a cell. During switching, a circuit dissipates internal power by the charging or discharging of any existing capacitances internal to the cell. Internal power includes power dissipated by a momentary short circuit between the P and N transistors of a gate, called short-circuit power. In most simple library cells, internal power is due mostly to short-circuit power. Library developers can model internal power by using the internal power library group [2].

2.2.3 Short-Circuit Power

The short-circuit power consumption, **P $_{\textbf{short-circuit,}}$** is caused by the current flow through the direct path existing between the power supply and the ground during the transition phase.

2.2.4 Leakage Power

The nMOS and PMOS transistors used in a CMOS logic circuit commonly have non-zero reverse leakage and sub-threshold currents. These currents can contribute to the total power dissipation even when the transistors are not performing any switching action. The leakage power dissipation, **P $_{\textbf{leakage}}$** is caused by two types of leakage currents

a) Reverse-bias diode leakage current

b) Subthreshold current through a turned-off transistor channel [23].

2.3 Tools Used

There has been a variety of tools involved in this thesis. Even though, this thesis is all about power calculations of macros which are done using tools; there are other tools that have been used prior to the usage of power tools to give the required input to the power tools. More emphasis is given to these tools that are mainly involved in power estimation. The usage of tools has been classified as Power tools and Non-Power tools.

2.3.1 Non-Power Tools

Non-power tools include Simulation tools, Synthesis tools, Layout tools, Extraction tools and Waveform viewers.

The tools that are discussed in this chapter are some of the non-power tools involved in the entire design flow. A short description of each of these tools along with their working flow is given in this chapter to understand their functionality. The subsequent chapter discusses each of the power tools in detailed manner as most of the thesis involves the use of these power tools. The following chapter also discusses the design flow from code writing to spice net-list simulation, clearly explaining the usage of these tools at the respective level.

2.3.1.1 Simulation Tool

Initially, to start with the Verilog or VHDL code for a particular design is written and tested. Simulation is done using Mentor's *ModelSim* for both VHDL and Verilog or other Verilog simulators. *ModelSim* is a simulation and a debugging tool for VHDL, Verilog, and other mixed-language designs from Mentor Graphics [21]. The basic simulation flow is as shown in Figure 2.2. To start with a working library is created and the code is compiled using

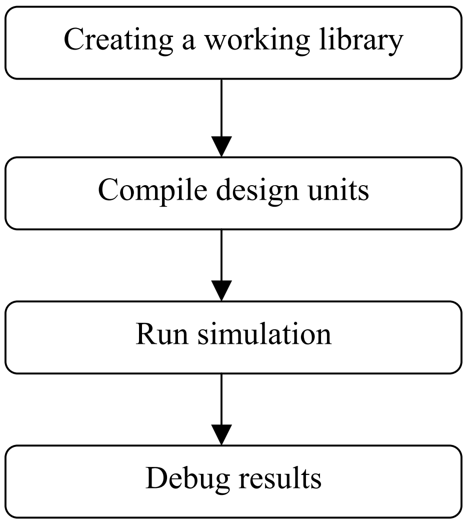


Figure 2.2 *ModelSim* simulation flow

the commands depending upon whether the code is VHDL or Verilog.

Verilog Compiled Simulator (VCS) [22] from Synopsys is a high-performance, high-capacity Verilog simulator that incorporates advanced high-level abstraction, verification into an open platform. The basic work flow for VCS consists of two basic steps:

a) Compiling source files into executable binary files

b) Running the executable binary file

This two step approach simulates the design faster and uses less memory than other interpretive simulators. The basic design flow is given in Figure 2.3.

Verilog-XL [23] simulator from Cadence provides a powerful environment for designing and verifying the functional building blocks of complex ASICs (Application

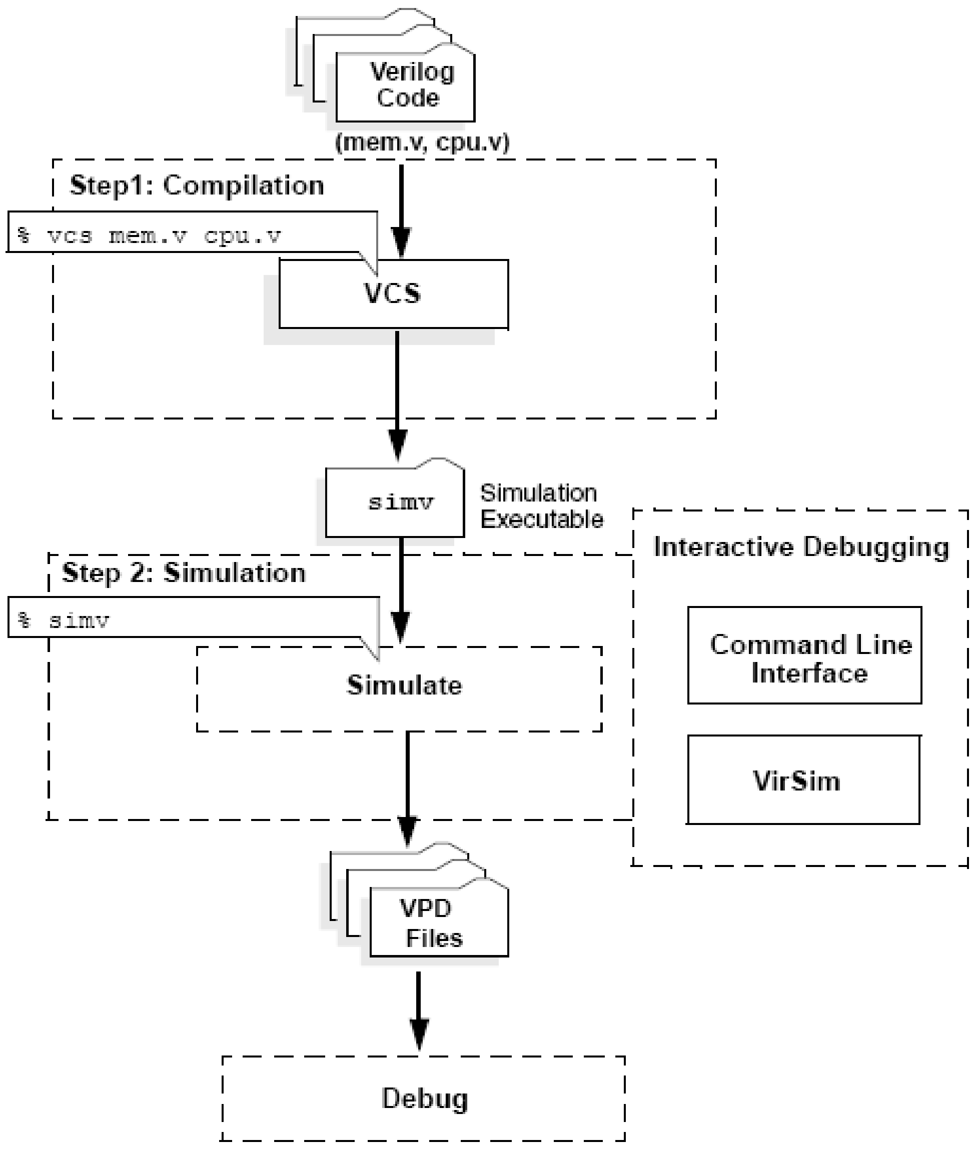


Figure 2.3 VCS work flow

Specific Integrated Circuits) and SoC (Systems on Chip). This digital simulator not only allows to check the functional integrity of a design but also helps in finding the design flaw. This simulator processes models which are descriptions of design that are developed using Verilog. The design flow starts with a design idea and ends with an verified design as shown in Figure 2.4.

Normally, *Verilog-XL* compiles all the modules that are defined in a source text file. Those that are not instantiated in the source file become top-level modules. Creating libraries avoids this unnecessary origination of top-level modules, saving compile time and memory. So whenever *Verilog-XL* cannot find the module in the design description it searches the libraries associated with the design description for the definition.

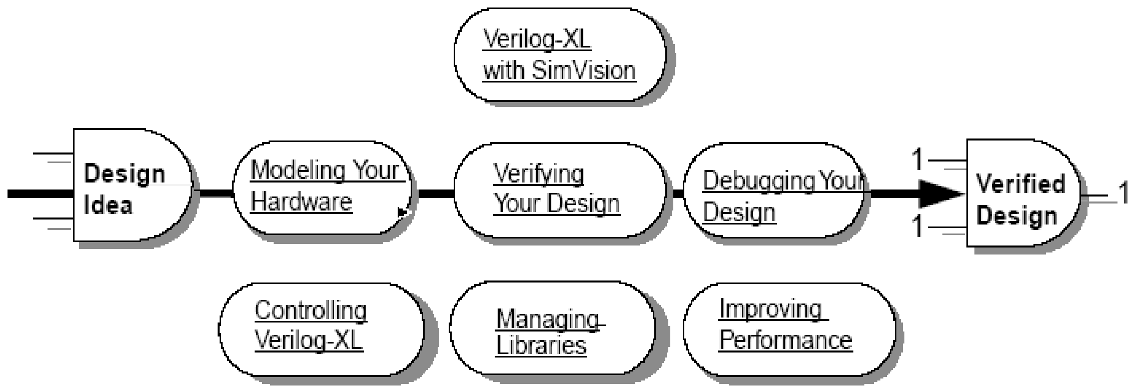


Figure 2.4	***Verilog-XL* design flow**

2.3.1.2 Synthesis Tool

Design Compiler [24] is the core of the Synopsys synthesis software products. It comprises tools that synthesize HDL designs into optimized technology-dependent, gate-level designs. It supports a wide range of flat and hierarchical design styles and can optimize both combinational and sequential designs for speed, area, and power.

Design Compiler reads and writes design files in all the standard EDA formats, including Synopsys internal database (.db) and equation (.eqn) formats. In addition, *Design Compiler* provides links to EDA tools, such as place and route tools, and to post-layout resynthesis techniques, such as in-place optimization. *Design Compiler* products include *DC Professional*, *DC Expert*, *DFT Compiler, DC Ultra, and DC Ultra Plus*.

The basic *Design Compiler* synthesis process is given in Figure 2.5.

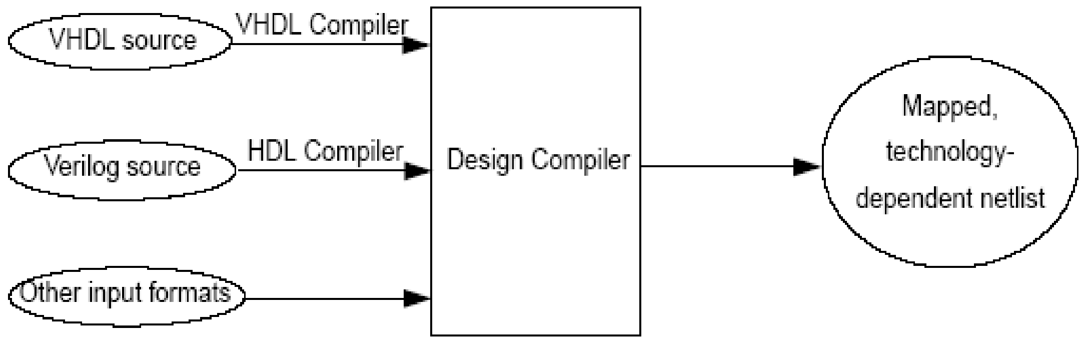


Figure 2.5 ***Design Compiler*** **synthesis process**

These products help in producing fast, area efficient ASIC designs by employing user-defined gate arrays, FPGA (Field Programmable Gate Arrays) or compiled libraries. The tools also help in exploring design tradeoffs involving design constraints such as timing, area and power under various loading, temperature and voltage conditions.

The *Design Compiler* is a powerful tool that other products can be run inside its environment using specific options. Some of the products that can be accessed are HDL compiler, Automated chip synthesis, *FPGA compiler*, *Behavioral compiler* and *Power Compiler*.

HDL compiler reads and writes Verilog or VHDL design files. The Verilog or VHDL compiler reads the HDL files and performs translation and architectural optimization of the designs. The appropriate HDL compiler is automatically called by *Design Compiler* when it reads an HDL design file.

2.3.1.3 Place and Route Tool

Silicon Ensemble [24] is Cadence's Automatic Place and Route tool. *Silicon Ensemble* can perform the physical placement and routing of a net-list of standard cells. Its uses compaction algorithms for giving the most effective way of placing and routing cells. The output from *Silicon Ensemble* is a DEF (Design Exchange Format) file which can be later imported into *Virtuoso* to get the layout.

Silicon Ensemble is a stand alone tool that needs some intermediate files, that describe the various aspects of the design, in an appropriate format. These files are called LEF (Library Exchange Format) which describes the rules on how wires can be drawn by the router and what the available cells and blocks look like. The other important file is DEF which is an output file from *Silicon Ensemble* that contains information on how the cells are connected along with the coordinates of where the blocks are placed with their wire informations. Another important output file is SDF (Standard Delay format) file which is used to get timing information of the design. The basic flow from *Silicon Ensemble* is as shown in Figure 2.6.

2.3.1.4 Layout Tool

Virtuoso Layout Editor [25] from Cadence is a custom physical layout tool. It supports the physical implementation of custom digital, mixed-signal, and analog designs at the device, cell, and block levels. Apart from GDS (Graphical Data Stream) file from *Silicon Ensemble*, a DEF file is also created and imported into *Virtuoso* to check that the final layout is according to the design rules. The gate level net-list from *Design Compiler* can also be imported into *Virtuoso* to get the schematic for the VHDL or Verilog code written. LVS (Layout versus Schematic) is performed to prove that the layout created is only for that particular schematic.

Simulations using *Verilog-XL* and *Spectre* can be performed from this environment to check if the circuit is functionally correct.

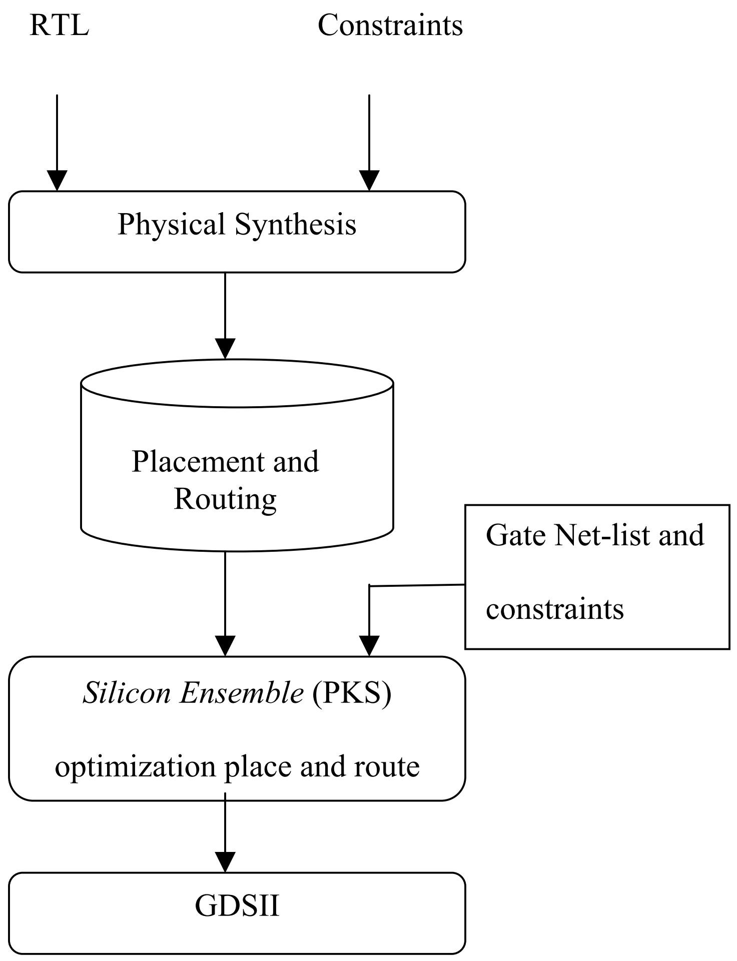

RTL
Constraints
Physical Synthesis
Placement and Routing
Gate Net-list and constraints
Silicon Ensemble (PKS) optimization place and route
GDSII

Figure 2.6 ***Silicon Ensemble* work flow**

2.3.1.5 Extraction Tool

The Extraction tool used in this thesis is Mentor Graphic's *Calibre* [37]. There are two versions of this tool; one is used for getting hierarchical parasitic extraction by *xCalibre*-H tool and other one is used for a flat net-list with RC parasitic extraction. Mostly *xCalibre* PX-C/PX- RC tool has been used to get flat net-list with RC extraction.

The *xCalibre* PX-X/PX-RC tool performs parasitic extraction on nets in transistor and cell-level IC designs. These designs produce either lumped or distributed net models suitable for resistance or capacitance net models. The tool creates net models for the design in the form of net-list. In other terms, in processing a design, the tool flattens the design's hierarchy, and outputs a net-list with capacitance and resistance values or ASCII reports. The tool performs both layout-based and source-based extraction. It is mandatory to supply the tool with layout database, whereas the source database is optional. The following Figure 2.7 shows the extraction flow.

The sequence involved in the process is to first prepare the design data, then prepare the SVRF (Standard Verification Rule Format) file, and then invoke the tool. Once the net-list and report files are generated, they are analyzed. One of the major advantages in using *xCalibre* is that it does DRC checking very fast compared to other extraction tools which largely helps in reducing time during a completion of layout verification.

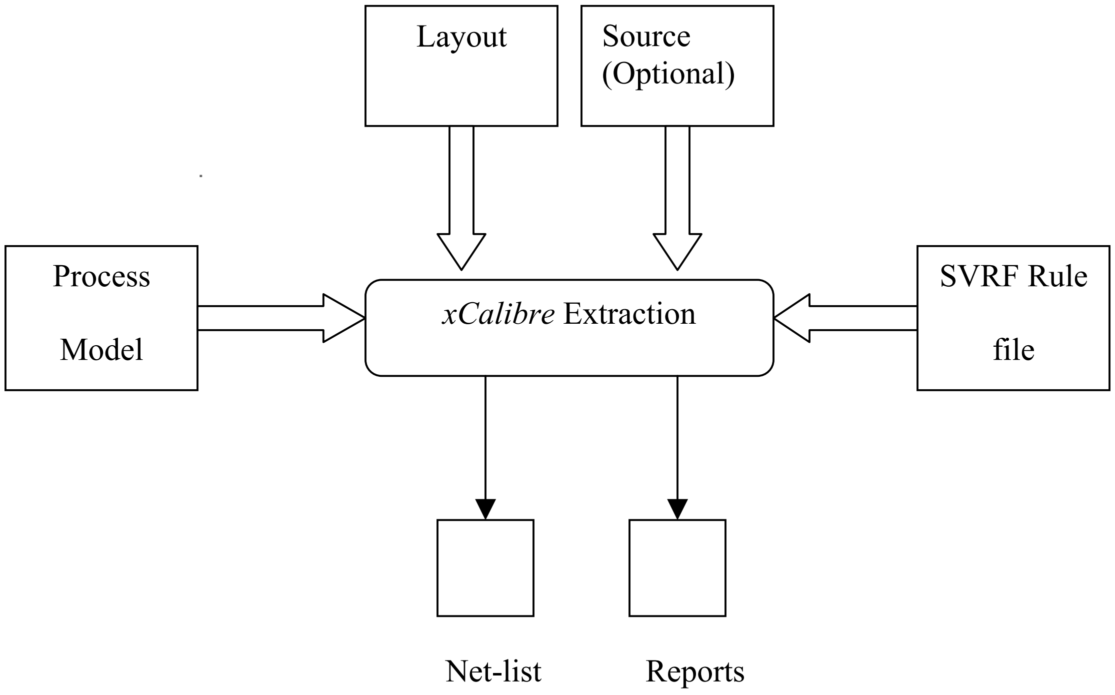


Figure 2.7 *xCalibre* Extraction flow

The tool produced two outputs:

a) Net-list

b) Reports

The outputs are controlled by the operations and statements in the SVRF file. The net-list contains the parasitic net models. These models are an extracted net's parasitic capacitance and resistance values. The tool produces one parasitic model per net. Using the application two types of net-list can be produced.

a) HSPICE

b) DSPF (Delayed Standard Parasitic Format)

The HPSICE or DSPF net-list are produced in one of the three modes:

- Simple – Net-list without parasitic data
- Lumped (PX-C) – Net-list with lumped C data
- Distributed (PX-RC) – Net-list with distributed RC data

The tool also produces ASCII reports which list net capacitances, resistances, and delays. They contain only formatted parasitic data without net-list information or syntax. The following are the output report modes:

- ASCII Distributed
- ASCII Lumped
- SDF

in accordance with the instructions specified in the rule file. Additionally, the tool can produce SDF suitable for analysis of pin-to-pin timing delays.

2.3.1.6 Waveform Viewer

CosmoScope [38] from Synopsys is a graphical waveform analyzer tool that allows to view and analyze results in the form of waveforms displayed on graphs, or as values displayed as lists. Some of the tools that are included with *CosmoScope* are

a) Signal Manager – which is used to open the files to plot them

b) Measurement tool – Used for measurements that can be applied to a waveform

c) Waveform calculator – calculator that interacts graphically with application

SimWave [39] from Synopsys is an interactive waveform display capable of displaying digital, analog, floating-point, enumerated, and string type signals. It can be used as a post-processing tool or an interactive tool capable of showing simulation results while a simulation is still running. Users can open multiple windows for viewing unlimited signals in each of those windows. Some of the features of *SimWave* are

a) Circuit development is accelerated using fast, clear displays

b) Complex relationships can be understood easily due to waveforms

c) Circuit operation can be documented

d) Has got better search so that debugging is easier

2.3.2 Power Tools

This thesis involves the usage of Synopsys power tools. The power products are tools that comprise a complete methodology for low-power design. Synopsys power tools offer power analysis and optimization throughout the design cycle, from RTL to the gate level. Analyzing power early in the design cycle can significantly affect the quality of the design. Improvements made to the design while it is at RTL level can get even better results eventually. Not only these power tools do accurate measurements but also can help in calculating power quicker.

Power consumption is calculated at three levels of abstraction. The tools used at these levels are

a) RTL Level - *RTL Power Estimator*

b) Gate Level – *Power Compiler* (based on switching activity), *PrimePower*

c) Transistor Level – *NanoSim*

2.3.2.1 DesignPower

DesignPower [29] is a power analysis tool that analyzes the design for switching power, internal cell power, and leakage power. The *DesignPower* tool analyzes the power of a gate-level design. It requires a gate-level net-list and some form of switching activity for the net-list. It computes average power consumption based on activity of nets in the design. *DesignPower* allows capturing the switching activity of primary inputs, primary outputs, and outputs of sequential elements during simulation.

Power analysis using switching activity from RTL simulation provides a much faster turnaround than analysis through switching activity from gate-level simulation. *DesignPowe*r supports interfaces to Synopsys *VHDL System Simulator* (VSS), *VCS* and *Verilog-XL* simulator.

2.3.2.2 RTL *Power Estimator*

The *RTL Power Estimator* [29] enables one to obtain design power estimates early in the design process. Its presynthesis estimation capabilities analyze power consumption at the RTL. It covers both synthesizable and instantiated parts of the circuit. One of the advantages is that power information can be obtained using limited input. Since power is calculated at a very early stage in the design flow it helps in estimating packaging and battery requirements. It is also used to identify the hotspots in a large, complex design.

2.3.2.3 ***Power Compiler***

Power Compiler [29] is an add-on product to *Design Compiler*. The *Power Compiler* tool optimizes the design for power. Working in conjunction with the *Design Compiler* tool, *Power Compiler* provides simultaneous optimization for timing, power and area. In addition to the standard inputs to synthesis (RTL or gate-level net-list, technology library, design constraints, and parasitics), *Power Compiler* uses two other inputs: Switching activity of design elements and power constraints. It contains all the analysis capabilities of *DesignPower*.

Power Compiler uses the same power analysis engine as *DesignPower*. This allows *Power Compiler* to the use the same switching activity for optimization that *DesignPower* uses for analysis. It accepts either user-defined switching activity, switching activity from simulation, or a combination of both. It provides RTL clock gating and optimizes the circuit based on circuit activity, capacitance, and transition times. *Power Compiler* cannot only be used as a standalone product but also can be used in coordination with *Design Compiler*, *Module Compiler*, *Physical Compiler* and *Floor plan Manager*.

2.3.2.3.1 *Power Compiler* Methodology

Power Compiler is used at RTL and Gate level to calculate power and do power optimization depending on the need. At each level of abstraction, simulation, analysis and optimization can be performed to refine the design before moving to the next lower level. Simulation and the resultant switching activity gives the analysis and optimization the necessary information to refine the design before going to next lower level of abstraction. The higher the level of design abstraction, the greater the power savings can be achieved. The following Figure 2.8 describes the power flow at each of the abstraction level. Figure 2.9 shows power flow from RTL to Gate level.

Cell internal power and net toggling directly affect dynamic power of a design. To report or optimize power, *Power Compiler* requires toggle information for the design. This toggle information is called Switching Activity.

