12-10-1996

An ASIC Power Analysis System for Digital CMOS Design

Du Van Nguyen
Portland State University

THESIS APPROVAL

The abstract and thesis of Du Van Nguyen for the Master of Science degree in Electrical and Computer Engineering were presented December 10, 1996, and accepted by the thesis committee and the department.

COMMITTEE APPROVALS: [redacted]
W. Robert Daasch, Chair

[redacted]
Y.C. Jenq

[redacted]
Gerald Recktenwald
Representative of the Office of Graduate Studies

DEPARTMENT APPROVAL: [redacted]
Rolf Schaumann, Chair
Department of Electrical Engineering

****

ACCEPTED FOR PORTLAND STATE UNIVERSITY BY THE LIBRARY

by [redacted] on 19 March 1997

ABSTRACT

An abstract of the thesis of Du Van Nguyen for the Master of Science in Electrical and Computer Engineering presented December 10, 1996.

Title: An ASIC Power Analysis System for Digital CMOS Design

Consumers demand for products with ever-increasing functionality, performance, and longer battery life is making low power a critical factor in most system designs. Lower-power designs are less expensive to produce, more reliable, and cost less to operate - factors which create a real competitive advantage. As a result, ASIC designers are demanding better ways to analyze the power of their designs.

In this thesis, an ASIC Power Analysis System (APAS) is developed. APAS is an interactive simulation-based power analysis tool. Using a non-intrusive design technique , APAS can dynamically "snap on" to an existing simulation environment. APAS currently supports three ASIC simulators: QuickSim II, QuickHDL, and Verilog-XL, all with a common library.

Diagnostics is one of the key features in designing APAS. APAS's hierarchical capabilities can be used to find out which blocks in the design are consuming the most power. The designer can then focus on reducing power in these blocks. Analyzing the dynamic behavior of the circuit is also key to power reduction. APAS can dynamically trace the power consumption of any block in the design, giving the user a graphical representation

of the periods of high and low activity.

The key to accuracy in APAS is the power models. APAS's power models represent dynamic switching, dynamic short-circuit and static currents. To Achieve optimum accuracy, APAS's models account for input slew, load and internal state. APAS's power models are written in the Power Modeling Format (PMF). PMF's combination of equations and interpolated tables provide the means to capture the most detailed cell characterization data under varying slew, load and state conditions. PMF is designed to support not only gates but also I/O cells, RAMs, and higher-level macros.

AN ASIC POWER ANALYSIS SYSTEM

FOR DIGITAL CMOS DESIGN

by

DU VAN NGUYEN

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
in
ELECTRICAL AND COMPUTER ENGINEERING

Portland State University
1997

ACKNOWLEDGEMENTS

I would like to thank Dr Robert Daasch, my advisor and friend. His brilliant guidance, invaluable suggestions, and continued encouragement has made this thesis possible.

I especially wish to thank my wife, HoangYen and my children, HoangAnh, Hoang-Lan and HoangPhuong who continue to support me with their love, belief and emotional encouragement. Your belief and encouragement is a constant source of inspiration to me.

I also want to acknowledge my colleagues and friends at Mentor Graphics Corporation for their endless contribution and enthusiastic support. I appreciate your help in sharing the workload especially during the critical time of the product release so that I could complete the last part of this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I
INTRODUCTION

Consumers continue to demand products of smaller size yet larger feature sets while market competition requires lower prices [1]. For example, today's hand-held cellular phones are less than one quarter the price, one quarter the size, and one quarter the weight, while batteries last four times longer as compared to early models. Systems that used to requires multiple chips on a circuit board are now designed on single integrated circuits. With product life cycle shrinking and product prices falling, the reality of time-to-profit requires development of more sophisticated design methodology and with clock rates and chip component densities on the rise, power analysis is quickly emerging as one of the "hottest" issues for today's designers [2].

THE NEED FOR POWER ANALYSIS

The demand for smaller size coupled with increased functionality has resulted in densely packed electronics products, increased thermal concerns, and headaches over how to sustain and improve battery life. Overheating and battery life are major concerns for portable consumer products [3]. A design that generates too much heat or shortens battery life can delay or sometimes obstruct the successful introduction of new product [4]. The best and most cost effective solution is for designs to consume less power which requires a methodology for practical power analysis. Accurately determining power consumption prior to production is a critical component for this methodology that is key to the success of today's as well as tomorrow's systems on silicon.