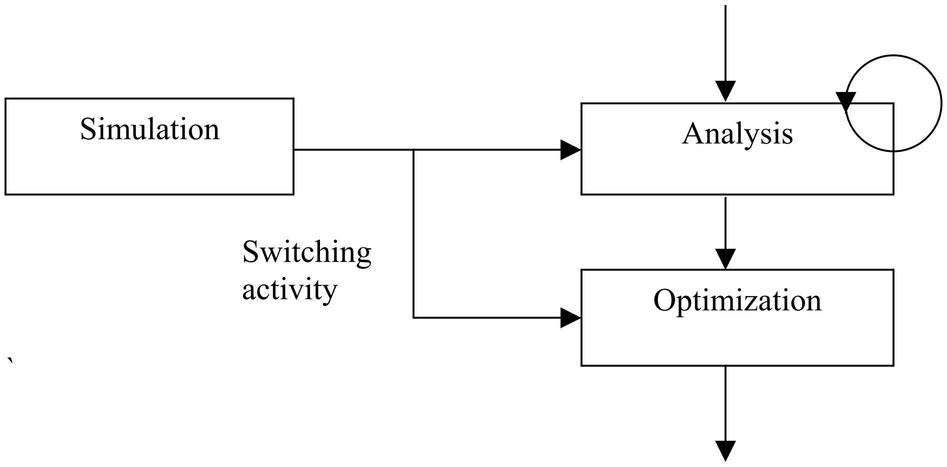


Figure 2.8 Power flow at each of the abstraction level

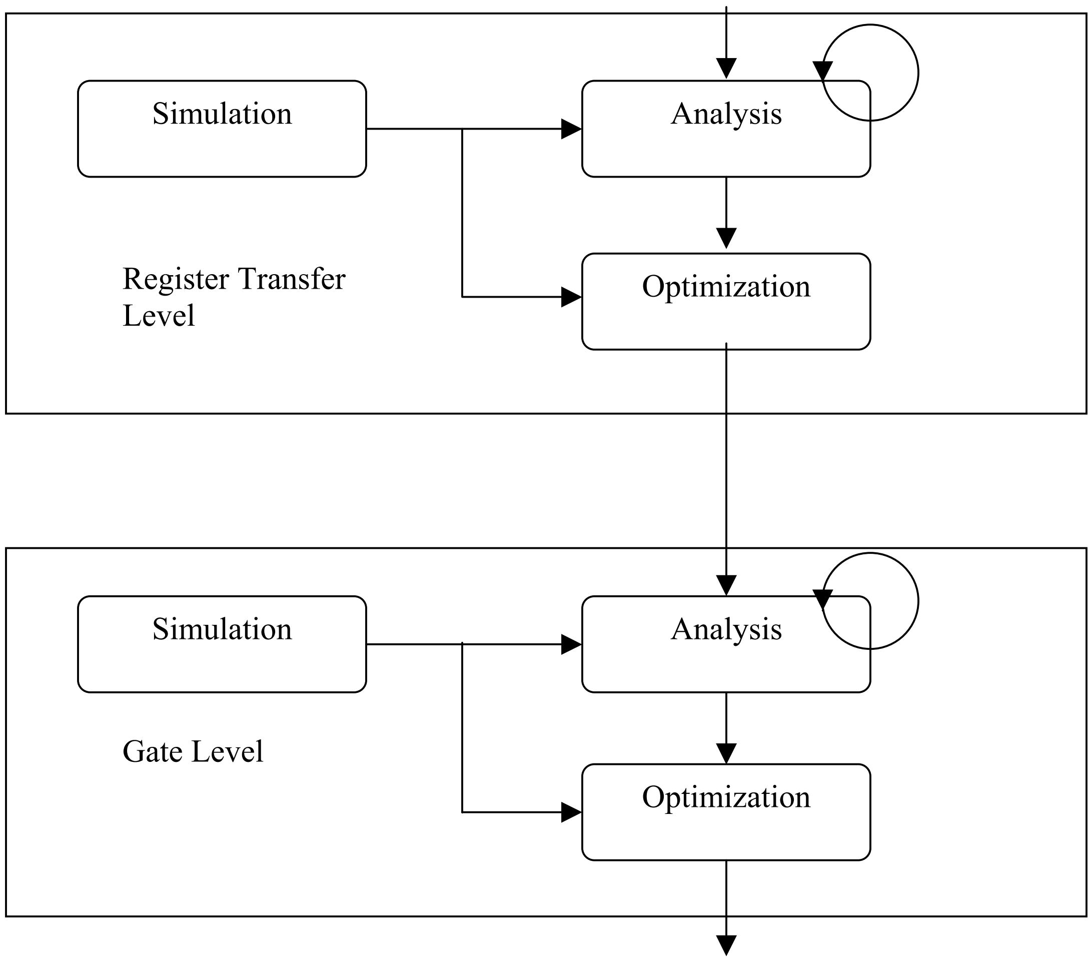


Figure 2.9 Power flow from RTL to Gate level

Power Compiler models switching activity in terms of static probability and toggle rate. Static probability is the probability that a signal is at a certain logic state and is expressed as a number between 0 and 1. It is calculated during simulation of the design by comparing the time of a signal at a certain logic state to the total time of the simulation. Toggle rate is the number of logic-0-to-logic-1 and logic-1-to-logic-0 transitions of a design object per unit of time.

The following Figure 2.10 shows the methodology of power calculation using the combination of *Power Compiler* and *Design Compiler*. The flow of data between the different steps and tools used are also shown. Before starting to calculate power using *Power Compiler* the desired gate-level net-list of the design should be first generated. The power methodology starts with the RTL design and finishes with a power-optimized gate-level net-list. Ultimately, *Power Compiler* is used to calculate power using the gate-level net-list produced by the *Design Compiler* or power-optimized gate net-list produced by *Power Compiler* itself. As seen in the figure most of the processes that take place are using *Design Compiler*, but the simulation process that is shown is outside *Design Compiler* tool and is done as part of power calculation. The main purpose of simulation is to generate information about the switching activity of the design and create a file called Back-annotation. This file can contain switching activity from RTL simulation or gate-level simulation. Initially, the RTL design is given to the HDL compiler to create a technology-independent format called as GTECH design. This is as a result of analyzing and elaborating the design by HDL compiler. This formatted design is given as an input to *Design Compiler*. Before it is

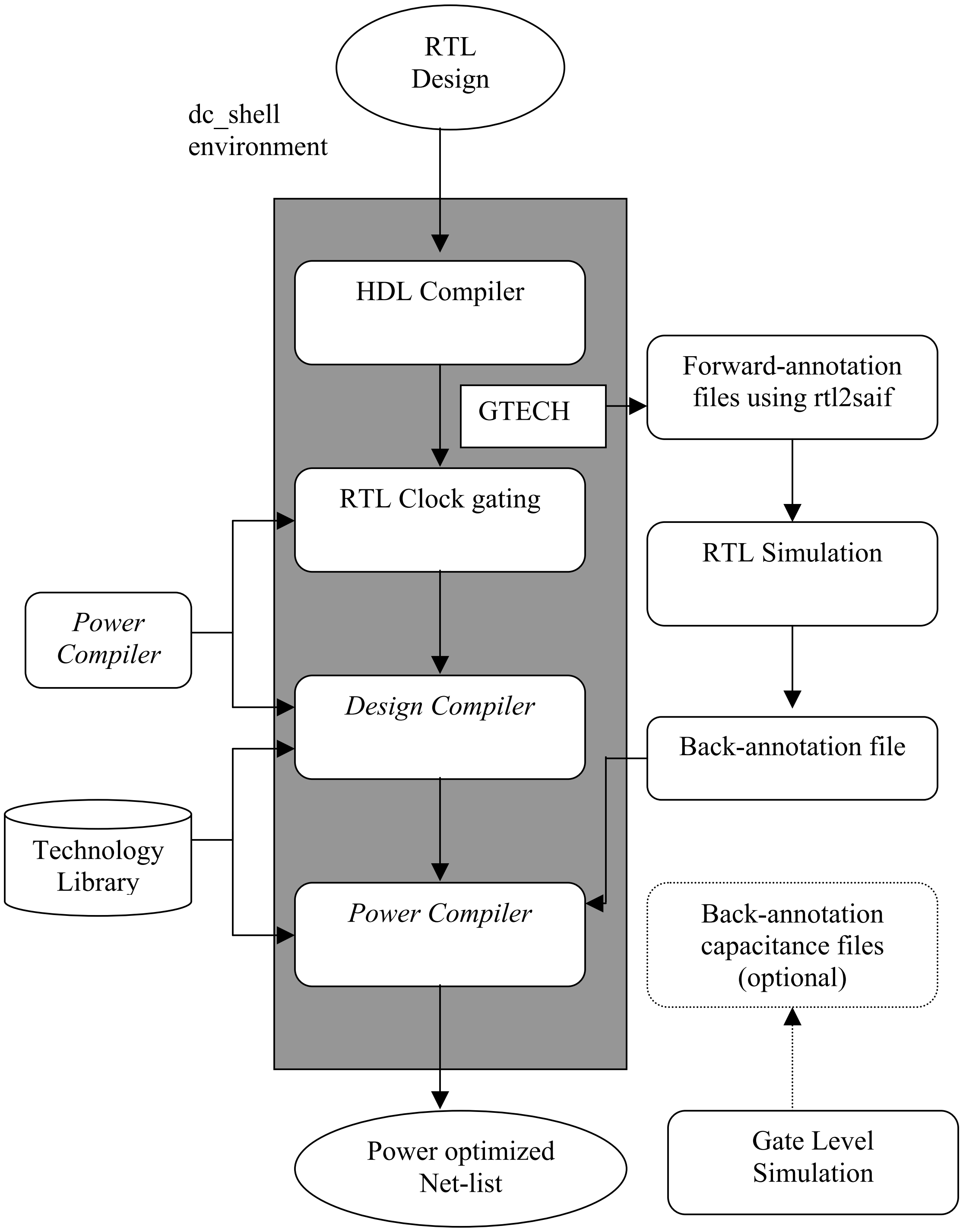


Figure 2.10 Power methodology in *Power Compiler*

compiled by the *Design Compiler*, "**rtl2saif**" command is used to create forward-annotation file which is later used for simulation. The formatted design GTECH is later given as input to *Design Compiler* which produces an output which is given to *Power Compiler*.

The Forward-annotation SAIF file is given as an input to do RTL simulation which gives a back-annotation SAIF file which is used by *Power Compiler*. This forward annotated file contains directives that determine which design elements to be traced during simulation. Gate-level simulation can also use a library forward-annotation file. This forward-annotation file used for gate level simulation has different information compared to RTL forward-annotation file. This file contains information from the technology library about cells with state and path-dependent power models. "**Lib2saif**" command is used to get this forward-annotation file.

During power analysis, *Power Compiler* uses the annotated switching activity to evaluate the power consumption of the design. During power optimization, *Power Compiler* uses the annotated switching activity to make decisions about the design.

2.3.2.3.1.1 Power Optimization

Power optimization achieved at higher levels of abstraction (RTL) has an impact in reducing power in the final gate-level optimization. *Power Compiler* performs clock gating when the design is elaborated using "**-gate_clock**" option. Design generally has synchronous load-enable registers. These registers are formed using feedback loops by *Design Compiler*.

These registers maintain the same logic value through multiple cycles and unnecessarily use power. When the “**-gate_clock**” option is used HDL compiler introduces gates in the clock network before *Design Compiler* does its processing. During the next step, *Design Compiler* checks the gated clock introduced by HDL compiler and uses simple registers without synchronous load-enabled functionality thus saving power. RTL clock gating is achieved without affecting timing or area of the design.

At the gate level, *Design Compiler* and *Power Compiler* are used to create gate-level net-list optimized for power. Once the RTL clock gating is done, the next output is the gate-level net-list which will be optimized for power. First constraints are set for timing and area. Then the design is compiled using the *Design Compiler*. This creates a gate-level design on which the switching activity can be annotated using the back-annotation file. The back-annotation file is read into *Power Compiler* using “**read_saif**” command. After this power constraints are set to trigger power optimization by *Power Compiler*. Then the design is compiled using *Power Compiler*. Using the switching activity and power constraints, *Power Compiler* produces a gate-level net-list which is optimized for timing, power and area.

Switching activity from RTL simulation provides good power optimization results. However, switching activity from gate level simulation provides much more accurate analysis and optimization. The power analysis of the gate-level design can be done at various points in the entire methodology. Once annotating the switching activity from the back-annotation file, power can be analyzed before compiling using

Power Compiler. This is done before power optimization. Once doing power optimization the power values can be compared. "**report_power**" is the command used to get detailed power results.

2.3.2.4 *PrimePower*

One of the tools apart from *Power Compiler* that is used to calculate gate level power is *PrimePower* [31]. *PrimePower* is a dynamic gate-level simulation and analysis tool that accurately analyzes power dissipation of cell-based designs. *PrimePower* achieves a high level of accuracy with precise modeling of power dissipation. Because of this feature it is a very useful tool for circuit designers who develop products that are power-critical such as portable computing and telecommunications. It also provides a very comprehensive power analysis reports which can be seen interactively by the user and hence can analyze better. *PrimePower* also works with standard simulators like *VCS*.

2.3.2.4.1 *PrimePower* Methodology

The methodology involved using *PrimePower* consists of two phases;

a) HDL Simulation

b) Power profiling

During the first phase which is HDL simulation, *PrimePower* interprets switching activity in the VCD (Verilog Change Dump) or PIF (*PrimePower*

Intermediate Format). Most of the simulators generate files in the VCD format using "**$dumpvars ( )**" command. The event information given by the VCD file is directly read by *PrimePower* during its input. VCD+ formatted file is generated using VCS simulator. The event information can be converted to VCD within *PrimePower* using "**vpd2vcd**" command.

The second phase involves power profiling gives a detailed power report. To generate the power profile, design connectivity, design switching activity and pin-to-pin delay information are required. The ASCII based power reports, is used to view the power profile at the cell, block, or chip levels. The design activity is available in the VCD data file. The design connectivity is provided in the form of Verilog or VHDL net-list files, and pin-to-pin timing is derived from the Synopsys libraries and updated when parsing the VCD file. Based on these inputs and any additional net capacitance, *PrimePower* builds the power profile. The inputs given to the first phase i.e. HDL simulation are SDF file, Test bench file and Verilog library. This results in the generation of VCD file which is given to *PrimePower* along with Synopsys db file and Net-list. Depending on the mode of operation of *PrimePower* which are Batch mode, Interactive batch mode, or GUI mode power reports can be obtained. The GUI provides templates to set up the *PrimePower* commands, and a suite of analysis tools to view the results graphically. The command line mode is run using the command "**pp_shell**". The "**pp_shell**" also provides script execution environment based on TCL (Tool Command Language). Interactive mode produces a set of text-formatted power reports based on data that is entered interactively at command-line prompt. The analysis tool of *PrimePower* is based on *PrimeTime*.

Power analysis is performed using *PrimePower* to determine the power consumption of the chip based on the switching activity. *PrimePower* is event based, so for every event it determines the supply current and leakage current dissipated given the states and dynamic conditions. *PrimePower* calculates both static and dynamic power. Static power is often referred to as Leakage power. *PrimePower* can also be used to determine Glitch power. If two toggles are very close to each other, and the time interval of two toggles is less than the rise and fall transition time of that particular pin, then these two toggles form a glitch. The following Figure 2.11 gives the methodology involved in *PrimePower*.

2.3.2.5 ***NanoSim***

NanoSim [26] is one of the most powerful power tools by Synopsys. *NanoSim* is a dynamic transistor level simulator. It is a dynamic power/timing analysis tool. It is a combination of two previous simulators, *PowerMill* and *TimeMill*. Some of the important features of using *NanoSim* are

a) It is faster than Spice

b) It can handle full chip net-list with extracted parasitic

c) It can do concurrent simulation with mixed net-list types like spice, Verilog, EPIC, Spectre, EDIF and CSPF.

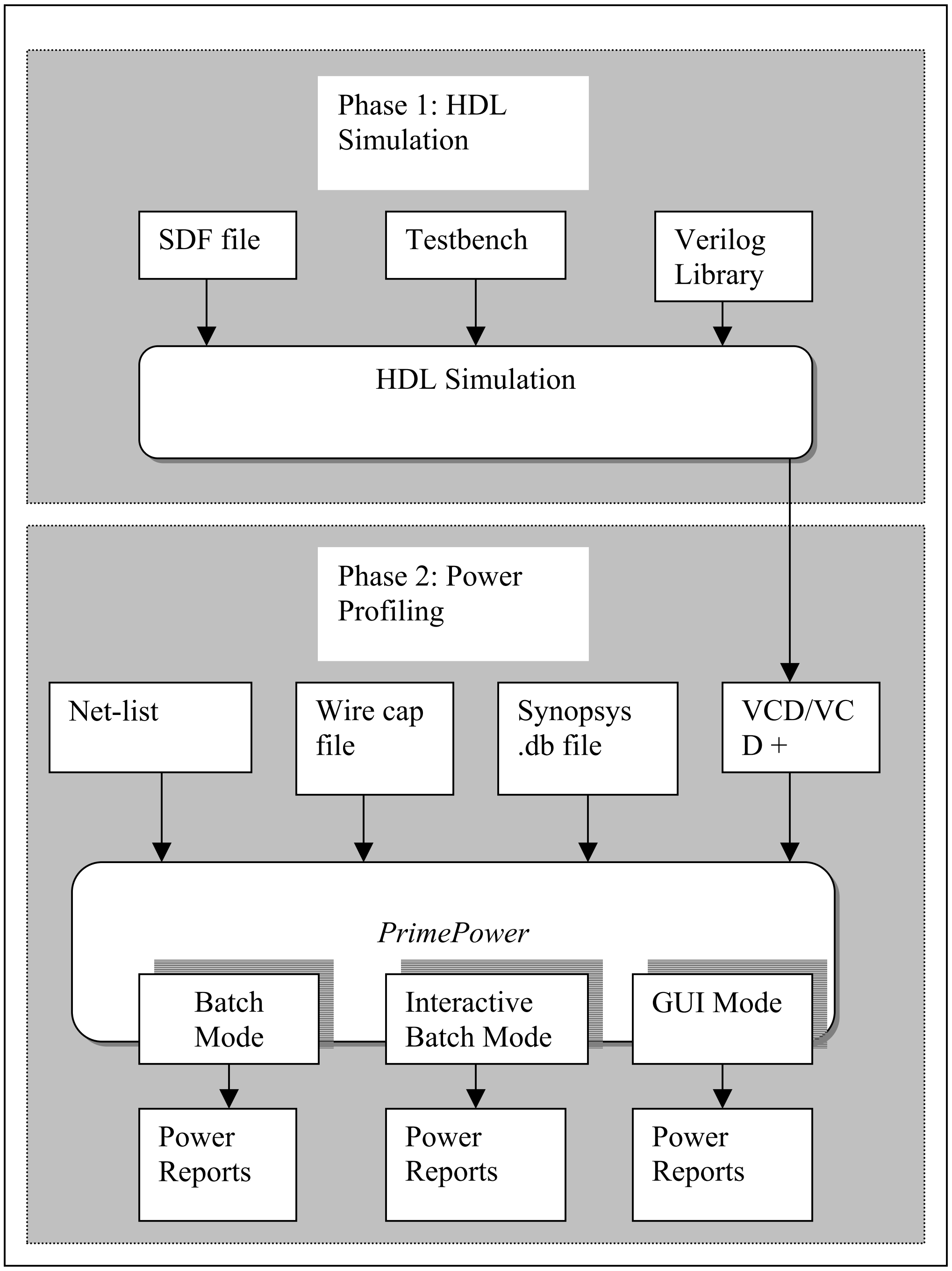


Figure 2.11 ***PrimePower*** **Methodology Flow**

2.3.2.5.1 *NanoSim* Methodology

NanoSim has a great deal of configuration commands that can give user a great deal of control over the simulation. Simulation results would significantly be affected either if the user does not provide a specific command or a command is not used appropriately. Accuracy of the results will also vary depending upon the use of the commands. *NanoSim* can report the following

a) Top-level and block-level power analysis

b) DC path report

c) Floating node report

Power analysis includes average, RMS and peak current reports in text format and also reports average, RMS and instantaneous current as waveform information that can be input in a waveform viewer and be analyzed. It can also report elements with excessive current. The DC path analysis reports conducting paths between specified voltage source nodes. It can also report nodes that stay in a high impedance longer than a certain period of time. *NanoSim* can also check a circuit for several unusual topological conditions that could result in DC leakage paths. Figure 2.12 sums up the inputs required and outputs that can be generated.

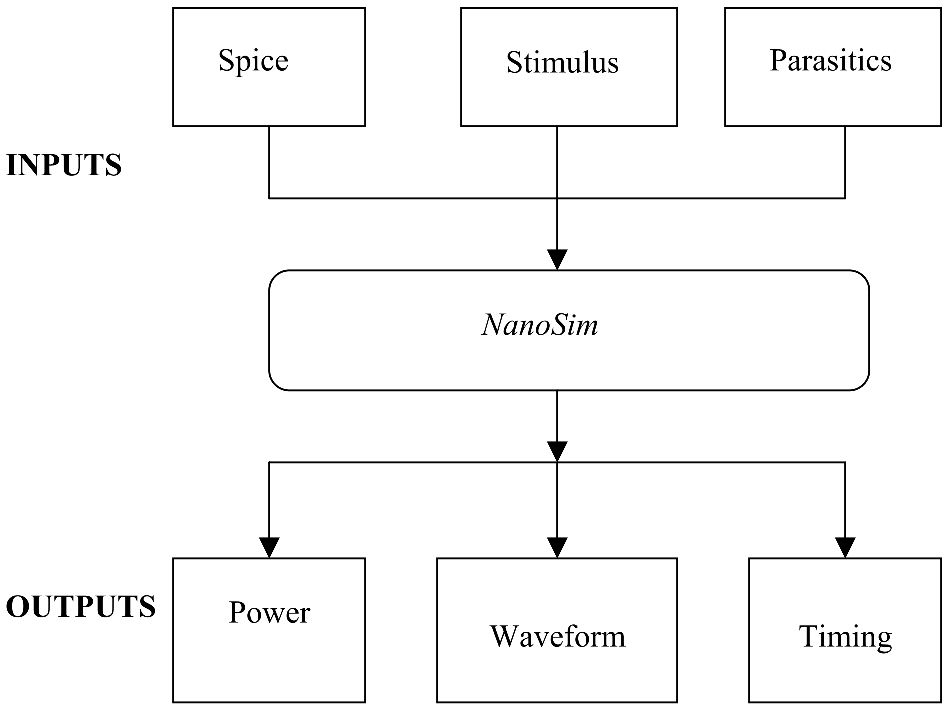


Figure 2.12 *NanoSim* Input/Output flow

2.3.2.6 *NanoSim* Integration with VCS

NanoSim integration with *VCS* is a feature that provides a mixed-signal verification solution. Figure 2.13 summarizes the flow. It enables a designer to co-simulate a design described in SPICE or other transistor-level description net-list that *NanoSim* supports and verilog-HDL. Some of the features involve direct-kernel interface during simulation, the output from *VCS* (vcd) and output from *NanoSim* (out) can be combined to one unified output file: the unified output display.

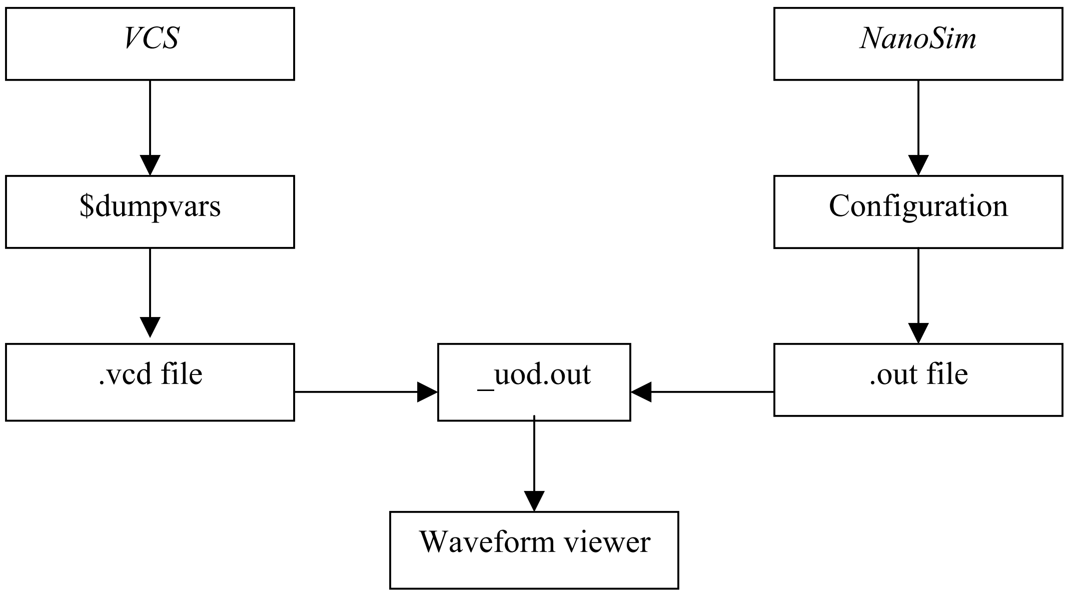


Figure 2.13 *NanoSim*-VCS flow

2.3.3 Comparison of Power Tools

The three important tools used in this thesis are *Power Compiler*, *PrimePower*, and *NanoSim*. *Power Compiler* is used at RTL and Gate level. *PrimePower* is used at Gate level and *NanoSim* is used at Transistor level. Both *Power Compiler* and *PrimePower* provide low power spectrum tools. *Power Compiler* provides analysis and power optimization at both RTL and gate level. *PrimePower* provides analysis at gate level. *Power Compiler* provides power optimizations using clock gating and gate level optimizations whereas *PrimePower* doesn’t provide power optimization.

Power analysis by *PrimePower* is much more in detail compared to the one given by *Power Compiler*. *NanoSim* comes in the last category power analysis which is transistor level. *NanoSim* not only gives power values depending on the inputs but also simulates the design to check the functionality. Power values get more accurate as we go down the level of abstraction and *NanoSim* gives the most accurate results. The following Figure 2.14 summarizes the same.

Analysis Level	**Analysis Tool**	**Optimization & Implementation**
RTL	*Power Compiler*	*Power Compiler*
Gate	*Prime Power*	*Power Compiler*
Transistor	*NanoSim*	

Figure 2.14 Power tools comparison chart

3 Experimental Design

3.1 Macros

The following macros were developed as part of a research project with the DARPA (Defense Advanced Research Projects Agency) and Boeing. The research work involved was to create macros for best PDA (Power, Delay, and Area) by exploring the best possible approach to improve those parameters. This thesis involves calculating the power of these macros.

The following are the macros used in this experiment

a) 16-bit Optimized Adder
b) 16-bit Optimized Multiplier
c) 16-bit Optimized Complex Multiplier
d) FIR (Finite Impulse Response) macro with best PDA developed using Clock Tree
e) Poly FIR with best PDA developed using Clock Tree

These macros were built for TSMC-0.18 micron process.

3.2 Power Estimation Techniques

Power values for each of these macros are done using four power tools of Synopsys spread through three levels of abstraction, RTL level, Gate level and Transistor level and in overall 5 different values for a macro being calculated. Power calculation for each of the tools at a specific level is done using a different methodology and with other non-power tools involved. One of the major non-power tools involved in this is an extraction tool. A table is built summarizing all the values.

The first method of power calculation is done using *Power Estimator* which is used at the RTL level. The second method involves using *Power Compiler* with RTL level switching activity and the third method involves using *Power Compiler* with Gate Level switching activity. The fourth method is by using *PrimePower* which also comes at Gate level. The final and the most accurate fifth method is by using *NanoSim* which is at the transistor level. The accuracy of the power values obtained using these tools gets better as we move from RTL level to transistor level. This is because the information required for calculating accurate power of a macro is given in more detail as the level goes to the lower levels of abstraction and also the tools involved get more complex at those levels. Finally, a table is made with power values filled for each of the macros together with the simulation time required to get those.

3.3 Basic design flow

The following Figure 3.1 gives a basic idea of the design flow that takes place from code writing of the macro to sending the final macro output for fabrication. Initially to start with the VHDL or Verilog hardware description language is used to describe the design. The design is verified using one of the different simulators to test its functionality. Once the test is fine, the next process of creating the net-list is carried out. The gate-level net-list is created using *Design Compiler*. The gate-level net-list

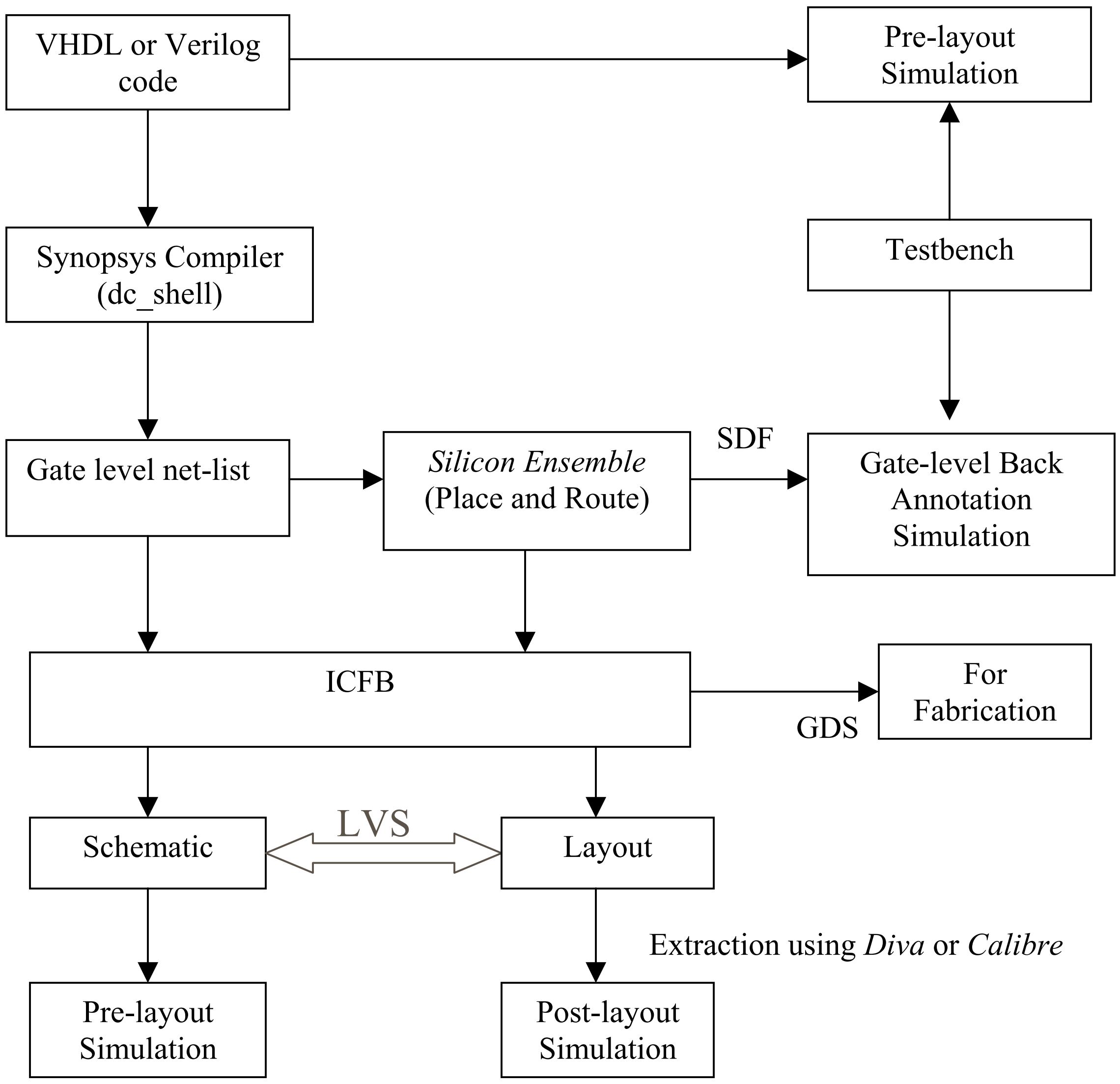


Figure 3.1 Design Flow

along with information about the standard cells are given to *Silicon Ensemble* to do place and routing of the design. SDF file is exported from *Silicon Ensemble* to do gate-level back-annotation simulation to verify the net-list .Once that is confirmed another exported file from *Silicon Ensemble*; DEF is imported into ICFB's Virtuoso layout editor to get the layout. Also the gate-level net-list obtained from *Design Compiler* is imported into ICFB's Schematic viewer to get the schematic for the design. DRC (Design Rule Check) is performed for the layout that is obtained from ICFB. The layout and Schematic is compared using a test called to confirm that the correct layout is obtained for the design. Pre-layout simulation of the schematic and Post-Layout simulation of the layout are also done to confirm the functionality. Post-layout simulation also helps in finding the delay and other details in the design. Then in order to send the design for fabrication, GDS file is exported from ICFB.

In this thesis all the processes in the design flow are carried out. Additionally, the power tools are used to estimate power at different levels depending on the tool used at a specific level of abstraction. The next sections in this chapter describe the process and methodology used in each of the power tools and how the power is calculated.

The following are the different methods of calculating power

a) *Power Estimator* using RTL level switching activity (Pre-Synthesis)

b) *Power Compiler* using Gate level net-list with RTL level switching activity

c) *Power Compiler* using Gate level net-list with Gate-level switching activity

d) *PrimePower* using VCD and Gate-level net-list

e) *NanoSim* using Spice net-list and Vector file as stimulus

3.4 Power Estimation at the Register Transfer Level

The *RTL Power Estimator* enables to obtain design power estimates early in the design process. Its pre-synthesis simulation capabilities enable to analyze the power consumption of the design at the RTL. These Architectural or RTL level tools can be used to quickly understand which modules in the entire design consume the largest amount of power. This is also the best level to evaluate the usage of clock gating strategies which are primarily used to reduce power consumption. The run time efficiency of running the tools at this level is also used to calibrate the fastness of the tool. Some of the features of using *Power Estimator* are

a) Obtain quick power estimation early in the design
b) Perform architectural tradeoffs early in the design flow
c) Identity the hotspots in the design so that more concentration can be put forth to reduce power in those areas.

3.4.1 Methodology

The following is the approach that has been followed to calculate power using *Power Estimator* which is part of the *Power Compiler* tool. Figure 3.2 gives the flow. As shown in the figure, the RTL design is first taken. There are two flows from the

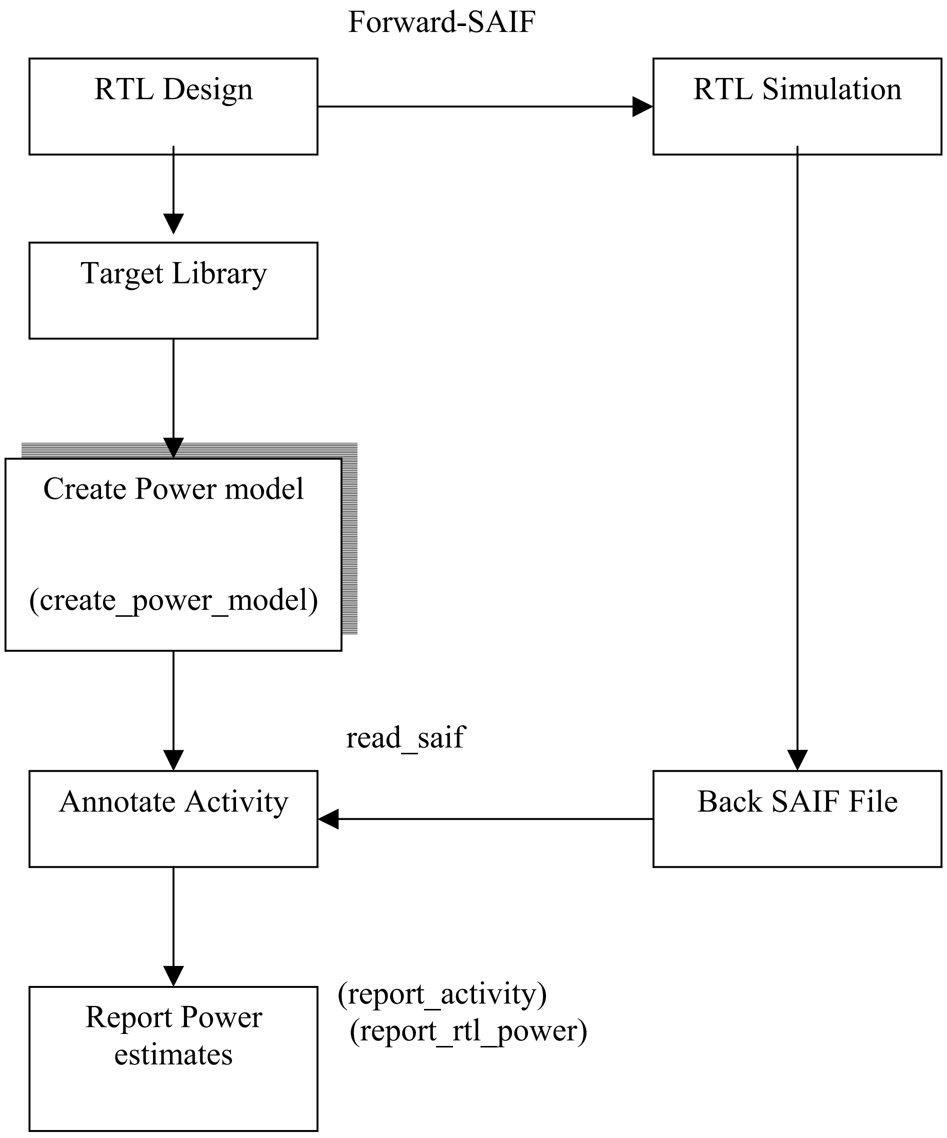


Figure 3.2 Power Analysis flow in *Power Estimator*

RTL design. One is the RTL code which is simulated using *ModelSim* simulator to get Back Switching SAIF file which contains the switching activity of the design and it is used to create power model for the design using "**create_power_model**" command. Then the design is annotated using the back annotated switching activity and power is reported using "**report_rtl_power**" command. All the commands can be added up in a script which can be used by invoking "**pp_shell**" command.

3.4.2 Capturing Forward and Backward Switching Activity

Power Compiler requires information about the switching activity of the design to do power analysis. The forward and back-annotation files are in SAIF format. SAIF is an ASCII format developed at Synopsys to facilitate the interchange of information between simulators and Synopsys power tools. Some of the power tools cannot understand SAIF file so in that case VCD file is used. Depending on the tool, either RTL level switching activity or Gate-level switching activity is used. *Power Compiler* has a methodology that enables the use of switching activity from RTL simulation as well as from Gate-level simulation. Using gate-level simulation the power values are much more accurate but doing that is time consuming. During RTL and gate level simulation the designer can direct the simulator to monitor and write out the switching activity of certain important elements in the design. For accurate analysis, synthesis-invariant elements should be closely monitored during RTL simulation. These are the elements that are not changed during simulation like primary inputs, sequential elements, black boxes, three-state devices and hierarchical ports.

3.4.2.1 SAIF file and RTL simulation

A SAIF forward-annotation file directs the simulation to monitor primary inputs and other synthesis-invariant elements. The backward SAIF file generated from the simulation contains the resultant switching activity of the elements monitored during the RTL simulation. Synopsys power tools can read the information in the back-annotation file and annotate it on the compiled design. The following steps as shown in the Figure 3.3 are done to get forward and finally the back switching activity file

a) Set the variable "**power_preserve_rtl_hier_name = true**"
b) Create a SAIF forward-annotation file from "**dc_shell**"
c) Include the SAIF forward-annotation file in simulation using *ModelSim*
d) Write a SAIF back-annotation file from simulation
e) Read the SAIF back-annotation file to annotate the design from "**dc_shell**"

As the design is analyzed and elaborated, HDL compiler creates a technology-independent design called GTECH design. Using GTECH design, HDL compiler creates the SAIF forward-annotation file when invoking the "**rtl2saif**" command.

The following is the methodology followed using RTL simulation and SAIF files.

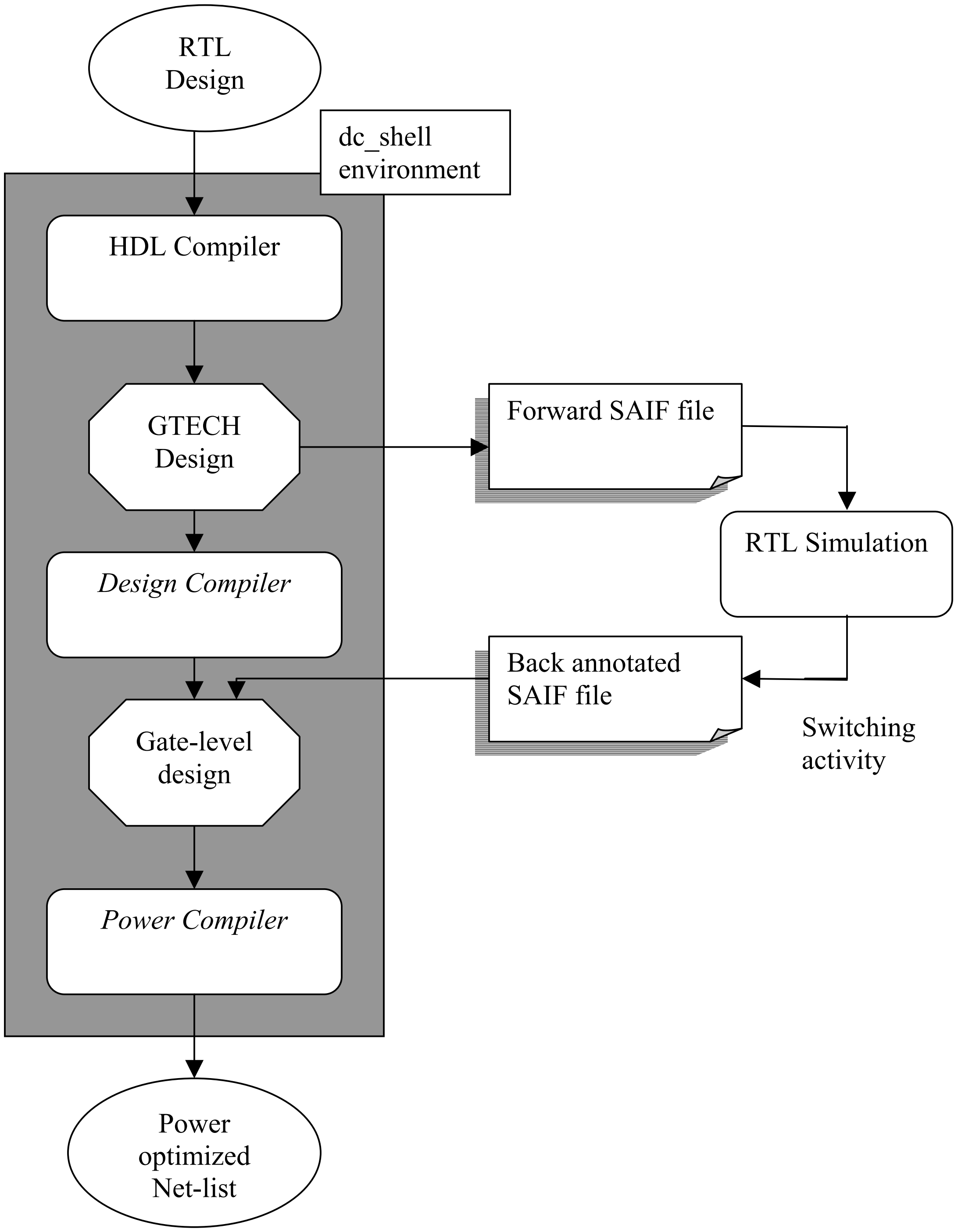


Figure 3.3 Methodology using RTL simulation and SAIF file

3.4.2.2 SAIF forward-annotation file

The following script has been used to create forward annotation file for "adder" design.

```
" power_preserve_rtl_hier_names = true
  analyze -f vhdl {adder_16.vhd}
  elaborate adder_16
  link
  rtl2saif -output adder_forward.saif -design adder_16   "
```

The following is the explanation of each of the command lines in the script. To start with the "**dc_shell**" command is used to invoke the *Design Compiler*.

a) **power_preserve_rtl_hier_names = true**

This variable is set true to preserve the hierarchy information of the RTL objects in the RTL design.

b) **analyze -f vhdl {adder_16.vhd}**
elaborate adder_16

The analyze and elaborate commands read the RTL design into active memory and converts it to a technology-independent format called the GTECH design.

c) **link**

The link command resolves instantiated references of the sub designs.

d) **rtl2saif -output adder_forward.saif -design adder_16**

The **rtl2saif** command creates the forward-annotation file using the GTECH format created during the analysis and elaboration of the RTL design. Here "**adder_forward.saif**" is the forward-annotation file for adder.

3.4.2.3 Creating Backward SAIF file

Now for *Power Estimator* to report power, Backward SAIF file is required which is obtained using Forward SAIF file. *ModelSim* simulator is used to create the backward SAIF file. First, the VHDL of adder along with the test bench are compiled and then the *ModelSim* simulator is invoked. Forward switching activity file generated by "**rtl2saif**" command as part of the *Design Compiler* is also fed to the simulator. The "**read_rtl_saif**" command reads the SAIF forward-annotation file and registers design objects for monitoring. The next subsection describes about the toggle command methodology in detail. The "**toggle_report**" command creates a SAIF back-annotation file from simulation. The back-annotation file contains information about the switching activity of the synthesis-invariant elements in the design. The "**read_saif**" dc_shell command back-annotates the information from the SAIF file onto the current design. Figure 3.4 shows the steps involved in creating the backward SAIF file.

3.4.2.3.1 Toggle command Methodology

The following is the overview of the toggle command methodology that is exclusively used in *ModelSim* simulator to get the back-annotation file. The following flow is same for RTL simulation and Gate level simulation to finally get a back-annotation file. This step is the main step in obtaining the switching activity of the design.

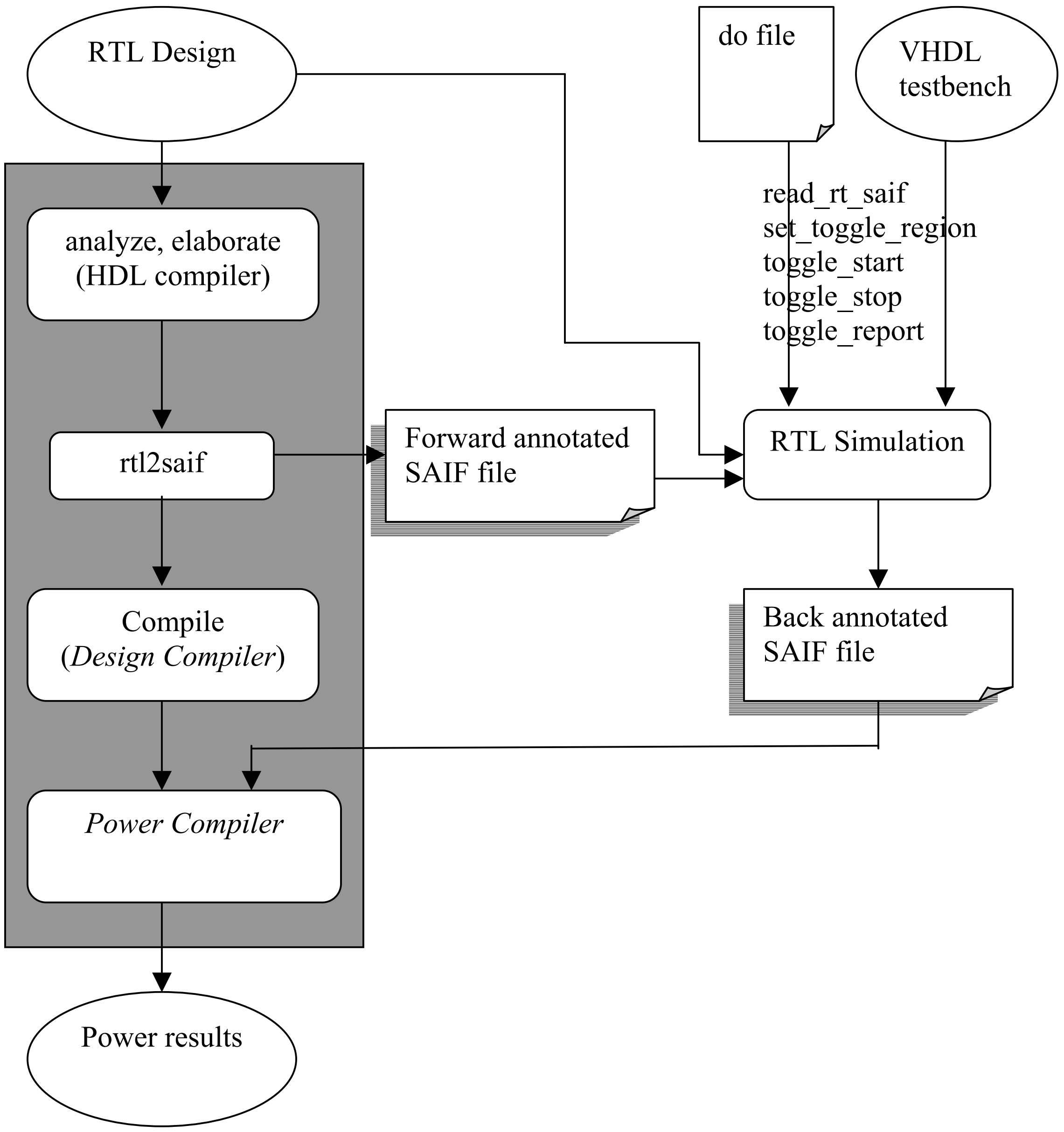


Figure 3.4 RTL backward switching activity using *ModelSim*

As seen in the Figure 3.5, first "**read_rt_saif**" command is used to read in the forward switching activity file Then "**set_toggle_region**" command is used to set the toggle region making the simulator to monitor the design objects within those regions. Then "**toggle_start**" and "**toggle_stop**" commands are used to Start and Stop the toggle monitoring respectively. "**toggle_report**" is used to write out the back-annotation file.

3.4.3 Power reporting using *Power Estimator*

The following script has been used to get the power report.

"
```
target_library = link_library =
/sw/CDS/ARTISAN/TSMC18/aci/sc/synopsys/typical.db}
create_power_model -format vhdl -hdl {adder_16.vhd} -top_design adder_16
read_saif -input FB.saif -instance adder_16 -rtl_direct
report_activity
report_rtl_power > power_report_PE "
```

Here the "**FB.saif**" file is the back-annotation switching activity file obtained from *ModelSim*. Assuming all the above commands are put inside a script, it can be run using the following command from the UNIX command prompt.

"pe_shell –f <script name> "

The power report for the adder design gets written to "**power_report_PE**" file.

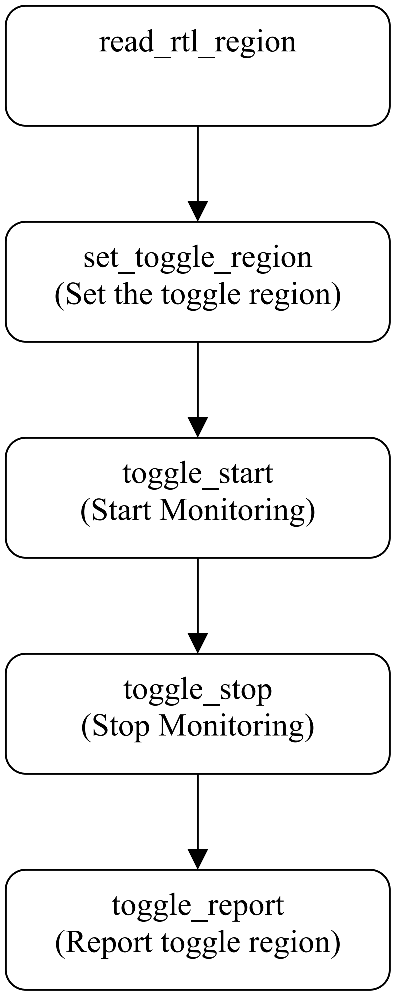


Figure 3.5 Toggle command Methodology

3.5 Power Estimation using *Power Compiler* with RTL switching activity

Power estimation at gate level using gate level power estimation tools is the next accurate method in calibrating power. These tools operate on the gate level net-list of the design together with the gate level power library. The power library consists of power models for each of the gates like inverters, NAND gates, and flip-flops. These models consists information about the parameters that contribute to power dissipation in each of the standard cells. In this thesis, *Power Compiler* is used as the gate level power estimation tool. *Power Compiler* not only estimates the power but also helps in optimizing the design for lower power. The gate level power consumption checks the power being consumed by logic transitions on wires and by capacitances and short circuits internal to gates during an input transition. In the case of smaller design, the designer can do some gate level changes to reduce power after estimating. If it is a larger design then it would be difficult for the designer to check all the gate-level changes. At this point, Power optimization tools come in handy. *Power Compiler* is also an optimization tool.

3.5.1 Methodology

In this method of power estimation, *Power Compiler* is used with the same RTL back-annotation switching activity used for power estimation using *Power Estimator* but instead of RTL code, it uses gate-level net-list of the design.

. Also for getting better power result, parasitic information of the system is also provided. In this case DSPF is obtained from Place and route tool, *Silicon Ensemble* using *HyperExtract* Extraction tool. The gate-level net-list is obtained from *Design Compiler*. The following script has been used to report power.