To further complicate matters, the added constraints of stricter power budgets is set against the background of increasing pressure to design more complex, yet less expensive devices in less time. As a result, designers requires that solutions fit easily into existing design flows. A power analysis solution must have the performance and capacity to address the complex designs. Finally, the solution must deliver high accuracy so that the results can be used to make critical design decisions: from architecture and circuit changes to reduce power, to package selection and cooling specifications to improve system reliability and lower cost.

THE VARIETY OF SOLUTIONS

Designers require a full spectrum of analysis choices at each level: estimation, optimization and simulation. Although research has produced some promising methods to help designers create reduced power systems and chips, a full power analysis solution presents a significant development challenge. While most agree that design changes at the earlier stages of the design process (architecture or RTL) offer the highest potential gain [1], a proven general methodology is not yet available. Available solutions are very limited in their scope. On the other end of the spectrum, there is transistor-level simulation primarily for back-end verification [2]. This requires a transistor-level description, which enables the highest level of accuracy, but is simply not an option for most designers. Gate-level power analysis is faster by orders of magnitude than transistor-level while still providing the accuracy necessary to produce lower power designs. The fact that gate-level simulators do not require transistor-level models makes them a prime candidate for an ASIC power simulation solution [1]. Gate-level power analysis tools can be applied at several stages in the design flow: after synthesis, after floor planning, and after layout parasitic have been extracted. Better estimates of the routing parasitic will lead to more accurate power analysis results.

Actually 'gate-level' is somewhat of a misnomer. This type of tool must solve the power analysis problem facing the ASIC designer. As such it must not only work at the gate level, but must also be able to handle the other kinds of blocks that occur in ASIC designs. Most notably, it is rare to find designs which do not include memories (RAMs or ROMs). Furthermore, with the growth of Intellectual Property (IP) to manage complexity, even more complex blocks such as cores (uProcessors, uControllers, DSP, etc.) will become commonplace [5].

ASIC LEVEL POWER ANALYSIS

The primary goal of any power analysis tool is to provide information about the power consumption of the design. The designers use these results to determine if the device has exceeded the power budget and, if possible, to explore ways to reduce the power. To this end, a power analysis tool must first of all provide accurate power estimates. Secondly, the tool should provide the performance, ease of use, and power diagnostics the designer needs to efficiently reduce power consumption.

Before detailing the different power analysis tools and how they compare, it is important to review what causes power consumption in a design. Power consumption can be divided into two types: dynamic and static. Dynamic power is the power consumption due to a signal change of state. This includes both the power consumed in charging the capacitance of the signal as well as the dynamic short-circuit current of the driver or receiver of the signal. Dynamic short-circuit is caused by the momentary path from supply to ground when the input of a complementary gate changes state and the pull-up and pull-down are both partially on. This current is highly dependent on the slew of the input signal and typically accounts for 10% to 30% of the total power consumption [6].

Static power is most commonly due to technology process issues. These include leakage of reversed-biased diodes and off transistors. While most designers tend to consider leakage power negligible today, it will become more significant as supply voltage and consequently threshold voltage continues to drop. There can also be state-dependent static power. Two examples are NMOS logic gates and current sources in sense amps. If a designer is not careful, or aware of how widely these kinds of circuits are used in a design, the static power can be a significant portion of the total power.

In an ASIC, the designer specifies the interconnection of a set of cells from a library. This makes it is convenient to group power in another way: nets versus cell internal. The power dissipated when a net changes state depends upon the capacitance (C) of the net and the supply voltage (V). The formula is 1/2*C*V**2 for a single transition. The power of the cell internal depends upon the internal structure of the cell, its state, and the slew of the inputs. The cells can be modeled for power analogously to the way they are modeled for functionality or timing. The model can represent both dynamic and static power consumption for the cell under a variety of input or state conditions. Power analysis tools differ fundamentally both in how they determine the activity of the circuit and in how they use power models.