"

```
read -f verilog -net-list adder_16_bestPDA.v
current_design adder_16
create_clock -name clk -period 50 -waveform {0 25}
set_load 0.005410 SUM[*]
read_parasitics -format DSPF adder_16_bestPDA.dspf   -elmore
read_saif -input FB.saif -instance adder_16
report_power > power_report_RTL "
```

As shown in the script first the gate-level net-list of the adder design obtained from *Design Compiler* is read inside the "**dc_shell**" environment. Depending on the clock frequency used, it has been assigned using the "**create_clock**" command. The parasitic is read in the form of DSPF file using "**read_parasitics**" command. Then the backward SAIF file is loaded using "**read_saif**" command. Then finally "**report_power**" command is used to report the power. Depending on the design, extra commands may be required in this script especially for designs having a clock tree. Designs having clock tree will report high fanouts when run in this environment. Additional commands will enable to remove the high fanouts.

3.6 Power Estimation using *Power Compiler* with Gate-level Switching activity

Another method of calculating power of a design which is more accurate than the previous *Power Compiler* method is to use gate-level net-list with gate-level switching activity. This method is better than the previous method because it uses the gate level net-list to get the switching activity of the design, but the time taken to do this procedure is more than previous two methods.

3.6.1 Creating Gate-level Switching Activity

The following Figure 3.6 shows the flow required to get the Back annotation gate level switching activity which will be later used to calculate power. The main difference between RTL back annotation switching activity and gate-level switching activity is that here gate level net-list is given as the input to the *ModelSim* simulator along with the testbench and the do file which contains all the toggle region definition and the actual running of the simulation and the reporting of the toggle activity. The resultant back-annotation SAIF file is read back to *Power Compiler* and power is reported. The do file that is used to capture switching activity follows the same procedure as RTL switching activity like defining the reading the forward SAIF file, defining the region for counting toggle information, starting and stopping the monitoring switching activity and finally using "**toggle_report**" command to report the activity in a SAIF file format.

```
"
 read -f verilog -net-list adder_16_bestPDA.v
current_design adder_16
create_clock -name clk -period 50 -waveform {0 25}
set_load 0.005410 SUM [*]
read_parasitics -format DSPF adder_16_bestPDA.dspf   -elmore
read_saif -input backgateadder.saif -instance testbench/design
report_power > power_report_Back   "
```

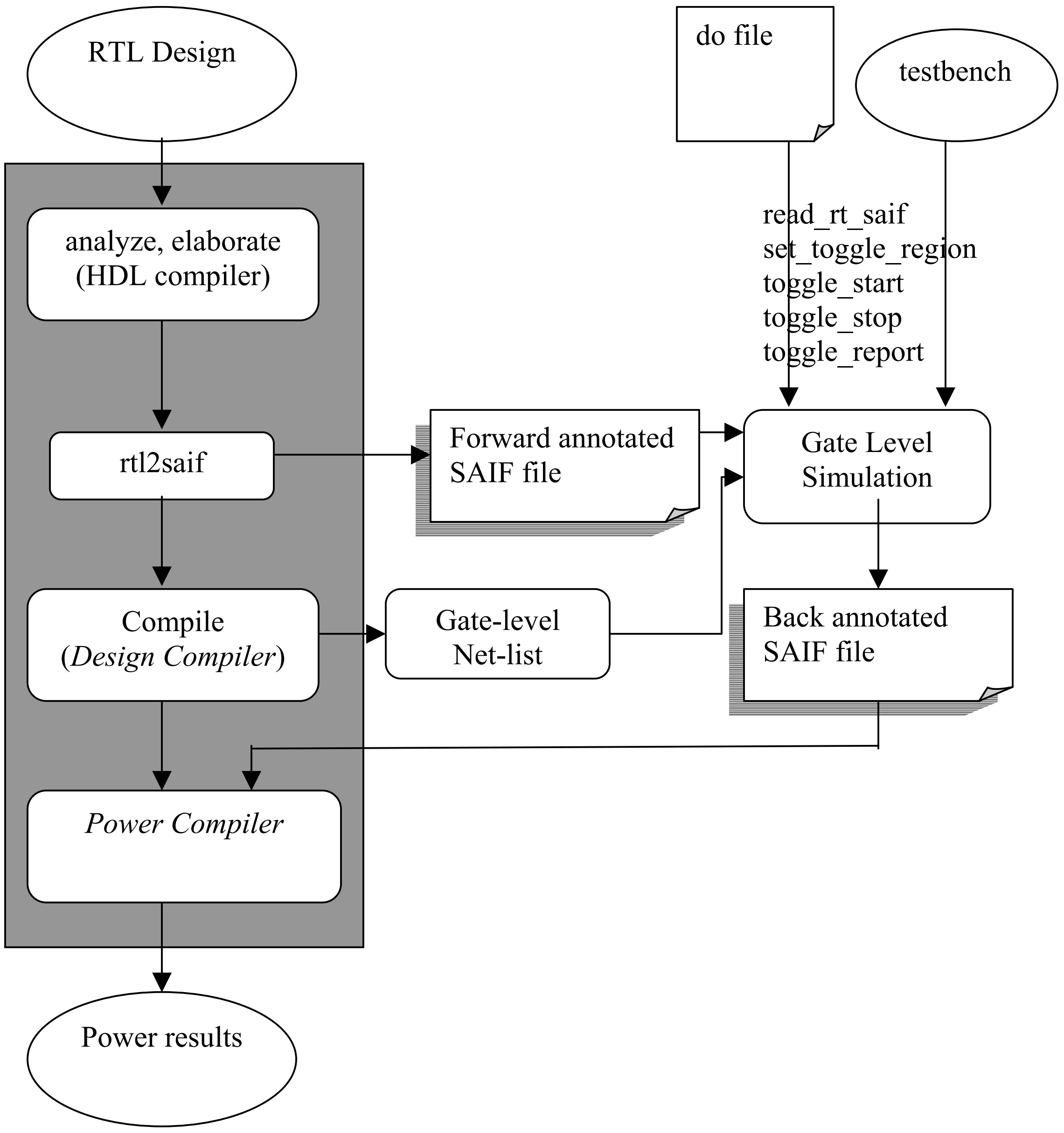


Figure 3.6 Gate-level backward switching activity using *ModelSim*

First the gate level net-list is read into dc_shell environment. Once the net-list is read the top level of the design is made as the current design to work on it. Then the clock is created depending on the frequency is run while calculating the power. Then a certain load is given to the output port which in this case is SUM. Then the parasitic values are read into as DSPF form. Then the backward annotation file is read which has the switching activity of the design. The switching activity file gives information to the tool at which points there is switching in the design. This is useful to report power of the design. "**report_power**" command is used to report the power of the design. This method gives power values much more accurate the other previous methods. Next method discussed is by using another Gate-level *Power Estimator* using almost the same input files except that it takes in the switching activity as VCD format. This tool is supposed to give almost equal power compared to *Power Compiler* using Gate level switching activity.

3.7 Power Estimation using *PrimePower*

PrimePower is a dynamic gate-level simulation and analysis tool that accurately analyzes power dissipation of cell-based designs. Let us see some of the differences between *Power Compiler* and *PrimePower* since both are Synopsys power tools used to calculate power at gate-level of a design. Not only does *Power Compiler* determine power at the gate-level, it can also calibrate power at the RTL level. *Power Compiler* can be used to do power optimization using clock gating, operand isolation whereas *PrimePower* cannot be used to do power optimization. *Power Compiler* reports for average power and are performed at the block level. *PrimePower* provides full chip power analysis that can include non-synthesizable cells such as I/O and memories. The output from *Prime Power*'*s* power report can be viewed in waveform viewers to analyze the determined power. *PrimePower* also helps in determining instantaneous power analysis that helps in identifying the hot spots in the design. It can read in VCD file that provides time-based simulation events. The methodology in *PrimePower* has already been discussed here.

3.7.1 Methodology

The following Figure 3.7 discusses the *PrimePower* analysis flow to perform power analysis of the design. The steps are here

a) Read the design data which includes the gate-level net-list and associated technology libraries

b) Read in the activity file in the form of VCD

c) Specify the environment and analysis conditions such as operating conditions and calculate the power consumption

d) Examine the power results using waveform viewers.

The first step is to read in the gate-level design description and the associated technology library information. *PrimePower* accepts design descriptions and library information in the form of .db (Synopsys database) format and gate-level net-list in Verilog, and VHDL format. "**read_db**" command is used inside *PrimePower* shell to read in the database formats. "**read_verilog**" command is used to read in the verilog

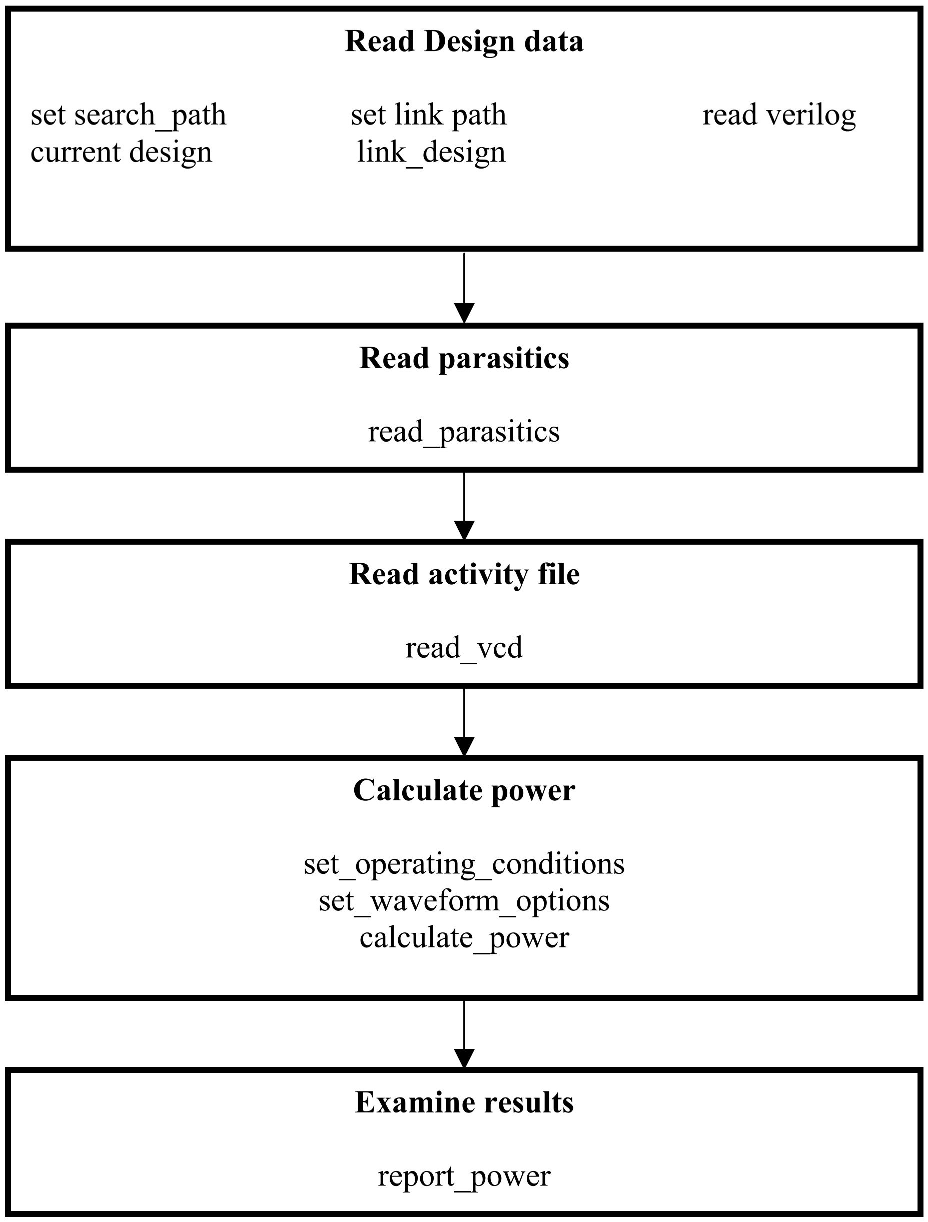


Figure 3.7 ***PrimePower*** **Analysis flow**

gate-level net-list and “**read_vhdl**” command is used to read in the VHDL gate-level net-list.

After the design files are read in, the “**link_design**” command helps in building a reference between all the modules in the hierarchy and creates an internal representation for the tool to do power analysis. Then the activity file is read in using “**read_vcd**” command.

Then the operating environment and conditions are specified using various commands like “**set_operating_conditions**”, “**set_load**”, etc. The “**set_operating_conditions**” command specifies the operating condition for analysis, so that *PrimePower* can use the appropriate set of parameter values in the technology library. “**set_load**” command is used to specify the amount of capacitance on a port or net. One of the most important inputs to *PrimePower* is DSPF file or any parasitic file obtained from place and routing tool. *PrimePower* uses “**read_parasitics**” to back-annotate the design with detailed parasitic resistance and capacitance information. Also the characteristics of the design can be verified using “**report_design**”, “**report_port**”, “**report_net**” commands. If the output of *PrimePower* needs to be viewed in a waveform then for that “**set_waveform_options**” is used. After all the design files and constraints are set, “**calculate_power**” command is used to calculate power. “**report_power**” is finally used to report the power.

3.7.2 Generating *PrimePower* inputs

The three inputs required for *PrimePower* are

a) Gate-Level net-list

b) Switching activity file in the form of VCD file

c) Standard parasitic file in the form of DSPF or any other standard formats

Gate-level net-list is obtained from *Design Compiler* in the "**dc_shell**" environment. Standard parasitic file is obtained from Cadence place and route tool, *Silicon Ensemble*.

VCD files are generated by many methods. One of the methods is to use **"$dumpfile (<filename>)** "and **"$dumpvars (<level>, <module>)**" commands in standard simulators. Second method is by using another simulator *VCS*. Using the gate-level net-list and the testbench associated with it "**vcs <testbench name> <net-list name>"** command can be used to generate the VCD file. The accuracy in creating these files is important as they directly affect the power values. Other formats of VCD are compressed VCD, gzipped VCD. These activity files are converted to VCD once they are given as input to *PrimePower*. They are done using appropriate tools associated with *PrimePower*. "read_vcd" command is used to read in the VCD file. Options are available with the reading of the VCD file.

3.7.3 Power Calculation

Power analysis by *PrimePower* depends on the conditions specified such as input transitions, port capacitance, wire load models and operating conditions. "**set_input_transitions**" command defines a fixed transition time for input ports. *PrimePower* uses the specified transition time in calculating the power of logic driven by the port. "**set_load**" is used to set load capacitance on ports. Wire load models are also specified using "**set_wire_load**" command. If the design has got clock tree, "**create_clock**" command is used to define the clock. Finally, after setting all the parameters, "**calculate_power**" is used to calculate power. There can be time limit set in this function so that *PrimePower* can be asked to calculate power only during those time intervals.

3.7.4 Power Reporting

PrimePower can give a detailed power report using the power reporting commands. The power dissipation report (*.rpt) is produced by default. Some of the powers that it can report are

a) Total power = Total dynamic and leakage power
b) Dynamic power = Total power consumption due to switching capacitances, glitches
c) Leakage power = Reverse-biased junction leakage and sub-threshold leakage.

d) Internal power = Dynamic power consumed inside a cell

e) Glitch power = Power dissipated into detectable glitches at the nets

Also the one of the forms of output as result of power analysis is an fsdb file. This file can be opened in a waveform viewer to view the consumption of power at the time intervals.

The following is the script used to calculate power for adder circuit.

```
“set search_path {. /sw/CDS/ARTISAN/TSMC18/aci/sc/synopsys}

 set link_library {* typical.db}
#-----------------------------------------------------------------------
#      Load Design and Activity Files
#-----------------------------------------------------------------------
read_verilog {adder_16_bestPDA.v}
current_design adder_16
link
set_hier_sep /
read_vcd  -strip_path testbench/adder_16  adder_16_vcd.vcd
#-----------------------------------------------------------------------
#      Apply Default Parameters
#-----------------------------------------------------------------------
set_input_transition .1 inst_A[*]
set_input_transition .1 inst_B[*]
set_operating_conditions typical
set_load 0.05410 "SUM[*]"
-----------------------------------------------------------------------
#      Back annotation: Uncomment the commands which apply
#-----------------------------------------------------------------------
read_parasitics adder_16_bestPDA.dspf
# current_instance fillin
# source fillin
#-----------------------------------------------------------------------
#      Power Analysis and Waveform Generation
#-----------------------------------------------------------------------
#set_operating_conditions fillin
set_waveform_options    -interval 0.01 -file adder_16_bestPDA -format fsdb
calculate_power         -waveform -reset_neg_power
report_power           -file adder_16_bestPDA -threshold 0 -sortby power -leaf
#-----------------------------------------------------------------------
```

```
#       report capacitance
#------------------------------------------------------------------------
#report_wire fillin
quit “
```

3.8 Power Estimation using *NanoSim*

The next power estimation of a design can be performed using a transistor-level power estimation tool. Transistor level power analysis tools are very accurate and values obtained from them are very reliable for design engineers. The transistor level of abstraction is also acceptable to get the power values. The tool used in this thesis is *NanoSim*. *NanoSim* is a high-speed, high-capacity circuit simulator that combines best-in-class circuit simulation technologies from Synopsys’s *PowerMill* and *TimeMill*. *NanoSim* simulates block and full chip current and power behavior and is much faster than HSPICE and comes into 2-5% of accuracy of SPICE. It is also used to diagnose the design flaws by checking the areas where power is consumed more. Once the specific area of high power consumption is detected, the design code can be rewritten and that hotspot can be concentrated more to reduce the unwanted power.

Inputs required for *NanoSim* are the SPICE net-list of the design, stimulus file giving the inputs for the design, configuration file that contains information that tells *NanoSim* how to perform the simulation, technology file that describes the key features of the process technology that *NanoSim* uses to predict the transistor behavior in the circuit. SPICE net-list of the design is obtained using an extraction tool, the stimulus file is written by the designer according to the design, technology files are got from the vendor dealing with the technology for which the design is built for and the configuration file has the required commands instructing *NanoSim* to simulate in the way designer wants.

3.8.1 Extracting SPICE net-list from *Calibre*

First once the design is written and its functionality tested, it is run through *Design Compiler* to get the gate-level net-list. The same gate-level net-list has been used in the prior power estimation methods. Now the gate-level net-list has to be imported to a place and route tool to get the routing done. Depending on the technology, the technology file, the verilog description of all the standard cells along with this gate-level net-list is imported into Cadence's *Silicon Ensemble*. Once the routing is done, the output that can be transformed into layout which is exported out in the form of DEF file .This DEF format is imported into Cadence's ICFB environment to get the layout. As said in the previous section one of the main inputs for *NanoSim* is the Spice net-list of the design. Depending on the complexity of the design the time taken to generate the SPICE net-list is more and also the tool should be powerful enough to generate SPICE net-list fast. Here comes the Extraction tool *Calibre*. As shown in the Figure 3.8, the layout of the adder design under discussion has been obtained from DEF using ICFB.

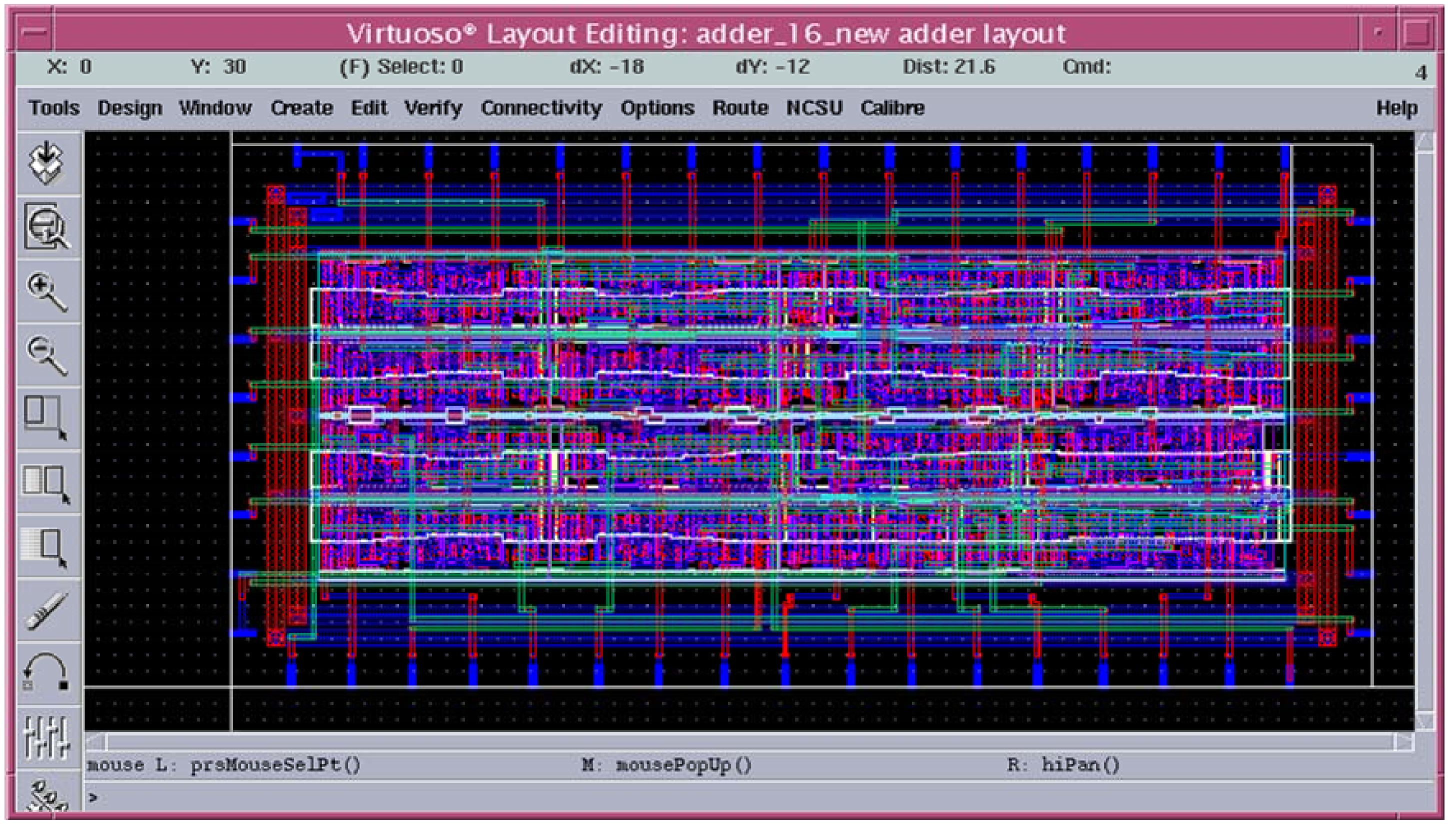


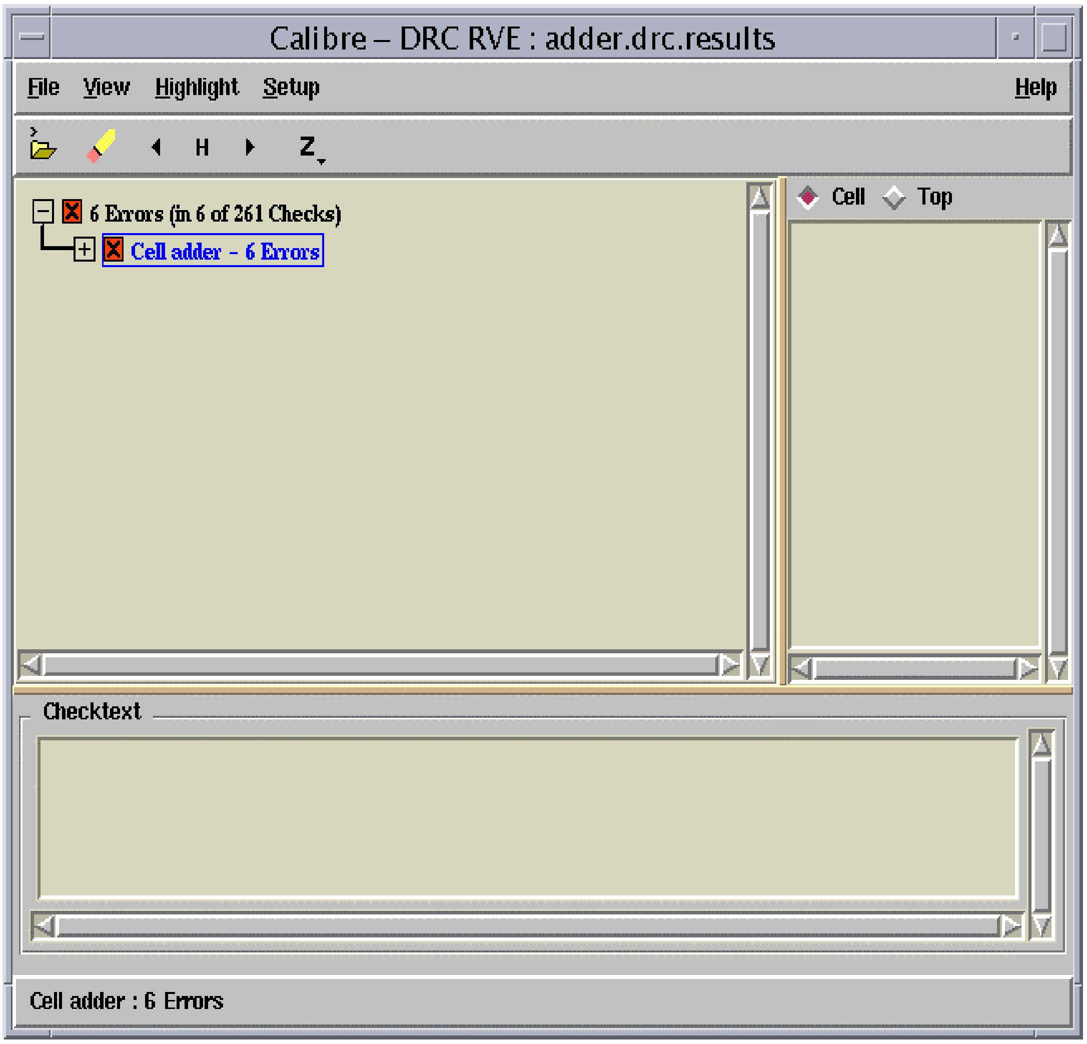
Figure 3.8 Adder Layout

3.8.1.1 Design Rule Check (DRC) of Adder Layout.

Now the obtained layout has to be checked for any design rule errors. This is done using *Calibre*. First the template file has to be imported which gives information about the creation of the GDS file which has all the layout paths. Using this *Calibre* checks the design for any errors and that all design rules have been followed. The Figure 3.9 shows the final output after DRC check is done.

3.8.1.2 Layout vs. Schematic (LVS) check for Adder Layout

Once the DRC check is over, the layout has to be checked if that is the corresponding layout for the schematic of the design. When *Calibre* checks for layout vs. schematic, it basically compares the net-list of the two. When the DRC check is done, it already creates the net-list for the layout. In order to get the net-list of the schematic, another Synopsys's tool, *NetTran* has been used. This tool takes in the gate level net-list of the design and gives us the net-list which corresponds to the schematic of the design. Now the template pertaining to LVS check is imported into *Calibre*. Now the LVS check will be performed. The following Figure 3.10 shows the LVS check for adder.

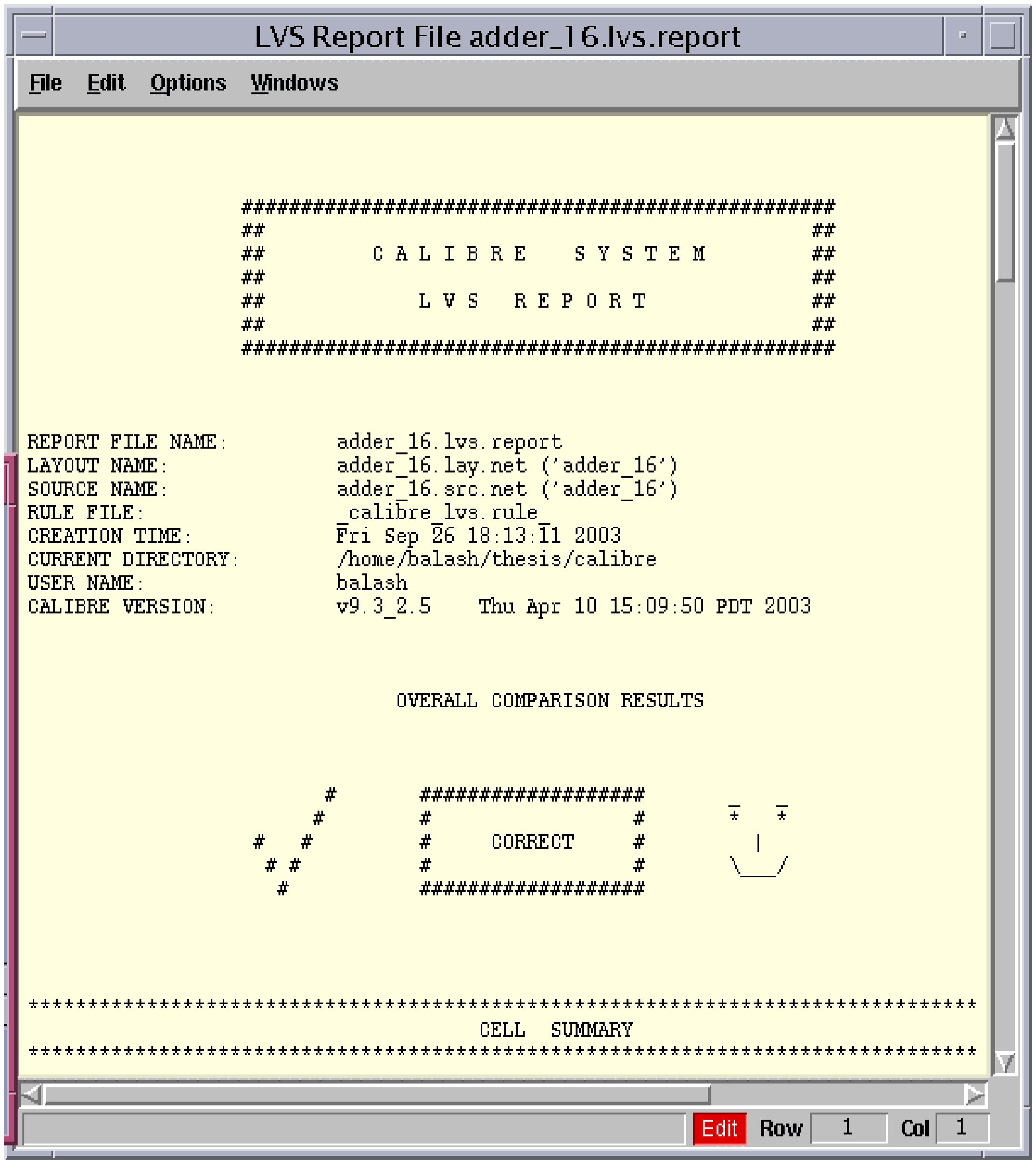

Calibre – DRC RVE : adder.drc.results
File View Highlight Setup Help
Cell Top
Checktext

6 Errors (in 6 of 261 Checks)

Cell adder - 6 Errors

Cell adder : 6 Errors

Figure 3.9 DRC check of Adder Design


LVS Report File adder_16.lvs.report
File Edit Options Windows
Edit Row 1 Col 1

```
                   ##################################################
                   ##                                              ##
                   ##          C A L I B R E    S Y S T E M        ##
                   ##                                              ##
                   ##              L V S   R E P O R T             ##
                   ##                                              ##
                   ##################################################


REPORT FILE NAME:          adder_16.lvs.report
LAYOUT NAME:               adder_16.lay.net ('adder_16')
SOURCE NAME:               adder_16.src.net ('adder_16')
RULE FILE:                 _calibre_lvs.rule_
CREATION TIME:             Fri Sep 26 18:13:11 2003
CURRENT DIRECTORY:         /home/balash/thesis/calibre
USER NAME:                 balash
CALIBRE VERSION:           v9.3_2.5    Thu Apr 10 15:09:50 PDT 2003


                              OVERALL COMPARISON RESULTS



                          #       ###################
                         #        #                 #       _   _
                    #   #         #     CORRECT     #       *   *
                     # #          #                 #         |
                      #           ###################       \___/


*********************************************************************************
                                  CELL   SUMMARY
*********************************************************************************
```

Figure 3.10 LVS check of Adder Design

3.8.1.3 Net-list extraction for Adder

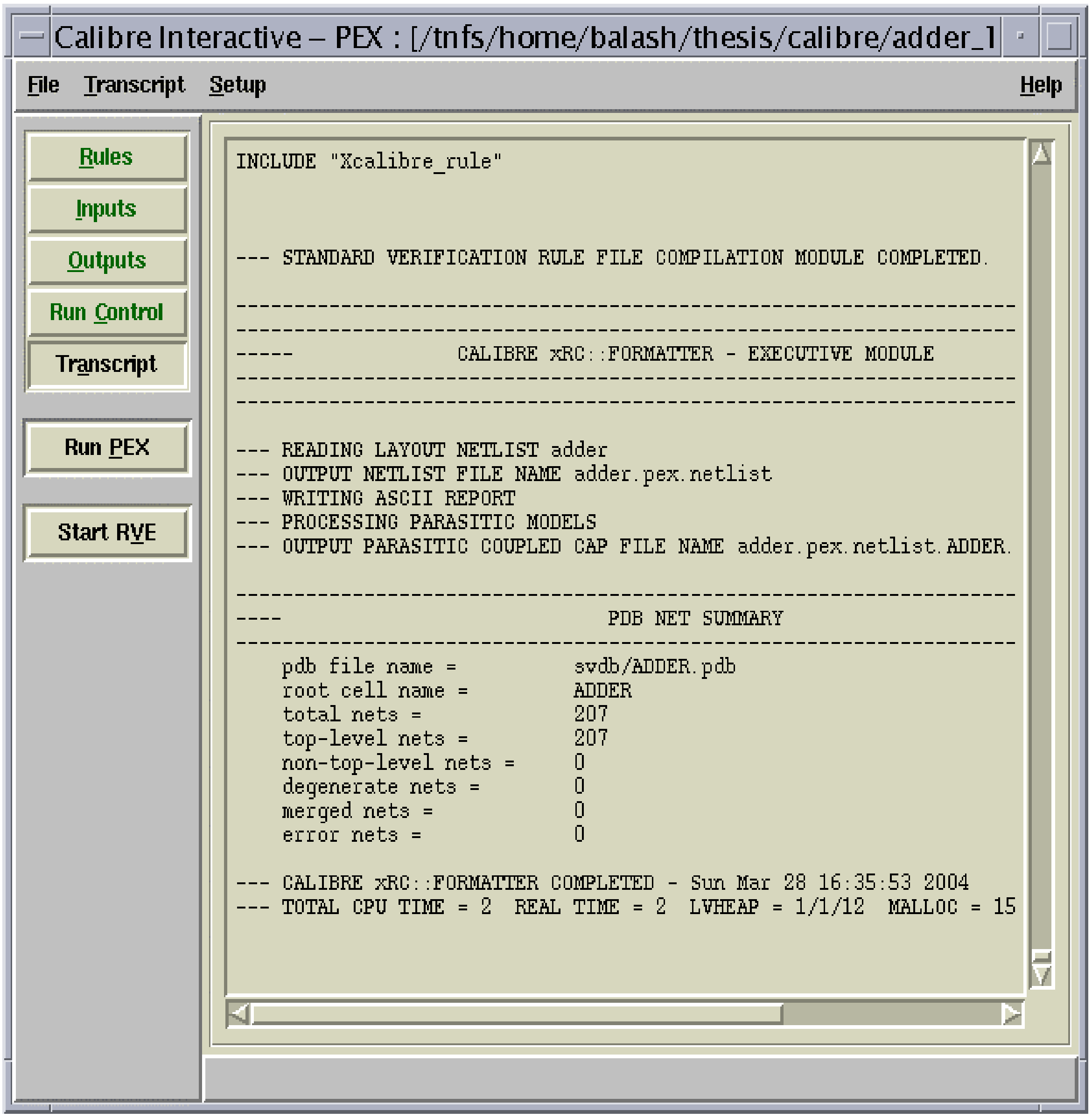
Now that the DRC and LVS check are performed, the net-list required for *NanoSim* has to be extracted. The template which tells *Calibre* where to look for the extraction rule file is imported. The required net-list type is selected. It can be Flat SPICE or Flat DSPF. Once they are selected, the operation PEX is performed to get the required net-list. The time to generate net-list depends on the design. The following Figure 3.11 is the output after extraction is done.

3.8.2 *NanoSim* inputs and Power reporting

The following are the inputs required for *NanoSim* simulation.

a) Net-list file – This has information about the circuit to be simulated
b) Stimulus file – This has the Input stimulus to the circuit
c) Configuration command file – This file tells *NanoSim* how to perform the simulation
d) Technology file – This has the look-up table for model characterization

NanoSim accepts multiple net-lists. In our case, the net-list is a flat DSPF file which has all the transistors and also RC parasitic values. SPICE net-list is given by –n option while running *NanoSim*.



```
INCLUDE "Xcalibre_rule"



--- STANDARD VERIFICATION RULE FILE COMPILATION MODULE COMPLETED.

-------------------------------------------------------------------
-------------------------------------------------------------------
-----              CALIBRE xRC::FORMATTER - EXECUTIVE MODULE
-------------------------------------------------------------------
-------------------------------------------------------------------

--- READING LAYOUT NETLIST adder
--- OUTPUT NETLIST FILE NAME adder.pex.netlist
--- WRITING ASCII REPORT
--- PROCESSING PARASITIC MODELS
--- OUTPUT PARASITIC COUPLED CAP FILE NAME adder.pex.netlist.ADDER.

-------------------------------------------------------------------
----                           PDB NET SUMMARY
-------------------------------------------------------------------
    pdb file name =          svdb/ADDER.pdb
    root cell name =         ADDER
    total nets =             207
    top-level nets =         207
    non-top-level nets =     0
    degenerate nets =        0
    merged nets =            0
    error nets =             0

--- CALIBRE xRC::FORMATTER COMPLETED - Sun Mar 28 16:35:53 2004
--- TOTAL CPU TIME = 2  REAL TIME = 2  LVHEAP = 1/1/12  MALLOC = 15
```

Figure 3.11 Extraction output of Adder design

If the file extension of the net-list is not .sp or .spi, then a different option is given depending on the file extension. Next is the Stimulus file which has the input stimulus for the circuit. The format used here is Vector stimulus which is one form of EPIC stimulus. If the EPIC vector function is called in a file, then –n option is used within *NanoSim*. In this case, the vector function call and vector stimulus are in two separate files. On the other hand, the type and signal commands can be included in a vector stimulus file directly and –nvec option is used while *NanoSim* is executed. Next important input to *NanoSim* is the configuration file which instructs it how to perform simulation. The accuracy and speed at which the results are obtained and simulated largely depends on the input commands given in this file. The number of output files generated as result of *NanoSim* execution also depends on the number of commands given in this file. The input commands given fall into the categories of net-list compilation, circuit modification, simulation control, circuit partitioning, simulation accuracy, and output reporting and message control. Some of the commands given during the simulation are given here

```
“set_print_format for=fsdb
 set_sim_tres 100ps
 set_sim_eou sim=4 model=4 net=4
 set_ckt_ctrl ba_process_fcap:1
 print_node_i VDD VSS
 report_block_powr x1 track_src=1 track_gnd=1 track_wasted=1 track_power=1
* “
```

The command “**set_print_format**” is used to tell *NanoSim* to give the output of the simulation in the form of FSDB file which can be used to be viewed in a waveform viewer. “**set_sim_tres“** is used to define the time resolution for the simulation.

Larger time resolution results in faster, but less accurate simulation. The variable to control the simulation for accuracy and performance is given by “**set_sim_eou”** command. Since the RC parasitic values are given in the net-list, “**set_ckt_ctrl ba_process_fcap:1”** is used to make *NanoSim* take all the capacitance values without ignoring them. “**print_node_i**” is used to print the node VDD and VSS. “**report_block_powr”** is used to report the power after the simulation. Depending on the flags given in this command, different power like RMS power, Average power of the circuit will be displayed. Using “report” command, the value is also written to a file. The output file .out and .fsdb are analyzed in waveform viewers like *TurboWave*, *CosmoScope* or *SimWave*.

4 Results and Discussion

This chapter gives details on the various results that have been obtained using the different macros that were discussed earlier. Results are given in three main tables;

a) Comparison power table for Default macros Vs Best-PDA macros.

b) Power table for Default/ Best-PDA Ratios

c) Simulation time taken by *NanoSim* simulator for each Default and Best-PDA macros.

Add-on tables for the above tables show the percentage of error for the comparison table and for the ratio table. Tables comparing the difference in version of the tools have also been created. The following Figure 4.1 shows the methodology of calculating different power values at the different levels of abstraction. Detailed methodology of how different power values are calculated using these tools has been discussed in chapters two and three.

The following section discusses the different power values that are obtained using different power tools as shown in the above figure.

1. ***Power Estimator* – P1 (RTL) :**

Power Estimator is used to calculate power at the RTL level. The inputs for *Power Estimator* are VHDL [1], RTL switching activity [4]. The input [4] is got from *ModelSim* giving [1] + [2] as inputs.

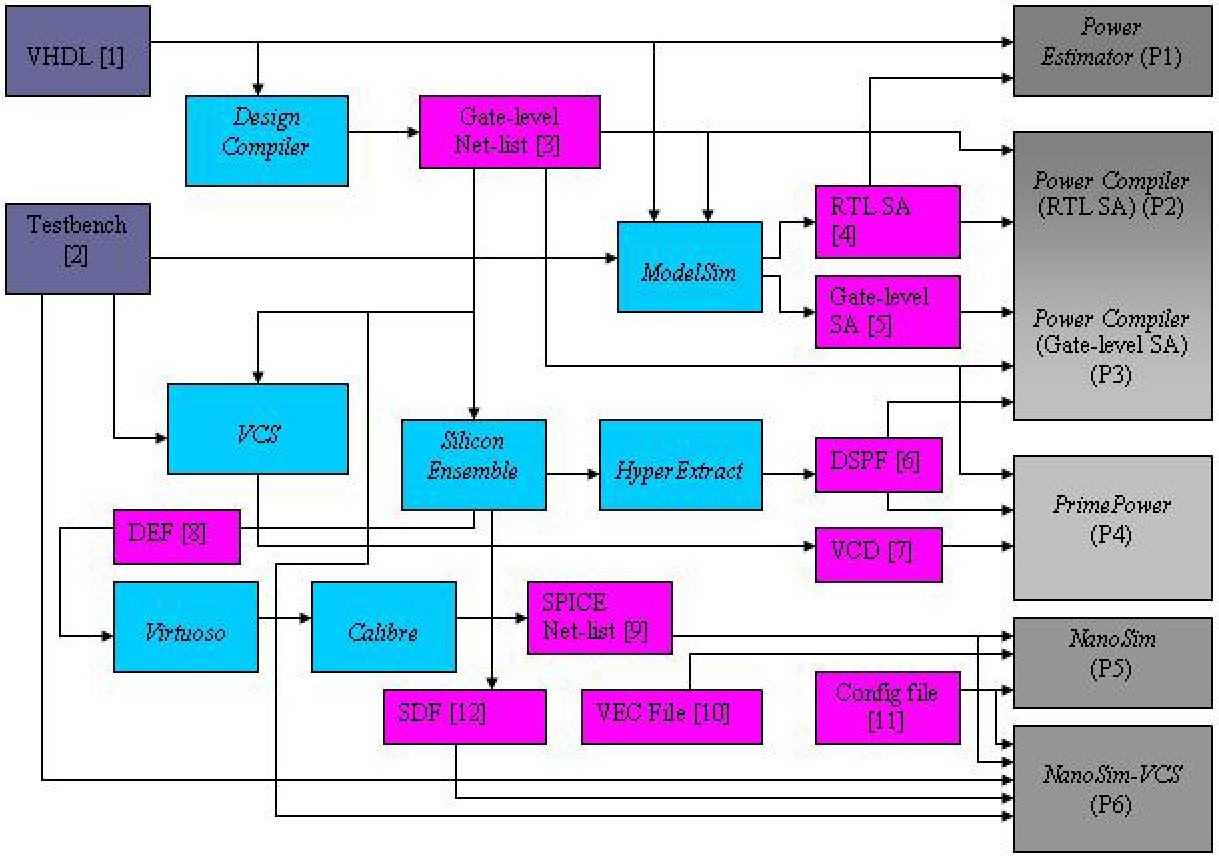


Figure 4.1 Design Methodology showing power calculations using different power tools

2. ***Power Compiler* – P2 (RTL) :**

The second power value is calculated using *Power Compiler* at the RTL level. The inputs to calculate power are Gate-level Net-list [3], RTL switching activity [4]. [3] is obtained from *Design Compiler* giving [1] as the input. [4] is obtained from ModelSim giving [1] + [2] as inputs.

3. ***Power Compiler* – P3 (Gate-level) :**

The third power value is calculated using *Power Compiler* at the Gate-level. The inputs to calculate power are Gate-level Net-list [3], Gate-level switching activity [5], Parasitic information [6]. [3] is obtained from *Design Compiler* giving [1] as the input. [5] is obtained from *ModelSim* using [2] + [3] as inputs. [3] is given as input to *Silicon Ensemble* to do the Place and Routing and after that [6] is obtained running *HyperExtract* in *Silicon Ensemble*.

4. ***PrimePower* – P4 (Gate-level):**

The fourth power value is calculated using *PrimePower* at the Gate-level. The inputs to calculate power are Gate-level Net-list [3], Testbench input in the form of VCD file [7], Parasitic Information [6]. [7] is obtained giving [2] as input to *NanoSim-VCS*. [3] and [6] are obtained as the same way as discussed in 3.

5. ***NanoSim* – P5 (Transistor level):**

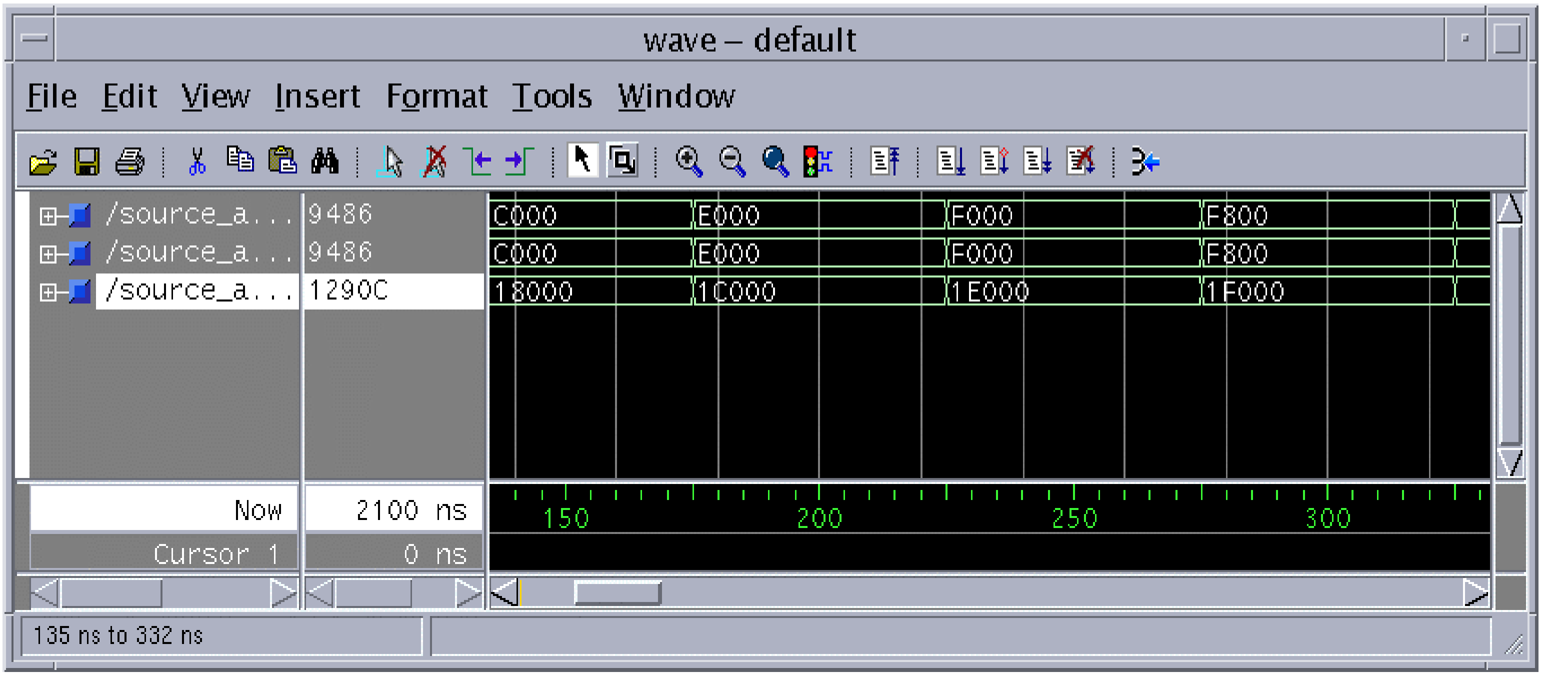
The fifth power value is calculated using *NanoSim* at the Transistor level. The inputs to calculate power are SPICE Net-list [9] , Input stimulus [10] and Configuration file [11]. The output DEF [8] from *Silicon Ensemble* is input into *Virtuoso* to get the layout which is then given as Input to *Calibre* to get SPICE Net-list [9]. Input Stimulus [10] is given to *NanoSim* in the form of vector file obtained from Testbench. Configuration file [11] is given for better simulation results.

6. *NanoSim-VCS* – **P6 (Transistor level):**

The sixth power value is calculated using *NanoSim-VCS* at the Transistor level. The inputs to calculate power are Gate-level Net-list [3], Testbench [2], SDF file [12] , SPICE Net-list [9] and Configuration file [11].