Probabilistic Estimation

One approach to power analysis today is to do a static power estimation of the activity of the design. The designer specifies signal switching frequencies at the circuit inputs and the tool uses a probabilistic technique to determine the switching frequencies of all of the internal nodes as well as to the outputs. Once the frequencies have been determined, the power is calculated based on the net switching power and possibly a simple power model for the cell. Probabilistic tools [7] provide fast, pattern-independent results at the expense of accuracy. It is not possible to predict how well, or poorly, the estimated frequencies will match the real circuit behavior. Furthermore, the cell power models can not be state-dependent since there is no correlation between the states of any two signals. Finally, a probabilistic tool is not able to analyze the effect of common power-down techniques such as gated clocks or latch insertion.

Post-processed Simulation Results

To address the unpredictable inaccuracy of the probabilistic approach, power analysis tools have been developed that connect to logic simulators to access the actual switching activity of the design. These simulation-based power analysis tools differ in the kind of information they receive from a host logic simulator. Most use the simulator to create a file of node toggle activity which is post-processed by the power analysis tool [8]. For example, a static estimation tool, instead of probabilistically computing signal switching frequencies, can use the actual frequencies of each net as determined by the simulation run. This solves some of the accuracy problems of the static estimation tool, but still does not provide the correlation of signal states needed to accurately determine cell internal power.

Instead of getting just the node frequencies from the simulator, it is possible to obtain a complete transaction log of all of the signal state changes and when they occur. This technique is used in some tools because it provide enough information to correctly determine the states of the cells. It provides the additional advantage of giving the designer an indication of when the design is consuming power. There is also a significant disadvantage to this approach. The generated log files can easily exceed a Gigabytes in size, creating problems with storage and management, not to mention the time required by the tools writing and reading such enormous files.

Dynamic Power Simulation

A better approach is to tightly connect the power analysis tool to the host simulators so that as the simulation progresses, the power is computed dynamically. This technique is even more accurate than the post-process tools, since effects such as multiple bus drivers, which may not appear as signal changes can be detected and analyzed. Furthermore, it does not require writing and manipulating large log files. Where the dynamic power simulation approach has tremendous advantages over the other methods is in the area of diagnostics. A power analysis tool that works dynamically with the simulator can be used interactively. This means that the designer can see how the power consumption of the design blocks varies with time as the simulation progresses. Even more significantly, the designer can stop the simulator at any time, and determine which blocks are consuming power at that moment. This is extremely valuable in exploring power-down techniques for reducing power.

Ease of use is another advantage of the dynamic power simulation. Since the power analysis tool "snaps-on" to a host simulator, the designer is able to perform power analysis at the same time as functional or timing simulation. Thus power analysis becomes part of the existing design flow - not an afterthought. No additional steps or separate tools are required. The designer is able to leverage existing simulator experience to quickly learn the power analysis tool.

THESIS OVERVIEW

In this thesis, an ASIC Power Analysis System (APAS) is created to provide ASIC designers a power analysis solution with accurate results, high performance, and diagnostics capability.

APAS is an interactive simulation-based power analysis tool. With a non-intrusive design technique, APAS is designed to work seamlessly with the existing ASIC simulators: Quicksim II, QuickHDL and IEEE-1364 compliant Verilog simulators.

Consistent results are critical for accurate power analysis. APAS guarantees the same behavior by relying on the ASIC simulator for functionality and timing. Accurate results required not only the proper simulation context but also a versatile modeling language.

APAS's power modelling language, the Power Model Format (PMF), is designed to handle gates, I/O cells and macros. PMF models are simulator-independent. PMF can be used to model the main contributors of power: dynamic (switching) current, dynamic (internal) short-circuit current, and static current. To gain the optimum accuracy, PMF models support input slew, output load and state-dependent power. State dependencies are key contributors at the gate level but their importance only increases with the development of IP like RAMs (including multi-ports), multipliers, other macros, and cores.

Performance is another important criteria in designing APAS. APAS's performance is compatible to the performance of the gate-level host simulator. The hierarchical nature of APAS allows the user to select specific blocks to work on, or the whole design. In addition the user has the ability to turn power analysis on and off interactively. The minor overhead for power analysis is controlled by the user, and allows the user to do power analysis at the same time as functional and timing analysis. This greatly helps the design throughput by making the most of the simulation time.

APAS leverages the power of consistency by using the designer's flow and simulation environment for power analysis. Technology and design challenge are changing rapidly so it is important to use consistency to improve productivity. APAS allows the user to leverage the simulation environment to gain consistent behavior and performance.