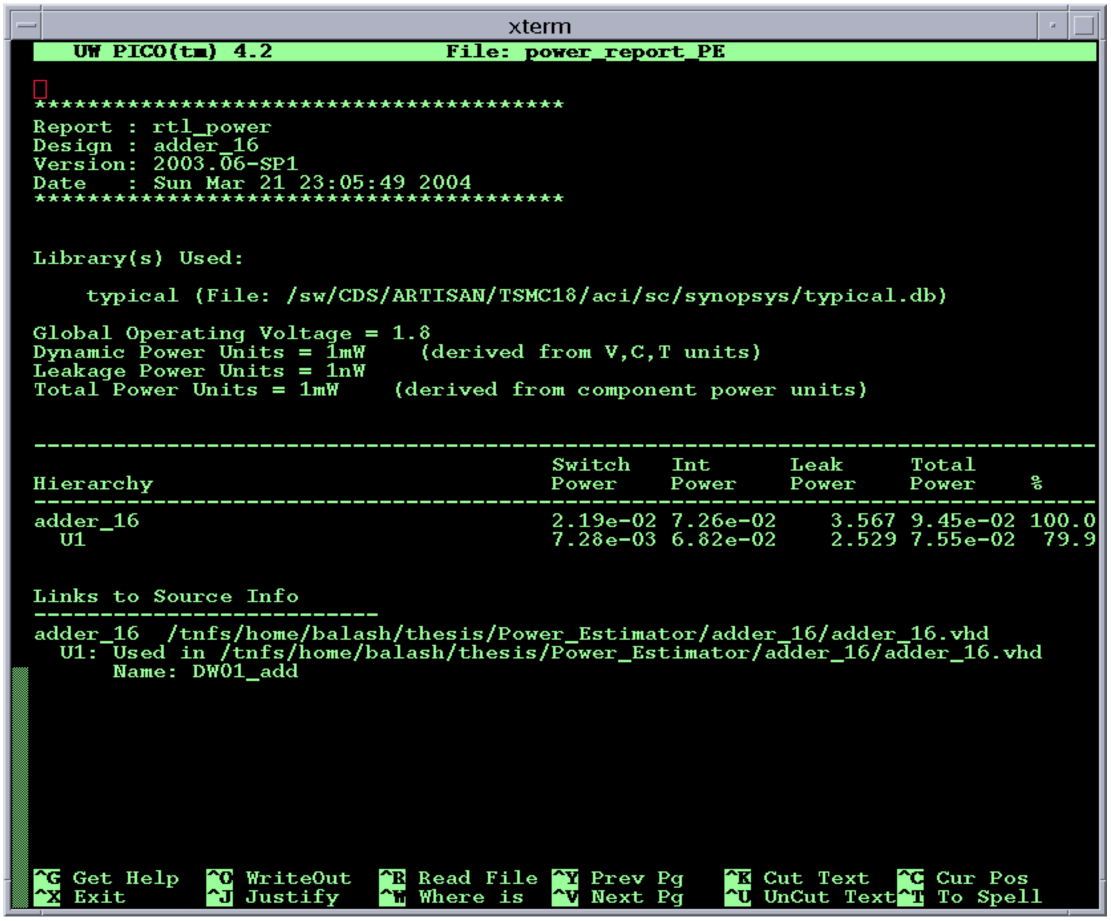
4.1 Power results – 16-bit Adder

The following section provides the results obtained for 16-bit adder macro developed for Best-PDA. The simulation is done for 1024 input vectors, running at frequency of 20 MHz (50 ns for 51200 ns). Simulation results of adder using *ModelSim* along with the power reports from *Power Estimator*, *Power Compiler*, *PrimePower* and *NanoSim* are shown in Figures 4.2 – 4.8. Waveform of the power result obtained using *CosmoScope* after running *PrimePower* is also shown in Figure 4.6.



```
/source_a...  9486    C000   E000   F000   F800
/source_a...  9486    C000   E000   F000   F800
/source_a...  1290C   18000  1C000  1E000  1F000
Now     2100 ns       150    200    250    300
Cursor 1   0 ns
135 ns to 332 ns
```

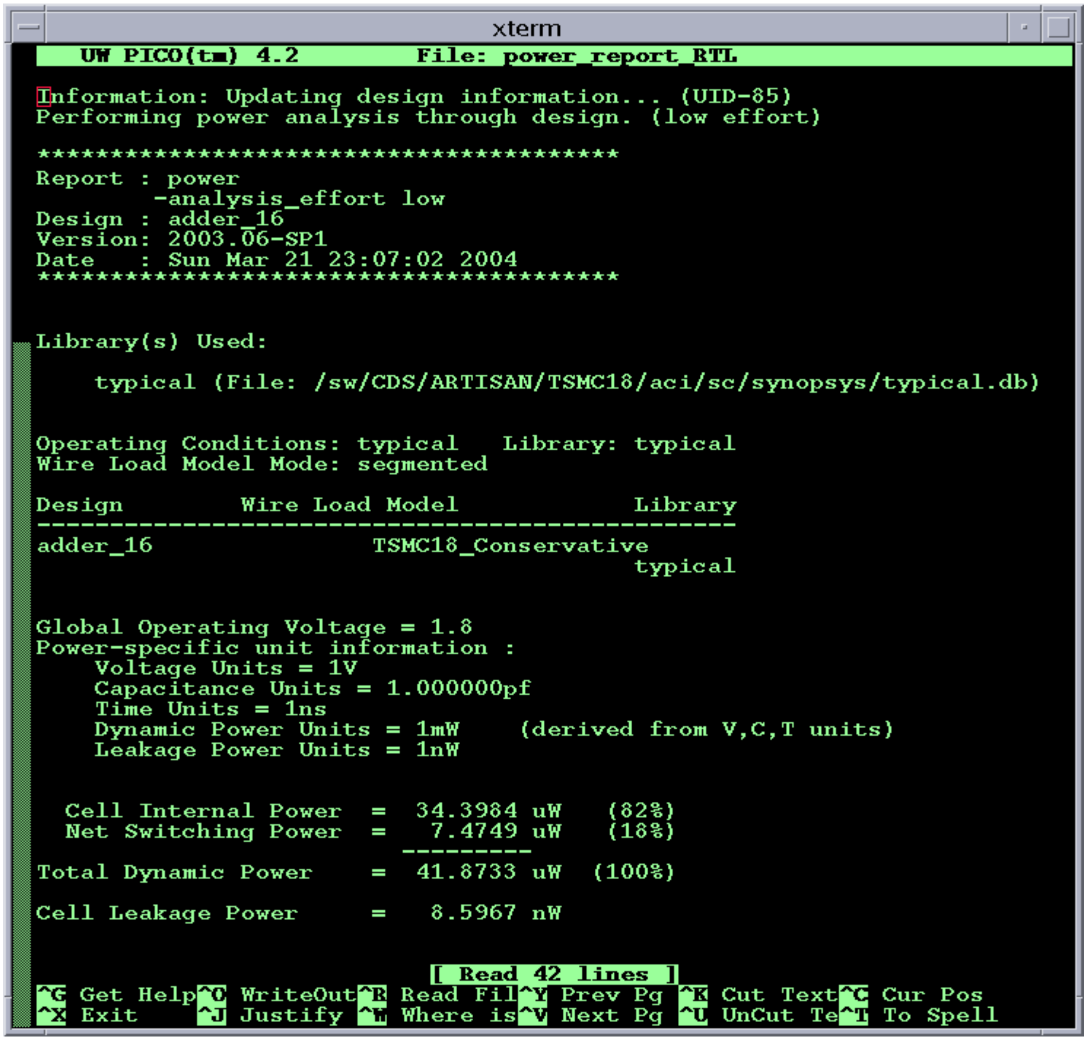
Figure 4.2 ***ModelSim*** **Simulation result for 16-bit adder**



```
  UW PICO(tm) 4.2               File: power_report_PE

****************************************
Report : rtl_power
Design : adder_16
Version: 2003.06-SP1
Date   : Sun Mar 21 23:05:49 2004
****************************************


Library(s) Used:

    typical (File: /sw/CDS/ARTISAN/TSMC18/aci/sc/synopsys/typical.db)

Global Operating Voltage = 1.8
Dynamic Power Units = 1mW    (derived from V,C,T units)
Leakage Power Units = 1nW
Total Power Units = 1mW    (derived from component power units)


--------------------------------------------------------------------------------
                                       Switch   Int      Leak     Total
Hierarchy                              Power    Power    Power    Power    %
--------------------------------------------------------------------------------
adder_16                               2.19e-02 7.26e-02    3.567 9.45e-02 100.0
  U1                                   7.28e-03 6.82e-02    2.529 7.55e-02  79.9


Links to Source Info
--------------------------
adder_16  /tnfs/home/balash/thesis/Power_Estimator/adder_16/adder_16.vhd
  U1: Used in /tnfs/home/balash/thesis/Power_Estimator/adder_16/adder_16.vhd
      Name: DW01_add
```

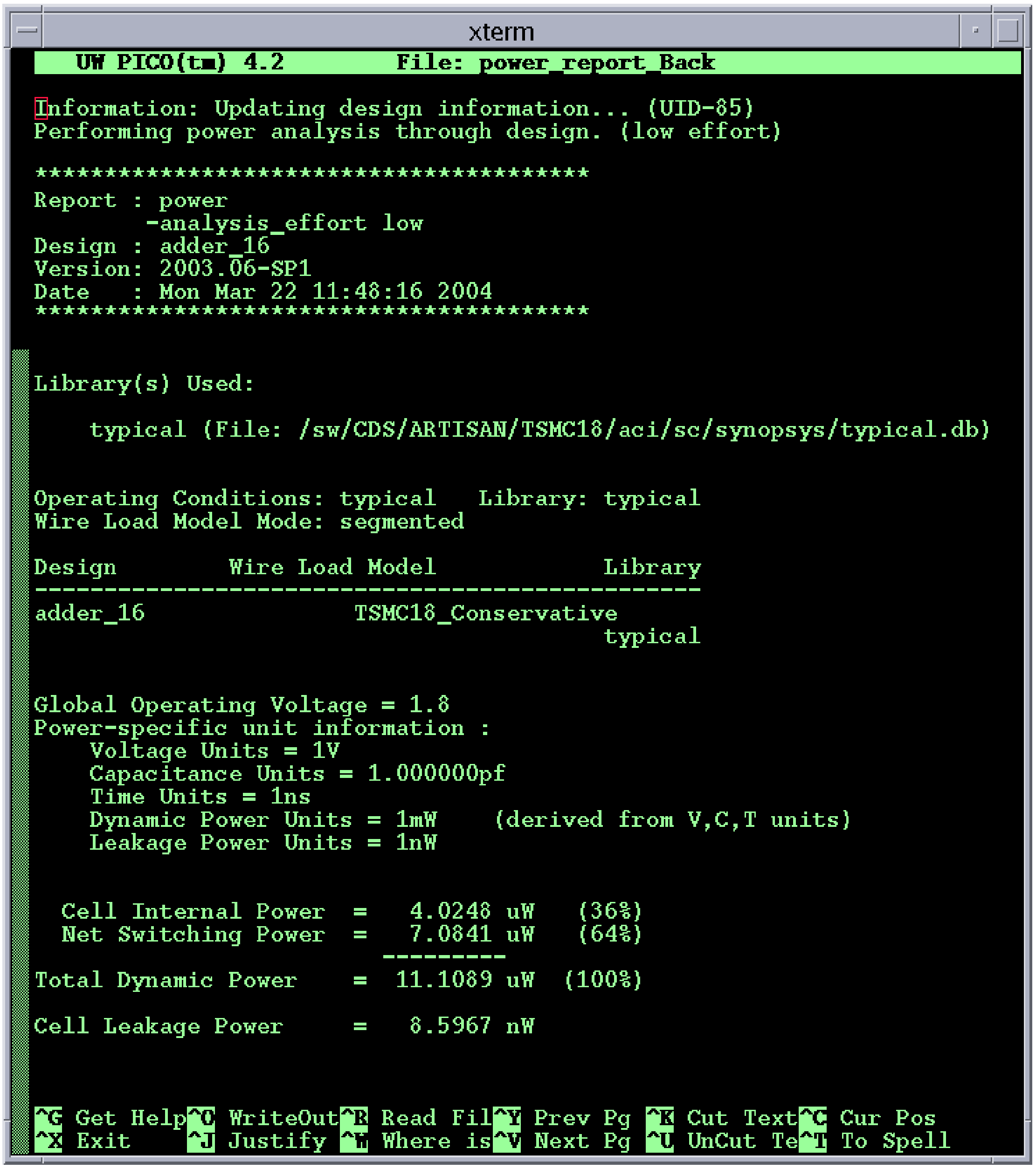
^G Get Help   ^O WriteOut   ^R Read File   ^Y Prev Pg   ^K Cut Text   ^C Cur Pos
^X Exit   ^J Justify   ^W Where is   ^V Next Pg   ^U UnCut Text   ^T To Spell

Figure 4.3 Power Report using *Power Estimator*



```
  UW PICO(tm) 4.2          File: power_report_RTL

Information: Updating design information... (UID-85)
Performing power analysis through design. (low effort)

****************************************
Report : power
        -analysis_effort low
Design : adder_16
Version: 2003.06-SP1
Date   : Sun Mar 21 23:07:02 2004
****************************************


Library(s) Used:

    typical (File: /sw/CDS/ARTISAN/TSMC18/aci/sc/synopsys/typical.db)


Operating Conditions: typical   Library: typical
Wire Load Model Mode: segmented

Design          Wire Load Model            Library
------------------------------------------------
adder_16                 TSMC18_Conservative
                                           typical


Global Operating Voltage = 1.8
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  =  34.3984 uW   (82%)
  Net Switching Power  =   7.4749 uW   (18%)
                         ---------
Total Dynamic Power    =  41.8733 uW  (100%)

Cell Leakage Power     =   8.5967 nW


                           [ Read 42 lines ]
^G Get Help^O WriteOut^R Read Fil^Y Prev Pg ^K Cut Text^C Cur Pos
^X Exit    ^J Justify ^W Where is^V Next Pg ^U UnCut Te^T To Spell
```

Figure 4.4 Power Report using *Power Compiler* with RTL Switching Activity



```
  UW PICO(tm) 4.2          File: power_report_Back

Information: Updating design information... (UID-85)
Performing power analysis through design. (low effort)

****************************************
Report : power
        -analysis_effort low
Design : adder_16
Version: 2003.06-SP1
Date   : Mon Mar 22 11:48:16 2004
****************************************


Library(s) Used:

    typical (File: /sw/CDS/ARTISAN/TSMC18/aci/sc/synopsys/typical.db)


Operating Conditions: typical   Library: typical
Wire Load Model Mode: segmented

Design        Wire Load Model            Library
------------------------------------------------
adder_16                TSMC18_Conservative
                                         typical


Global Operating Voltage = 1.8
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  =   4.0248 uW   (36%)
  Net Switching Power  =   7.0841 uW   (64%)
                         ---------
Total Dynamic Power    =  11.1089 uW  (100%)

Cell Leakage Power     =   8.5967 nW



^G Get Help^O WriteOut^R Read Fil^Y Prev Pg ^K Cut Text^C Cur Pos
^X Exit    ^J Justify ^W Where is^V Next Pg ^U UnCut Te^T To Spell
```

Figure 4.5 Power Report using *Power Compiler* with Gate-level Switching Activity

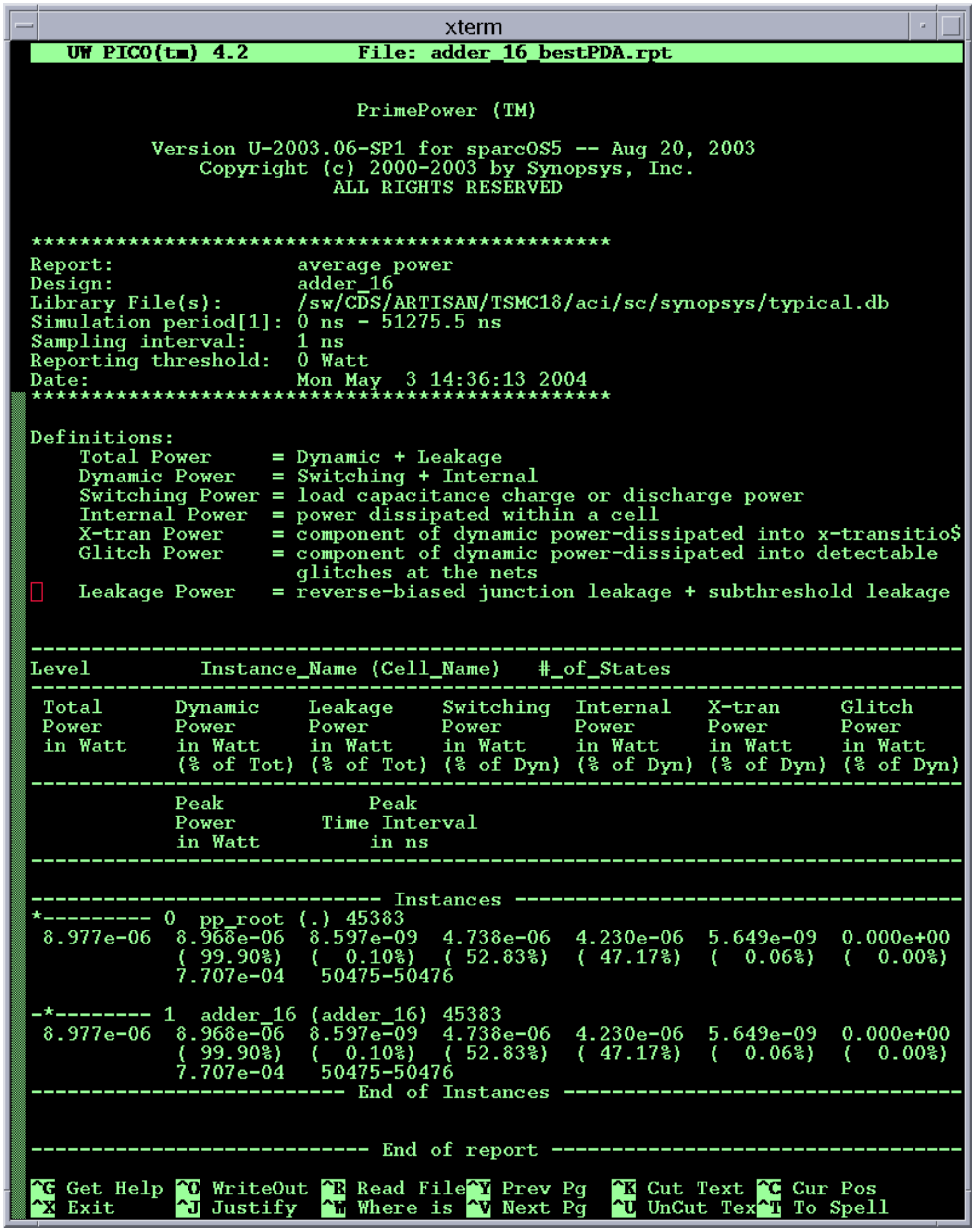

```
xterm
  UW PICO(tm) 4.2          File: adder_16_bestPDA.rpt


                              PrimePower (TM)

          Version U-2003.06-SP1 for sparcOS5 -- Aug 20, 2003
             Copyright (c) 2000-2003 by Synopsys, Inc.
                       ALL RIGHTS RESERVED


************************************************
Report:                 average power
Design:                 adder_16
Library File(s):        /sw/CDS/ARTISAN/TSMC18/aci/sc/synopsys/typical.db
Simulation period[1]: 0 ns - 51275.5 ns
Sampling interval:      1 ns
Reporting threshold:    0 Watt
Date:                   Mon May  3 14:36:13 2004
************************************************

Definitions:
    Total Power     = Dynamic + Leakage
    Dynamic Power   = Switching + Internal
    Switching Power = load capacitance charge or discharge power
    Internal Power  = power dissipated within a cell
    X-tran Power    = component of dynamic power-dissipated into x-transitio$
    Glitch Power    = component of dynamic power-dissipated into detectable
                      glitches at the nets
    Leakage Power   = reverse-biased junction leakage + subthreshold leakage


-----------------------------------------------------------------------------
Level          Instance_Name (Cell_Name)    #_of_States
-----------------------------------------------------------------------------
 Total       Dynamic     Leakage     Switching   Internal    X-tran      Glitch
 Power       Power       Power       Power       Power       Power       Power
 in Watt     in Watt     in Watt     in Watt     in Watt     in Watt     in Watt
             (% of Tot)  (% of Tot)  (% of Dyn)  (% of Dyn)  (% of Dyn)  (% of Dyn)
-----------------------------------------------------------------------------
             Peak              Peak
             Power       Time Interval
             in Watt          in ns
-----------------------------------------------------------------------------

----------------------------- Instances -------------------------------------
*---------- 0  pp_root (.) 45383
 8.977e-06   8.968e-06   8.597e-09   4.738e-06   4.230e-06   5.649e-09   0.000e+00
             ( 99.90%)   (  0.10%)   ( 52.83%)   ( 47.17%)   (  0.06%)   (  0.00%)
             7.707e-04    50475-50476

-*--------- 1  adder_16 (adder_16) 45383
 8.977e-06   8.968e-06   8.597e-09   4.738e-06   4.230e-06   5.649e-09   0.000e+00
             ( 99.90%)   (  0.10%)   ( 52.83%)   ( 47.17%)   (  0.06%)   (  0.00%)
             7.707e-04    50475-50476
-------------------------- End of Instances ---------------------------------


---------------------------- End of report ----------------------------------

^G Get Help ^O WriteOut ^R Read File^Y Prev Pg  ^K Cut Text ^C Cur Pos
^X Exit     ^J Justify  ^W Where is ^V Next Pg  ^U UnCut Tex^T To Spell
```

Figure 4.6 Power Report using *PrimePower*

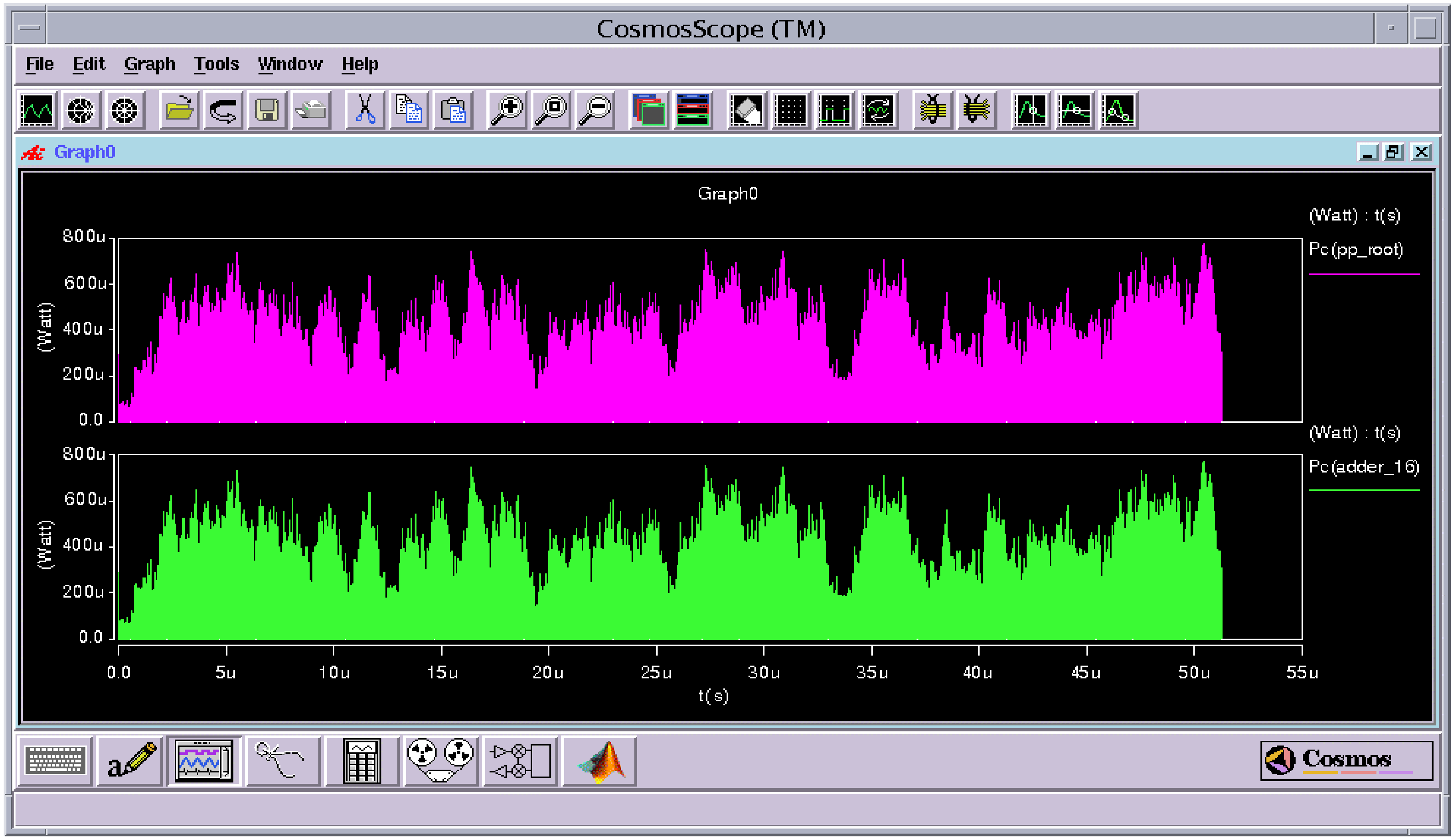
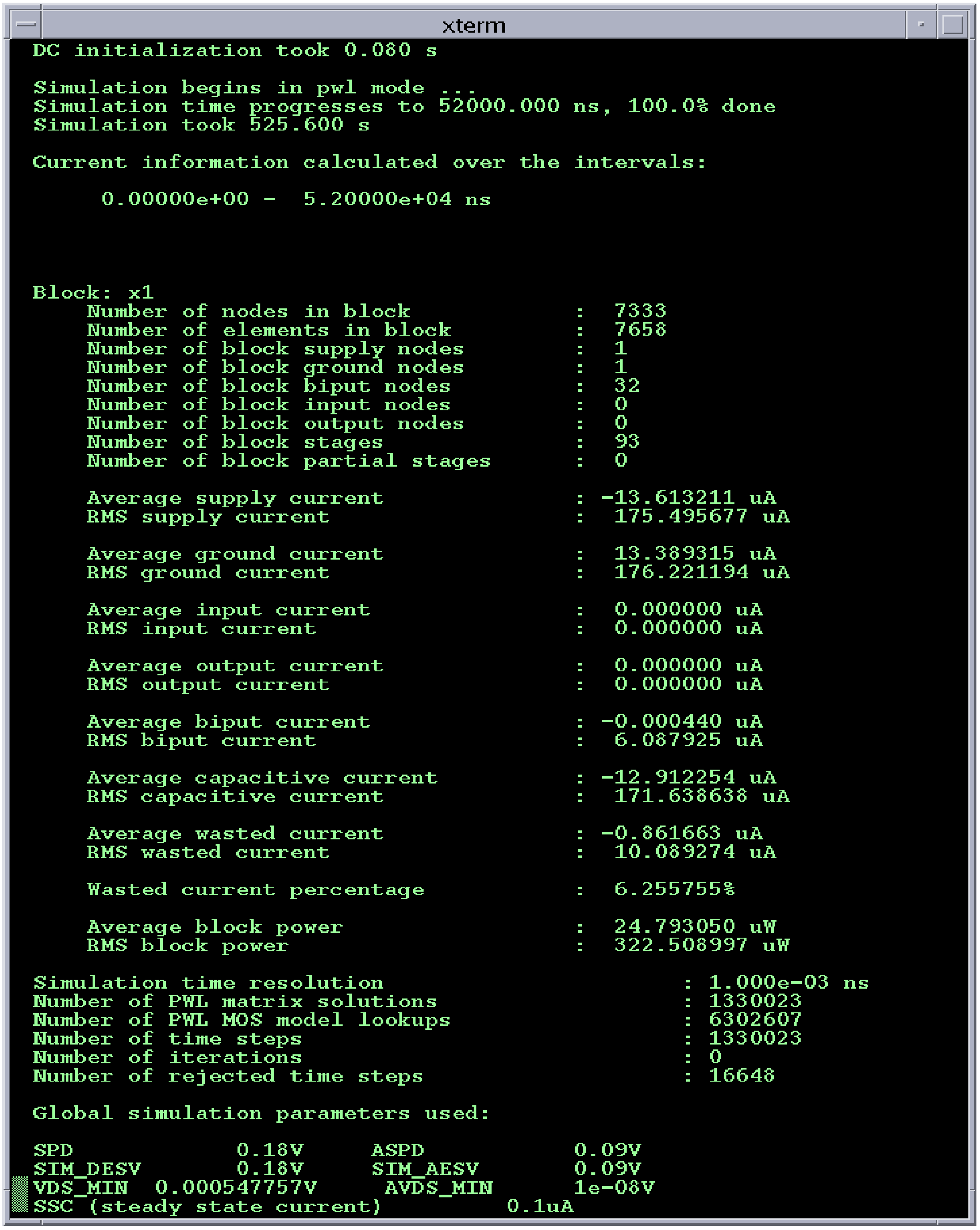


Figure 4.7 Power Graph using *CosmoScope*



xterm

```
DC initialization took 0.080 s

Simulation begins in pwl mode ...
Simulation time progresses to 52000.000 ns, 100.0% done
Simulation took 525.600 s

Current information calculated over the intervals:

     0.00000e+00 -  5.20000e+04 ns



Block: x1
    Number of nodes in block            :  7333
    Number of elements in block         :  7658
    Number of block supply nodes        :  1
    Number of block ground nodes        :  1
    Number of block biput nodes         :  32
    Number of block input nodes         :  0
    Number of block output nodes        :  0
    Number of block stages              :  93
    Number of block partial stages      :  0

    Average supply current              : -13.613211 uA
    RMS supply current                  :  175.495677 uA

    Average ground current              :  13.389315 uA
    RMS ground current                  :  176.221194 uA

    Average input current               :  0.000000 uA
    RMS input current                   :  0.000000 uA

    Average output current              :  0.000000 uA
    RMS output current                  :  0.000000 uA

    Average biput current               : -0.000440 uA
    RMS biput current                   :  6.087925 uA

    Average capacitive current          : -12.912254 uA
    RMS capacitive current              :  171.638638 uA

    Average wasted current              : -0.861663 uA
    RMS wasted current                  :  10.089274 uA

    Wasted current percentage           :  6.255755%

    Average block power                 :  24.793050 uW
    RMS block power                     :  322.508997 uW

Simulation time resolution                      : 1.000e-03 ns
Number of PWL matrix solutions                  : 1330023
Number of PWL MOS model lookups                 : 6302607
Number of time steps                            : 1330023
Number of iterations                            : 0
Number of rejected time steps                   : 16648

Global simulation parameters used:

SPD            0.18V     ASPD           0.09V
SIM_DESV       0.18V     SIM_AESV       0.09V
VDS_MIN  0.000547757V     AVDS_MIN      1e-08V
SSC (steady state current)         0.1uA
```

Figure 4.8   Power Report using *NanoSim*

As shown in the Figures, the following are the power values obtained using different power tools for Best_PDA Adder.

a) *Power Estimator* – 0.094 mw

b) *Power Compiler* with RTL Switching Activity – 0.046 mw

c) *Power Compiler* with Gate-level Switching Activity - 0.011 mw

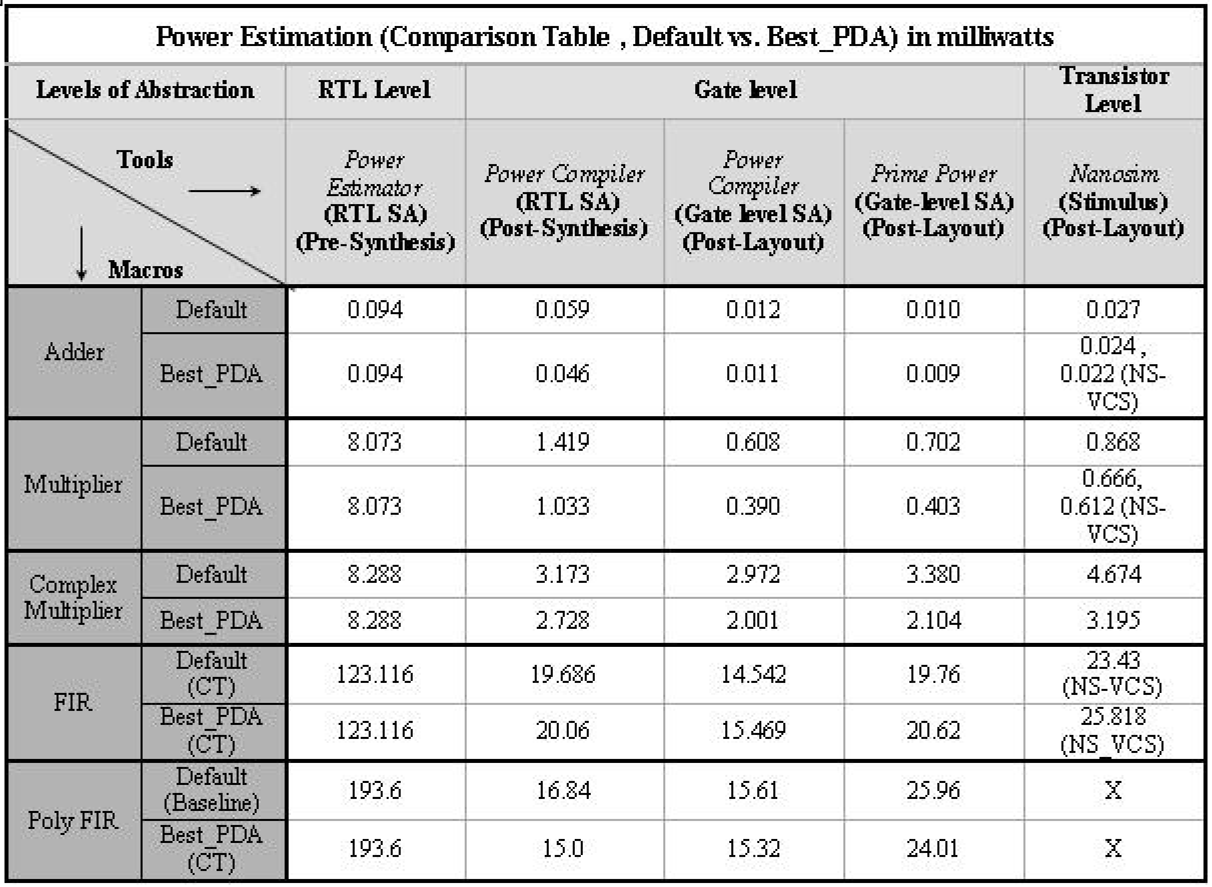
d) *PrimePower* - 0.009 mw

e) *NanoSim* – 0.024 mw

4.2 Power Tables

Similarly, power reports for other macros are also determined and values reported in a power table. There are three main power tables created to summarize the values obtained from the power tools used in this thesis. Table 4.1 shows the values obtained each of these tools for Default and Best_PDA version of the macros considered in this thesis. The three levels of abstraction, Tools used and Type of simulation performed is mentioned in the table. Table 4.2 shows the ratios of power values between Default and Best-PDA macros. This table is used to show consistency in the values of Default and Best_PDA for each of the macros. Table 4.3 shows simulation time taken for Default vs. Best–PDA macros using *NanoSim* tool.

Table 4.1 Comparison Table between Default vs. Best-PDA macros



Power Estimation (Comparison Table , Default vs. Best_PDA) in milliwatts						
Levels of Abstraction		**RTL Level**	**Gate level**			**Transistor Level**
Tools → ↓ **Macros**		*Power Estimator* **(RTL SA) (Pre-Synthesis)**	*Power Compiler* **(RTL SA) (Post-Synthesis)**	*Power Compiler* **(Gate level SA) (Post-Layout)**	*Prime Power* **(Gate-level SA) (Post-Layout)**	*Nanosim* **(Stimulus) (Post-Layout)**
Adder	Default	0.094	0.059	0.012	0.010	0.027
	Best_PDA	0.094	0.046	0.011	0.009	0.024 , 0.022 (NS-VCS)
Multiplier	Default	8.073	1.419	0.608	0.702	0.868
	Best_PDA	8.073	1.033	0.390	0.403	0.666, 0.612 (NS-VCS)
Complex Multiplier	Default	8.288	3.173	2.972	3.380	4.674
	Best_PDA	8.288	2.728	2.001	2.104	3.195
FIR	Default (CT)	123.116	19.686	14.542	19.76	23.43 (NS-VCS)
	Best_PDA (CT)	123.116	20.06	15.469	20.62	25.818 (NS_VCS)
Poly FIR	Default (Baseline)	193.6	16.84	15.61	25.96	X
	Best_PDA (CT)	193.6	15.0	15.32	24.01	X

Table 4.2 Default/ Best-PDA Ratio

Comparison of Default/ Best-PDA Ratio					
Levels of Abstraction	**RTL Level**	**Gate level**			**Transistor Level**
Tools (SA) → **Macros** ↓	*Power Estimator* **(RTL SA) (Pre-Synthesis)**	*Power Compiler* **(RTL SA) (Post-Synthesis)**	*Power Compiler* **(Gate- level SA) (Post-Layout)**	*Prime Power* **(Gate-level SA) (Post-Layout)**	*Nanosim* **(Stimulus) (Post-Layout)**
Adder	1	1.28	1.09	1.11	1.12
Multiplier	1	1.37	1.56	1.74	1.30
Complex Multiplier	1	1.16	1.49	1.61	1.46
FIR	1	0.98	0.94	0.96	0.91
Poly FIR	1	1.12	1.02	1.08	X

Table 4.3 Simulation time taken for macros using *NanoSim*

Simulation Time (in hrs) using *Nanosim*		
Tool → / **Macros ↓**		**Time**
Adder	Default	0.38
	Best_PDA	0.36
Multiplier	Default	16.29
	Best_PDA	12.56
Complex Multiplier	Default	81.29
	Best_PDA	46.66
FIR	Default	678.85
	Best_PDA	462.69
Poly FIR	Default	X
	Best_PDA	X

4.3 Power Graphs

This section summarizes all the comparison power values as Bar graphs for better understanding. Fig 4.9 – 4.13 gives the bar graph for each of the macros.

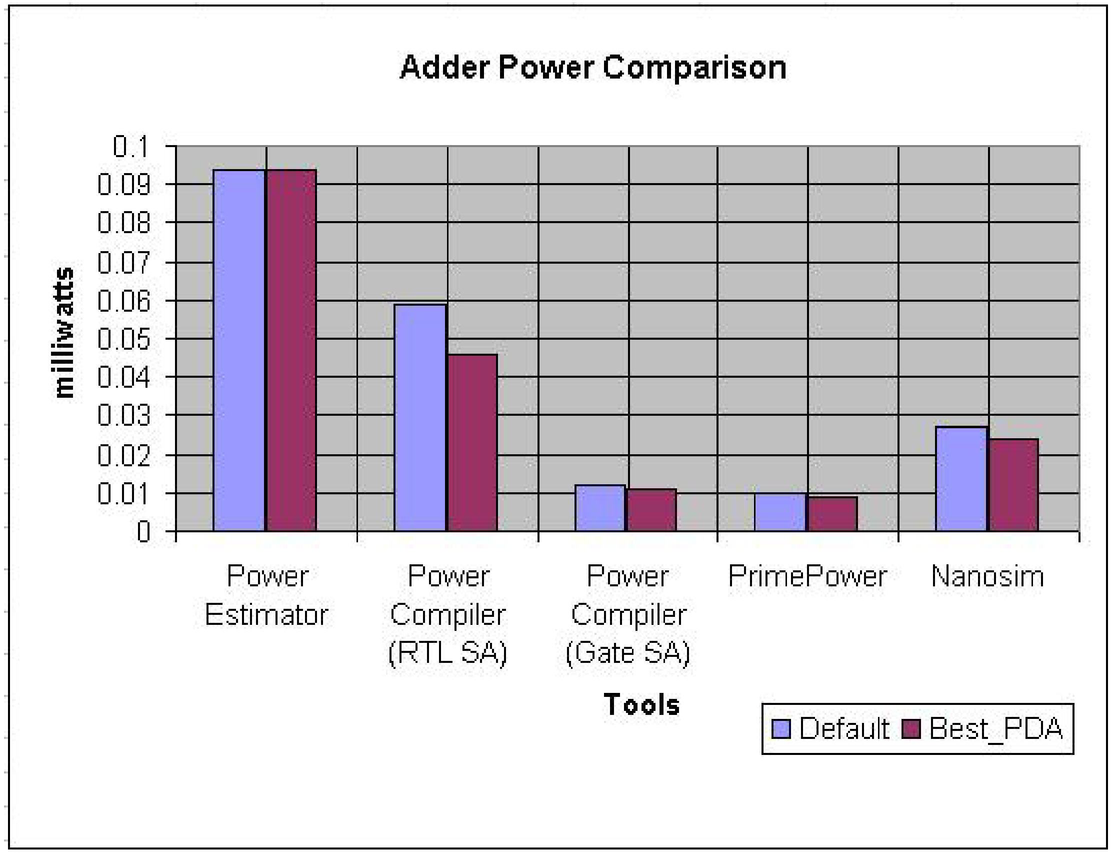


Figure 4.9 Comparison between Adder macro's Default and Best_PDA power values

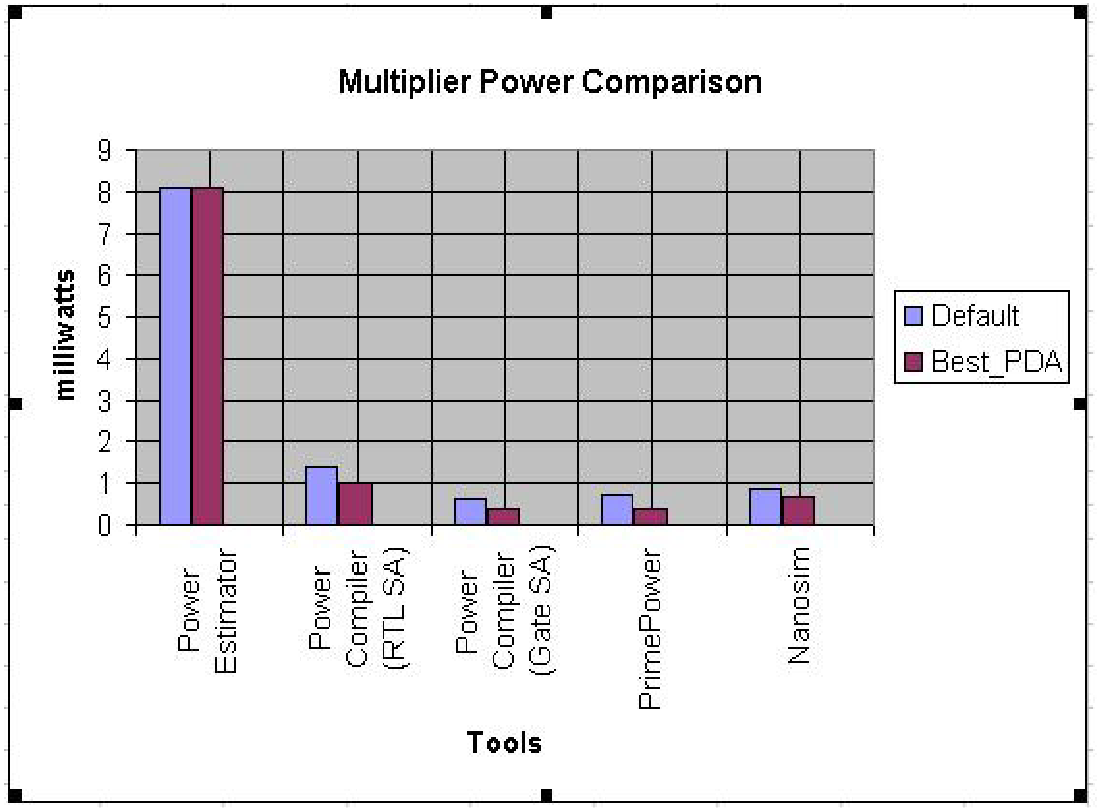


Figure 4.10 Comparison between Multiplier macro's Default and Best_PDA power values

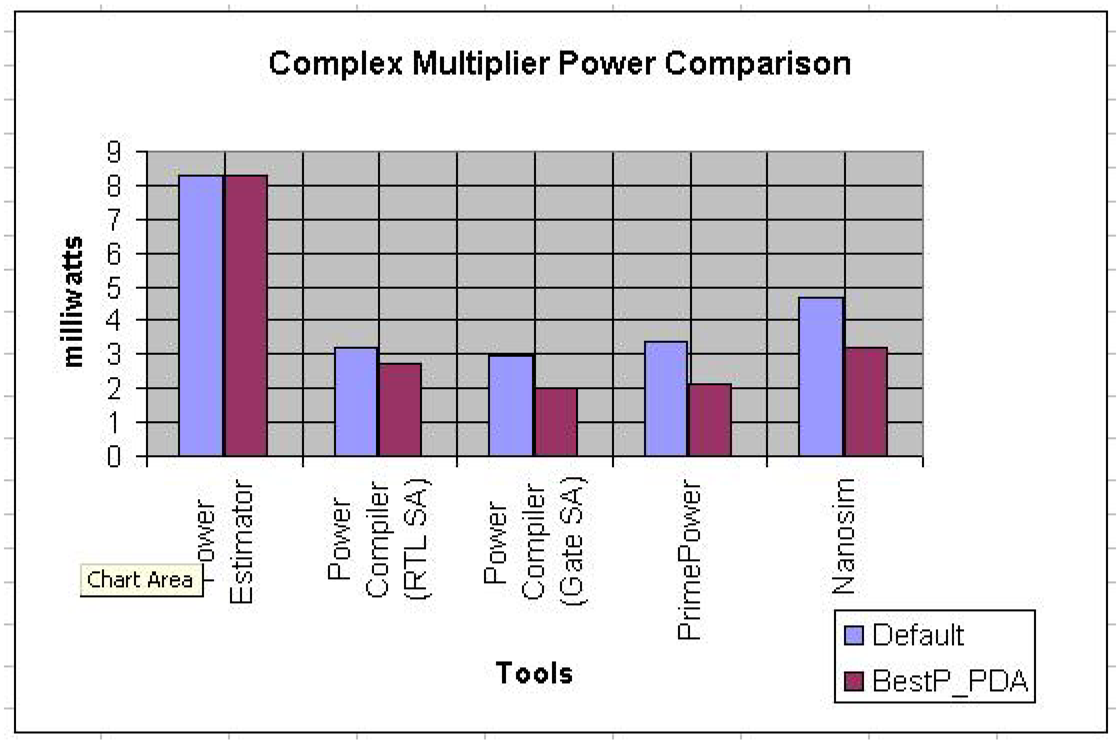


Figure 4.11 Comparison between Complex Multiplier macro's Default and Best_PDA power values

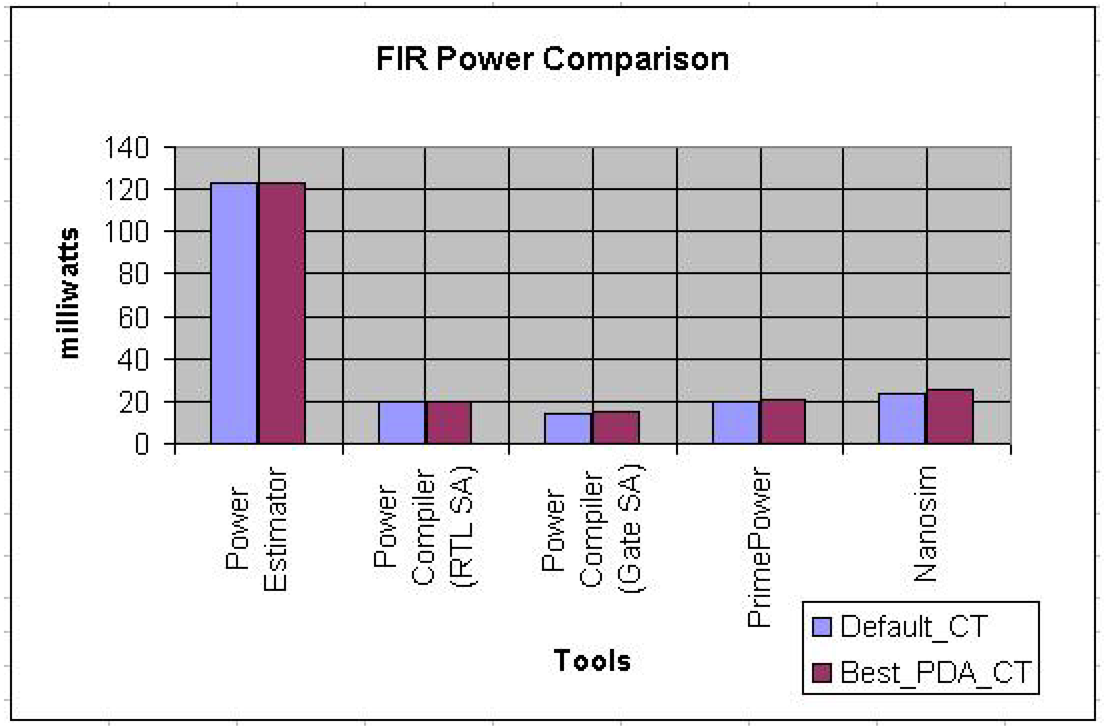


Figure 4.12 Comparison between FIR macro's Default and Best_PDA power values

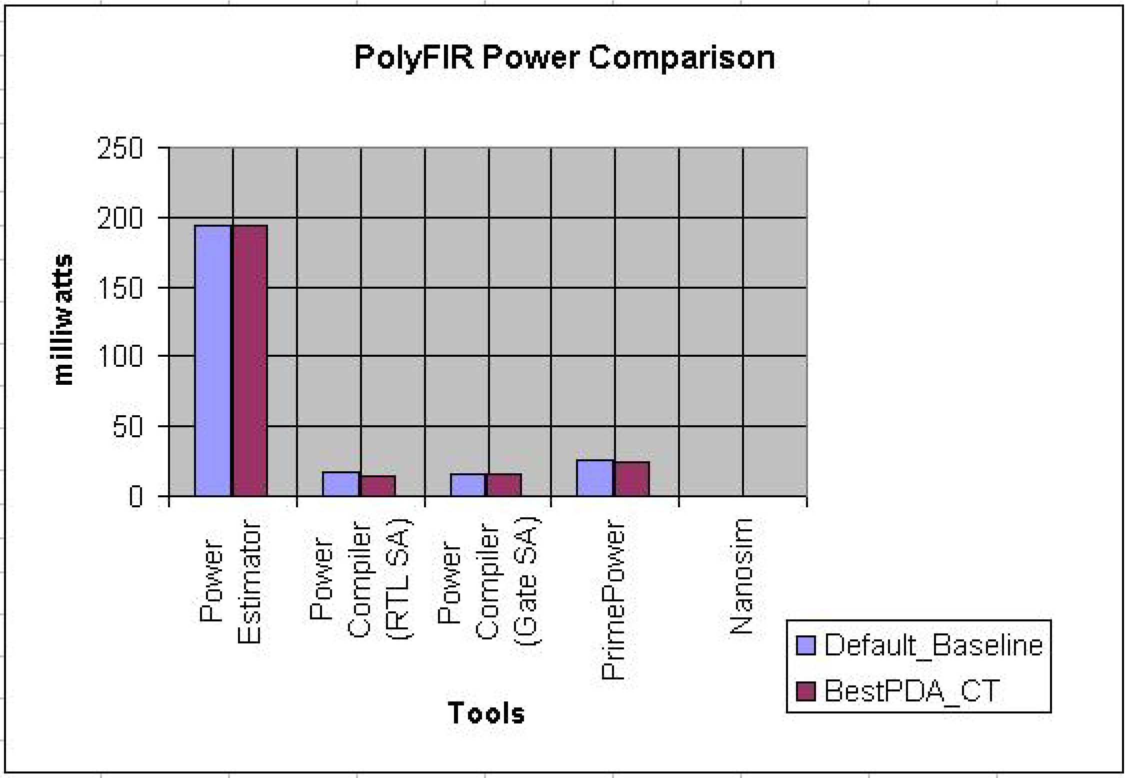


Figure 4.13 Comparison between Poly-FIR macro's Default and Best_PDA power values

4.4 Analysis of obtained power values

This section discusses the percentage of error in the power values between the different tools using Table 4.1 assuming the value obtained using *Nanosim* to be the most accurate value or the one closer to reality. Percentage of error is calculated using the following formula: **(*NanoSim* value – Reported value from a specific tool) / *NanoSim* value.**

Table 4.4 shows the percentage of error for each of the macros both default and Best_PDA using the above formula. Table 4.5 shows the percentage of error using the Default vs. Best_PDA ratios. As seen in Table 4.4, there is a consistency in the error percentage for smaller circuits, Adder, Multiplier and Complex Multiplier compared to big circuits like FIR and Poly-FIR. As expected the power values using *Power Compiler* using Gate-level switching activity and *PrimePower* are nearly the same. The *Power Estimator* value shows a tremendous amount of deviation from NanoSim value showing how inaccurate it would be to report power at the RTL level. More than the percentage of error for the actual power values, we are interested how they differ when it comes to ratio of Default vs. Best_PDA values. The percentage of error moving down to lower abstraction level is not big compared to the error percentage as we see in the power value table.