Diagnostics is key to reducing power. Designers of low power devices not only need to know how much power the ASIC consumes, but what can be done to reduce the power consumption. Using APAS, the designer can selectively delve deeper into problems, working with a combination of graphical and textual modes to find out where and when the design is consuming power.

THESIS ORGANIZATION

In this chapter, INTRODUCTION, we have discussed the power dissipation issue and hence the need for a power analysis solution. Diverging from some previous approaches, our approach is based on interactive simulation to provide ASIC designers a power analysis solution with high accuracy, high performance and diagnostic capability.

In chapter II, POWER MODELING, sources of power dissipation are examined and modeled from cell instance perspective. A Power Model Format (PMF) to describe power view of an ASIC cell is described. This format is capable of describing essential factors that attributes to the accurate simulation of power dissipation.

Chapter III, POWER CHARACTERIZATION, discusses power characterization technique to create accurate power model for standard cell. Accurate power model is a major factor in estimation of power dissipation. Power characterization can be done using existing cell timing characterization environment.

Chapter IV, ASIC POWER ANALYSIS SYSTEM, describes the design and features of the power analysis system, a dynamic simulation-based power analysis tool for cell-based ASIC designs.

Chapter V, EXPERIMENTS AND RESULTS, reports results of experiments to measure the power estimation accuracy, performance and capacity of the ASIC Power Analysis System.

Chapter VI, CONCLUSION, gives a summary of the ASIC Power Analysis System developed in this thesis and future work.

Appendix A, POWER MODEL FORMAT.

CHAPTER II
POWER MODELING

INTRODUCTION

ASIC designs differ from traditional integrated circuit design in that standard gate-level primitives (cells), such as nand, nor, buffer, and flip-flop are the lowest level constructs used, as opposed to transistors of varying sizes. Designing at this higher level of abstraction permits greater productivity and allows for increased design automation[5]

Power dissipation at the transistor level is relatively well understood [2][24], however despite significant advances in ASIC design tools and methodologies, effort in modeling and analyzing power dissipation for ASICs from an ASIC design methodology is still in its infancy[1].

In this chapter, we will describe a dynamic method to model and analyze power dissipation for digital CMOS ASIC design. Since this method is based on the dynamic information available at the functional instance during simulation, combined with pre-characterized data, it provides a very accurate estimation of the power behavior of the instance in the design. We will also propose a simple, flexible format to describe a power model. This simulator-independent power model format plays a crucial role in the design of our non-intrusive power analysis system that will be described in chapter IV.

Previous Work

There are reports of power estimation techniques which use the well perceived formula: P = a*f*C*Vdd**2 to perform a quick power estimation [21][22] assuming that static power and short-circuit power is very small, therefore ignored. In the above formula which represents the capacitive load switching power, a is the activity factor or number of node transitions per clock cycle, f is clock frequency, C is switching capacitance, and Vdd is the operating supply voltage. To compute capacitive load switching power, It is important to accurately determine the activity factor and the node switching capacitance [23]. Switching capacitance can vary, depending on whether pre or post-layout information is used. The activity factor is not only a function of the inputs to the circuit, but also of the delay model used [10] . Zero, unit and gate-specific delay models can each contribute varying magnitudes of switching activity due to glitches, hazards and spikes [9]. Determining the accurate activity factor is the main challenge for these power estimation techniques. In general, these power estimation techniques fall into two categories: probabilistic and simulation based [18].

Probabilistic techniques have been proposed to efficiently propagate activity factors within a circuit [11][23][28]. These approaches have been demonstrated on combinational circuits, though Devadas[11] also discusses an approach to estimate activity in sequential circuits. Probabilistic estimation work well in the power-directed logic minimization and technology mapping[29][30], since combinational logic synthesis can use these fast estimation algorithms inside the synthesis loop. Moreover, while doing minimization and technology mapping, relative accuracy is often sufficient. But using probabilistic techniques for large block level power estimation, requires the handling of different circuit styles, dynamic/precharged logic, tristate drivers, latches, flip-flops etc. Developing models for some of these scenarios is not entirely feasible [22]. In addition, the effects of known inaccuracies (in the probabilistic approaches) such as reconvergent fanout at internal nodes, temporal and spatial correlation of primary inputs, and glitching activity (which arise due to signal skew) need resolution.