Table 4.4 Percentage of Error on the actual power values for each macro

Percentage of Error considering *Nanosim* values to be most accurate values						
Levels of Abstraction		**RTL Level**	**Gate level**			**Transistor Level**
Tools → ↓ **Macros**		*Power Estimator* **(RTL SA) (Pre-Synthesis)**	*Power Compiler* **(RTL SA) (Post-Synthesis)**	*Power Compiler* **(Gate level SA) (Post-Layout)**	*Prime Power* **(Gate-level SA) (Post-Layout)**	*Nanosim* **(Stimulus) (Post-Layout)**
Adder	Default	248	119	- 56	-63	0
	Best_PDA	292	92	- 54	- 63	0
Multiplier	Default	830	63	- 30	- 19	0
	Best_PDA	1112	55	- 41	- 39	0
Complex Multiplier	Default	77	32	- 36	- 28	0
	Best_PDA	159	15	- 37	- 34	0
FIR	Default (CT)	425	- 16	- 38	- 16	0
	Best_PDA (CT)	377	- 22	- 40	- 20	0
Poly FIR	Default (Baseline)	X	X	X	X	X
	Best_PDA	X	X	X	X	X

A "+"sign indicates the reported value from that specific tool to be greater than *Nanosim* value
A "–"sign indicates the reported value from that specific tool to be lesser than *Nanosim* value

Table 4.5 Percentage of Error on the Ratios of Default vs. Best_PDA for each macro

Percentage of Error considering *Nanosim* values to be most accurate values					
Levels of Abstraction	**RTL Level**	**Gate level**			**Transistor Level**
Tools (SA) → / **Macros** ↓	*Power Estimator* **(RTL SA) (Pre-Synthesis)**	*Power Compiler* **(RTL SA) (Post-Synthesis)**	*Power Compiler* **(Gate- level SA) (Post-Layout)**	*Prime Power* **(Gate-level SA) (Post-Layout)**	*Nanosim* **(Stimulus) (Post-Layout)**
Adder	- 22.7	14.2	- 2.6	- 12.8	0
Multiplier	- 23.0	5.3	20	33.8	0
Complex Multiplier	- 31.5	- 20.5	2	10.2	0
FIR	9.8	7.6	3.2	5.4	0
Poly FIR	X	X	X	X	X

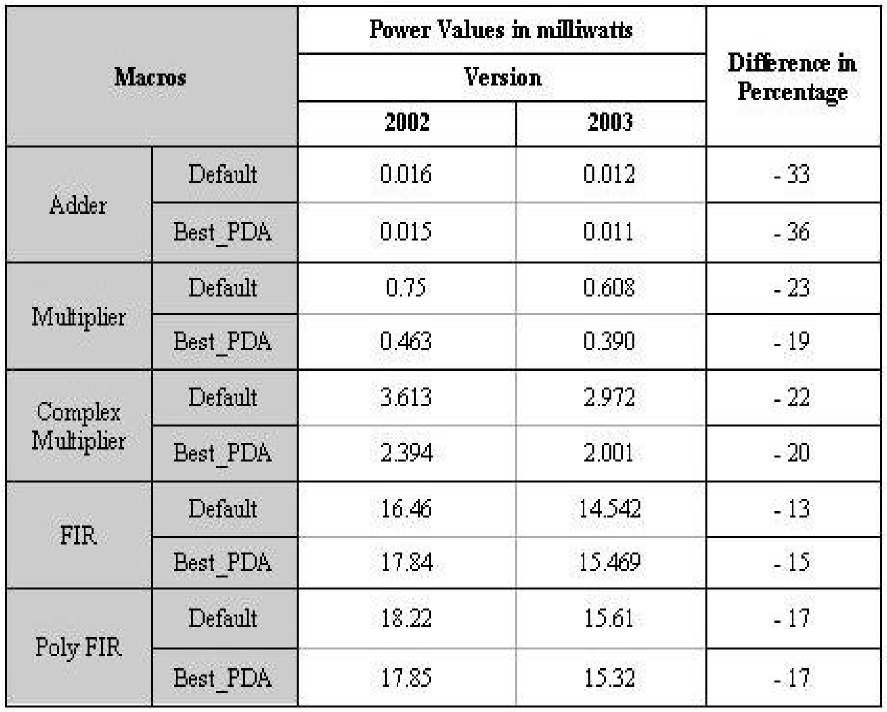
A "+"sign indicates the reported value from that specific tool to be greater than *Nanosim* value
A "–"sign indicates the reported value from that specific tool to be lesser than *Nanosim* value

4.5 Power value difference between Tool versions

More than the difference in the power values between different power tools, there is also a significant difference between different versions of the same tool. This makes mentioning the version of the tool during any power reporting, very important. A comparison table has been created showing difference in power values using versions 2002 vs. 2003 for both *Power Compiler* and *PrimePower*. Table 4.6 shows the power values between 2002 and 2003 version of *Power Compiler*. Lower value in power of 2003 compared to its predecessor version shows how the algorithm of the tool has been improved together with the usage of a different database file used for power calculation. Table 4.7 shows the power values between 2002 and 2003 version of *PrimePower*. 2003 version of the tool gives higher values compared to 2002 version. This shows a better power reporting algorithm used in the latest version of *PrimePower*. An important point to be noted is that latest version of the tool need not necessarily give a lower value. It shows the accuracy in power estimation as the tool improves. The percentage difference between these versions are also calculated and tabulated. The formula used to calculate the percentage of difference is

(2003 version – 2002 version) / (2003 version)

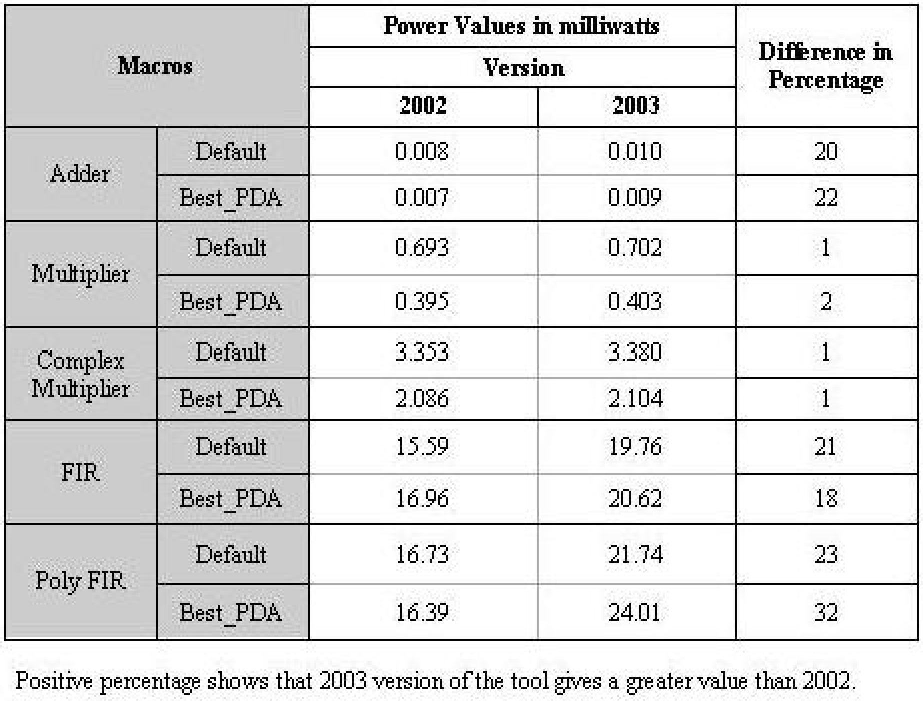
Table 4.6 ***Power Compiler's*** **2002 vs. 2003 version**



Macros		Power Values in milliwatts		Difference in Percentage
		Version		
		2002	2003	
Adder	Default	0.016	0.012	- 33
	Best_PDA	0.015	0.011	- 36
Multiplier	Default	0.75	0.608	- 23
	Best_PDA	0.463	0.390	- 19
Complex Multiplier	Default	3.613	2.972	- 22
	Best_PDA	2.394	2.001	- 20
FIR	Default	16.46	14.542	- 13
	Best_PDA	17.84	15.469	- 15
Poly FIR	Default	18.22	15.61	- 17
	Best_PDA	17.85	15.32	- 17

A "-"sign indicates that 2003 version of the tool gives a lesser value than 2002.

Table 4.7 PrimePower's 2002 vs. 2003 version



Macros		Power Values in milliwatts		Difference in Percentage
		Version		
		2002	2003	
Adder	Default	0.008	0.010	20
	Best_PDA	0.007	0.009	22
Multiplier	Default	0.693	0.702	1
	Best_PDA	0.395	0.403	2
Complex Multiplier	Default	3.353	3.380	1
	Best_PDA	2.086	2.104	1
FIR	Default	15.59	19.76	21
	Best_PDA	16.96	20.62	18
Poly FIR	Default	16.73	21.74	23
	Best_PDA	16.39	24.01	32

Positive percentage shows that 2003 version of the tool gives a greater value than 2002.

The following Table 4.8 shows the version of the different power tools used in this project.

Table 4.8 Tool versions

Tool Versions	
Tool	**Version**
Design Compiler	2003.06 SP1
PrimePower	2003.06 SP1
NanoSim	2003.03-SP2
VCS	7.0.2
Cadence ICFB	4.4.6
Silicon Ensemble	05.30-s173
Calibre	9.3_1.1
HyperExtract	4.5.0
ModelSim	SE vsim 5.7d

4.6 Discussion about *PrimePower* vs. *NanoSim* power calculations

A study has been made to compare the power results between *PrimePower* and *NanoSim* as these tools give the closest power to reality. In the case of the adder, the standard cell ADDFX2 has been used multiple times to realize the function of a 16-bit adder. A 4 bit adder was first drawn using this standard cell and the entire path from schematic creation to power calculation using *PrimePower* and *NanoSim* was performed. 20 input vectors were used to test the power using *PrimePower* and *NanoSim* and the following table lists the values.

Table 4.9 4-bit adder power values using *PrimePower* and *NanoSim*

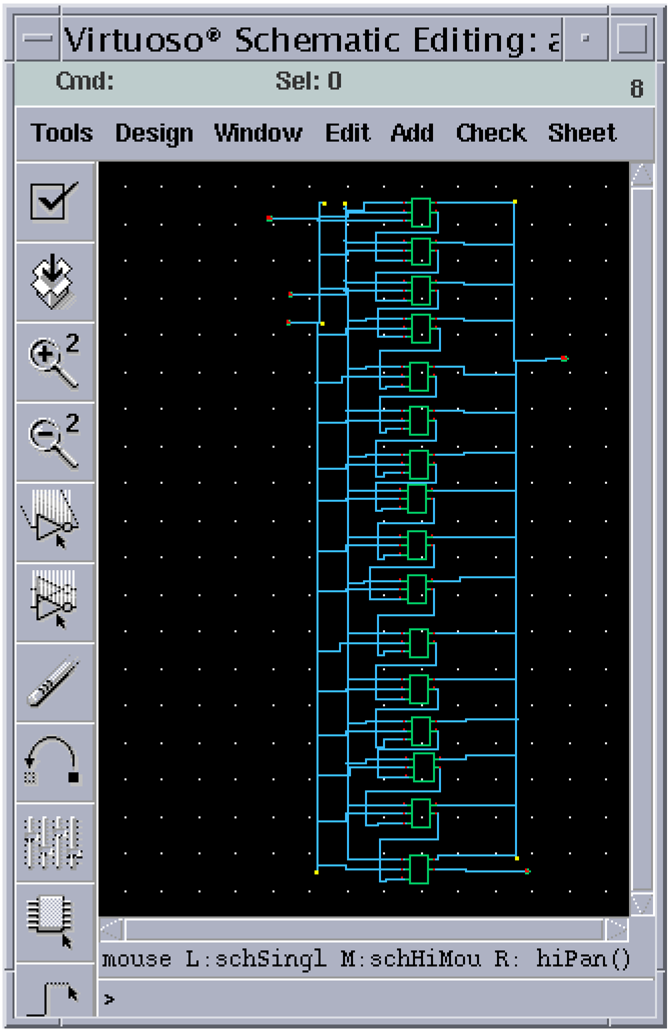
4-bit Adder power estimation for 20 input vectors	
Tool used	**Power Estimation**
PrimePower	5.424 µw
NanoSim – VCS	5.611 µw
NanoSim	5.901 µw

As shown in the table, the power values for a 4-bit adder between *PrimePower* and *NanoSim* are nearly the same. Likewise, the whole path from schematic creation to power calculation using *PrimePower* and *NanoSim* has been performed for 16-bit adder constructed using ADDFX2.

4.6.1 Steps involved in calculating power using ***PrimePower*** and ***NanoSim***

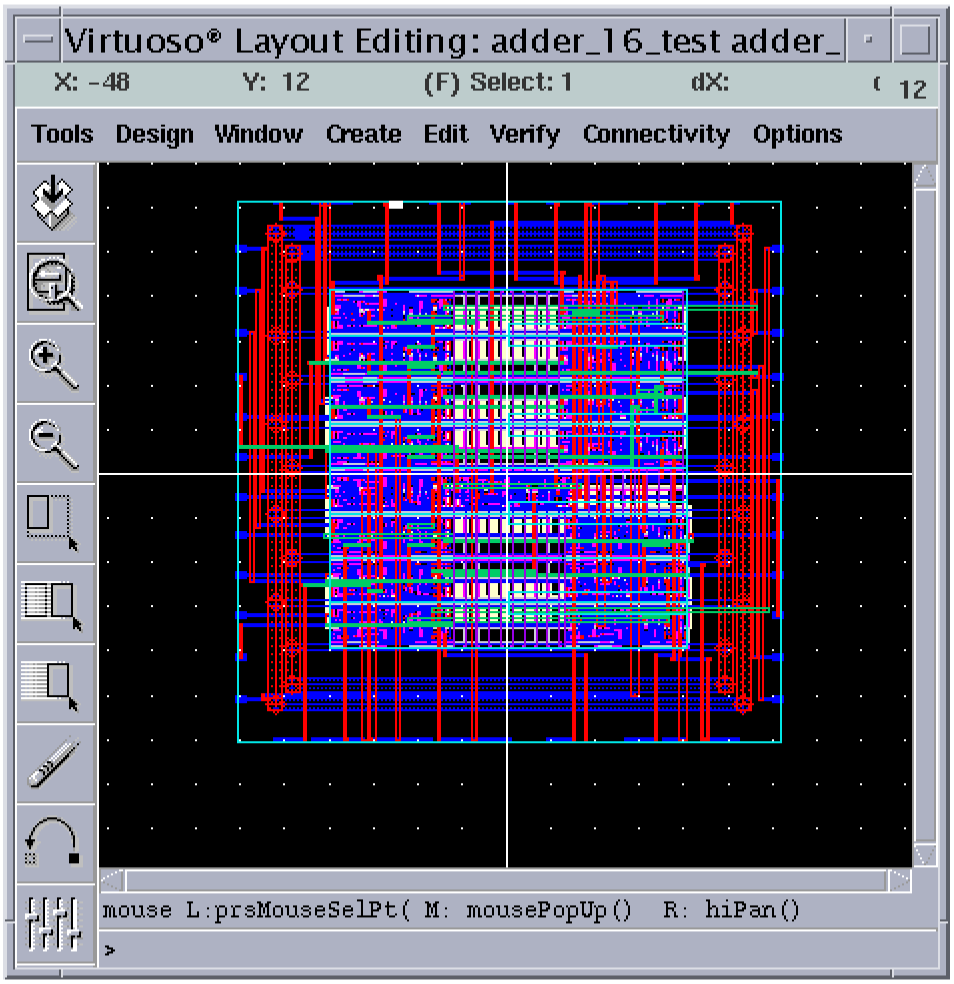
First using the ADDFX2 symbol, a 16-bit schematic has been drawn as shown in Figure 4.14. Gate-level net-list is obtained using a simulator. The gate-level net-list, tsmc18.v file (the Verilog file containing descriptions of all the standard cells of tsmc18 library) along with the tsmc18's LEF file, were imported into *Silicon Ensemble* to do the place and routing. Once the place and routing was done, *HyperExtract* was used to get the parasitic values for the entire circuit. Then DEF and Standard Delay Format file were exported from *Silicon Ensemble*. The DEF file was used in the ICFB environment to get the layout which is shown in Figure 4.15. Once the layout has been generated, *Calibre* was used to extract the circuit to get the SPICE net-list for the 16-bit adder.

Now power calculation using *PrimePower* and *NanoSim* were performed for different input vectors for comparison purposes. The inputs required for *PrimePower* were Testbench to get the VCD file, DSPF file using *HyperExtact* and the Gate-level net-list. Using these as the input, power value for 16-bit adder was estimated. The SPICE net-list along with input stimulus was used to estimate the power using *NanoSim*.



```
mouse L:schSingl M:schHiMou R: hiPan()
>
```

Figure 4.14 16-bit adder schematic using ADDFX2



mouse L:prsMouseSelPt(M: mousePopUp() R: hiPan()

>

Figure 4.15 16-bit adder Layout using ADDFX2

Table 4.10 lists the comparison values. The closeness of the values between *PrimePower* and *NanoSim* confirms that the procedure of estimating power using these tools for all the macros were correct.

Table 4.10 Comparison of power values for 16-bit adder using ADDFX2

Comparison of power values for 16-bit adder using ADDFX2			
Tool used	**Power Estimation**		
	100 vectors	**500 vectors**	**1024 vectors**
PrimePower	22.87 µw	25.28 µw	26.23 µw
NanoSim – VCS	25.23 µw	28.56 µw	29.93 µw
NanoSim	24.82 µw	27.46 µw	28.22 µw

5 Summary, Conclusions and Future Work

5.1 Summary

Power Estimation for different macros from RTL level to Transistor level using different power estimation tools has been performed. The methodology involving the usage of these tools at different levels of abstraction has been shown with examples. Scripts have been developed for each of these levels to automate the flow for each of the macros involved. A table has been formed documenting the results of power for each of the macros developed for best Power, Delay and Area and also a comparison table has also been formed comparing the power values between Default macros Vs. Best PDA macros.

5.2 Conclusions

It can be concluded from these power estimations at different levels of abstraction how inaccurate values at RTL are compared to Transistor level. The deviations in the measurement can be seen in the tables for each of the macros. Power estimation at that level is done mainly because we can get faster results and can be used to decide on optimizing the circuit depending on the specification. Secondly, a study has been made between the power values obtained between *PrimePower* and *NanoSim* because they give the most accurate results at the Gate and Transistor level respectively. The study proves that the methodology of calculating power is correct taking into example, ADDER circuit. There had been a lot of experiments conducted while performing power calculations using *NanoSim*. It can be concluded from those experiments that proper usage of *NanoSim* needs to be done keeping in the mind what circuit we are performing the simulations on, how much of accuracy is needed in the measurements and how fast we need the results. Careful understanding of all these are to be deployed to get the best results from the High-Speed circuit simulator. The tabulation of percentage of error tables shows how dramatic values can change from RTL to Transistor level for the actual values when compared to the Ratio of Default vs. Best_PDA values. Since the thesis is about power estimation, it would have been good to compare the values of Default vs. Best_Power macros than Best_PDA. PDA macros are developed so they have got the best Power, Area and Delay product. For bigger circuits this value has proved to be more than their Default macros. The power results obtained using *Power Estimator* (P1)*, Power Compiler* using RTL Switching Activity (P2), *Power Compiler* using Gate-Level Switching Activity (P3) and *PrimePower* (P4) used power characterized library provided by Artisan whereas except for the technology file from TSMC18, the rest of the inputs given to *NanoSim* to calculate P5 were provided by researchers at our Microelectronic Systems Laboratory.

Different versions of the same macros were developed as part of the DARPA project. They are Macro with Minimum Area, Macro with Minimum Power, Macro with Minimum Delay and Macro with Best Power, Area and Delay. It has been concluded after seeing FIR results in which Power for Best PDA macro was greater than the Baseline macro that we should have compared the power values between Default and Minimum Power macros. Power, Area and Delay being the three major constraints in designing digital circuits there are applications like tactical missile applications and other defense related projects that would require circuits to be kept in a smaller area, dissipate power and perform really fast. Keeping this mind checking power in a Best_PDA macro is afterall useful to be implemented in these devices.

Some other conclusions are since power values are dependent not only the macro but also on the tools used, versions of the tool, power characterized library used, the input stimulus used and what the output load is, it would be only valid to compare results from different EDA power tools only if the above are identical. Seeing the large difference between power estimation at RTL level using *Power Estimator* and *NanoSim* at the Transistor level, it is concluded that *Power Estimator* values are useful only to start the power estimation flow from a top level abstraction. It would not be possible to get to any specific conclusion analyzing the results of *Power Estimator* other than trying to rewrite the RTL code (VHDL or Verilog) to get lesser power value as much as possible. The values obtained using *NanoSim* are considered to be the true values since all the inputs given to the tool contain fine information about the circuits and it would be logical to compare it with the testing of the real chips in the lab. The power values obtained by the other tools are used only to select the best possible netlist that are capable of giving less power when realized into real chips.

.

5.3 Future Work

As seen in the conclusion, the real power comparison should have been between Default vs. Best_Power macros. With the flow of power estimation already developed as part of this work, Timing analysis for macros can also be performed. Various comparisons of power estimation can be reported. Placement and Routing of the macros can be done using Synopsys tools as apposed to Cadence tools and power values obtained as the result of that can be compared. Other comparison like getting net-list using a different extraction tool can also be done. With the use of sophisticated power measuring equipments, each of the macros can be tested for power in real time for the same input vectors used here and values can be compared with the tool's estimation. For simulating bigger circuits with many input stimulus, *NanoSim* simulations can be run in parallel on multiple processors partitioning the stimulus for faster results. This method of speeding up is required after seeing that the results of FIR took around three weeks.

References

References

[1] W. Goh, S. Rofail, and K. Yeo, “Low-Power Design: An Overview”, Prentice Hall.

[2] D. Soudris, C. Piguet, and C. Goutis, ”Designing CMOS Circuits for Low Power”, Kluwer Academic Publishers, 2002.

[3] A. Bellaouar, and M. Elmasry, “Low-Power Digital VLSI Design: Circuits and Systems”, Kluwer Academic Publishers, 1995.

[4] F. Najm, “A Survey of Power Estimation Techniques in VLSI Circuits”, IEEE Transactions on VLSI Systems, vol. 2, pp. 446-455, 1994.

[5] J. Rabaey, “Digital Integrated Circuits: A Design Perspective”, Prentice Hall, 1996.

[6] N. H. E. Weste, and K. Eshraghian, “Principles of CMOS VLSI Design”, Addison Wesley, 1994.

[7] A. Chandrakasan, and R.W. Brodersen, “Low Power Digital CMOS Design”, Kluwer Academic Publishers, 1995.

[8] A. Raghunathan, N. K. Jha and S. Dey, “High-Level Power Analysis and Optimization” , Kluwer Academic Publishers, 1998.

[9] A. Chandrakasan, and R.W. Brodersen, “Minimizing Power Consumption in Digital CMOS Circuits”, Proceedings of IEEE 83(4), April 1995.

[10] D. Singh, J. Rabaey, M. Pedram, F. Catthoor, S.Rajgopal, N. Sehgal, and T. Mozdzen, “Power-conscious CAD tools and methodologies: a perspective,” Proceedings of the IEEE, pp. 570-594, April 1995.

[11] P. Landman, R. Mehra, and J. Rabaey, “An Integrated CAD Environment for Low-Power Design”, IEEE Design and Test of Computers, pp. 72-82, Summer 1996.

[12] P. Landman and J. Rabaey, “Power Estimation for High Level Synthesis”, In Proceedings of the European Design Automation Conference, pages 361–366, Paris, Feb. 1993.

[13] C-Y. Tsui, J. Monteiro, M. Pedram, S. Devadas, A. M.Despain, and B. Lin, “Power Estimation in Sequential Logic Circuits", IEEE Trans. on VLSI Systems , Vol. 3, No. 3, pp. 404-416, 1995.

[14] C. M. Huizer, "Power Dissipation Analysis of CMOS VLSI Circuits by means of Switch-Level Simulation", IEEE European Solid State Circuits Conf., pp. 61-64, 1990.

[15] M. Nemani, and F. Najm, "Towards a High-Level Power Estimation Capability “, IEEE Trans. on CAD, Vol. 15, No. 6, pp. 588-598, 1996.

[16] K. Keutzer, and P. Vanbekbergen, "The Impact of CAD on the Design of Low Power Digital Circuits", IEEE Symposium on Low Power Electronics, San Diego CA, Oct. 1994.

[17] L. Benini, R. Hodgson, and P. Siegel, "System-Level Power Estimation and Optimization", International Symposium on Low Power Electronics and Design, pp. 173-178, Aug. 1998.

[18] K. Roy and S. Prasad, "Low Power CMOS VLSI: Circuit Design", John Wiley & Sons, 1999.

[19] G. K. Yeap, and F. N. Najm , "Low Power VLSI Design and Technology".

[20] R. Jacob Baker, Harry W. Li, David E. Boyce, "CMOS Circuit Design, Layout, and Simulation".

[21] *ModelSim* SE User's Manual, Version 5.71, June 2003.

[22] VCS/VCSi User Guide, Version 7.0.2, September 2003.

[23] Verilog-XL User Guide, Product Version 3.4.

[24] Cadence's *Silicon Ensemble*, http://www.cadence.com/products/digital_ic/sepks/index.aspx.

[25] Cadence's *Virtuoso* Layout Editor, http://www.cadence.com/datasheets/4888_*Virtuoso*LE_DSfnl.pdf.

[26] Profilic tools, http://www.prolificinc.com/progenesis.html.

[27] Synopsis's *Design Compiler*, http://www.synopsys.com/products/logic/design_compiler.html.

[28] Synopsys's *Power Compiler*, http://www.synopsys.com/products/power/power_ds.pdf.

[29] *Power Compiler* User Guide, Version U-2003.03, March 2003.

[30] *PrimePower* Manual, Release U-2003.06-QA, June 2003.

[31] Synopsys's *PrimePower*,
http://www.synopsys.com/products/power/*PrimePower*_ds.pdf

[32] Synopsis's *NanoSim*,
http://www.synopsys.com/products/mixedsignal/*NanoSim*/*NanoSim*.html

[33] *NanoSim* User Guide, U-2003.03, March 2003 Version

[34] *NanoSim* Quick Reference Guide, Version V-2003-12, December 2003.

[35] *NanoSim* Integration with VCS Manual, Version U-2003.03, March 2003.

[36] Circuit Simulation and Analysis Tools Reference Guide, Version U-2003.03, March 2003.

[37] Using the X-*Calibre* PX-C/PX-RC Tool , Mentor Graphic's *Calibre* 2002.1

[38] CosmoScope Reference Manual, Version V-2003.12. December 2003.

[39] SimWave User Guide, Version 3.18.

VITA

Ashwin Balakrishnan was born in Neyveli, India. He did his schooling in Jawahar Higher Secondary School, Neyveli. He then went to the Crescent Engineering College affiliated to University of Madras, Chennai and obtained his Bachelor of Engineering degree in Electronics and Communication Engineering in 2001. Since August 2001, he has attended graduate school at the University of Tennessee, Knoxville and plans to graduate with a Master’s degree in Electrical Engineering in August 2004. He has held Graduate Research Assistant positions at the Department of Electrical and Computer Engineering, Innovative Computing Laboratory at the Department of Computer Science and UT Telehealth Network.