Simulationed-based techniques such as Entice-Aspen[8] are based on a logic simulator to monitor power activities during simulation and combine with pre-characterized power models[19] to compute total power dissipation in a cell-based ASIC design. Entice-Aspen - an in-house power analysis system - is a post-simulation process, also suffers inaccuracy due to dynamic problems such as glitch power and bus contention. Moreover, this post-simulation technique is not able to provide diagnostic capability to determine where and when power is wasted and observe the dynamic power profile.

Our approach to ASIC power analysis

Our approach is based on dynamic simulation. Power dissipation is modeled and estimated during simulation in contrast to post simulation. In this approach, a shadow power instance is created for each instance in the design to monitor power activities based on its power model. A power model contains characterized power/energy data, and power statements.

MODELING SOURCES OF POWER DISSIPATION

When considering an ASIC's power dissipation, the total power can be viewed from several different perspectives: static vs dynamic, cell vs load, core vs I/O. For the purposes of calculating an ASIC's total power, all three perspectives will be valuable; however the first two, static vs dynamic and cell vs load, will form the backbone of this analysis methodology.

Static Power

As described above, total power dissipation may be split into static and dynamic power, otherwise known as DC and AC power. Static power represents the power dissipated when the gate is in a steady state, and dynamic power represents the power dissipated when the gate is switching.

Although static power is only a small component of the total power, it is becoming important for portable (battery-operated) designs where static power dissipation during idle phases becomes a significant part of the overall power dissipation [1].

The sources of static power consumption in a digital CMOS device are leakage currents, and static currents.

Leakage currents consist of reverse-bias diode leakage at the transistor drains, and sub-threshold leakage through the channel of an "off" device. Diode leakage occurs when a transistor is turned off and another active transistor charges up/down the drain with respect to the former's bulk potential. Sub-threshold leakage current from source to drain is caused by reduced threshold voltages that prevent the gate from completely turning off.

Leakge power is processing technology dependent and typically in the microamp range [15]. While this is typically a small fraction of the total power consumption, it could be significant for a system application which spends much of its time in standby operation, since this power is always being dissipated even when no switching is occurring. Regardless of the physical reasons, leakage power dissipated in a cell can be modeled as a cell-specific value, and computed as:

$$P_{leakage} = V_{dd} \cdot I_{leakage} \qquad (1)$$

In CMOS technology, when circuits are in steady state operation (i.e. non transient operation), static currents exist in two situations: reduced voltage levels feeding into complementary static gates and pseudo NMOS circuit styles. Usually static currents are dependent on the state of the device.

As a result of reduced input voltage level (Vdd - Vt), as shown in Figure 1. one device (NMOS) of the complementary static gate is ON while the other device (PMOS) is also weakly ON; there exists a static current from the supply to ground. This might consume significant power if the circuit is idle most of the time.

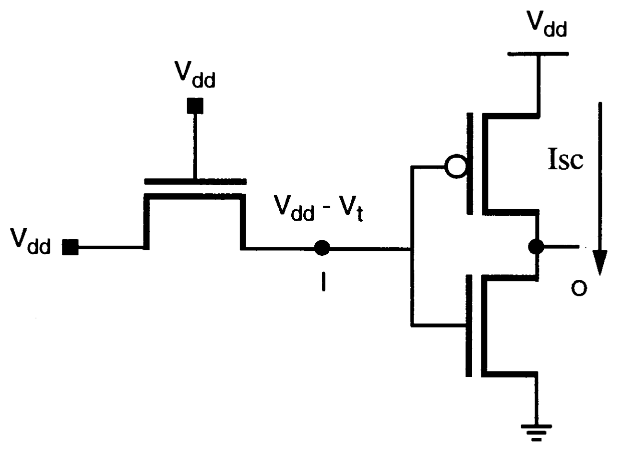

Figure 1. Reduced Voltage Level Feeding a CMOS Gate

In the pseudo-NMOS logic implementation style, as an example shown in Figure 2, the PMOS device is always ON. When the output is driven low by a resistive path from output node to ground through the NMOS network, there is a conducting path from supply to ground.

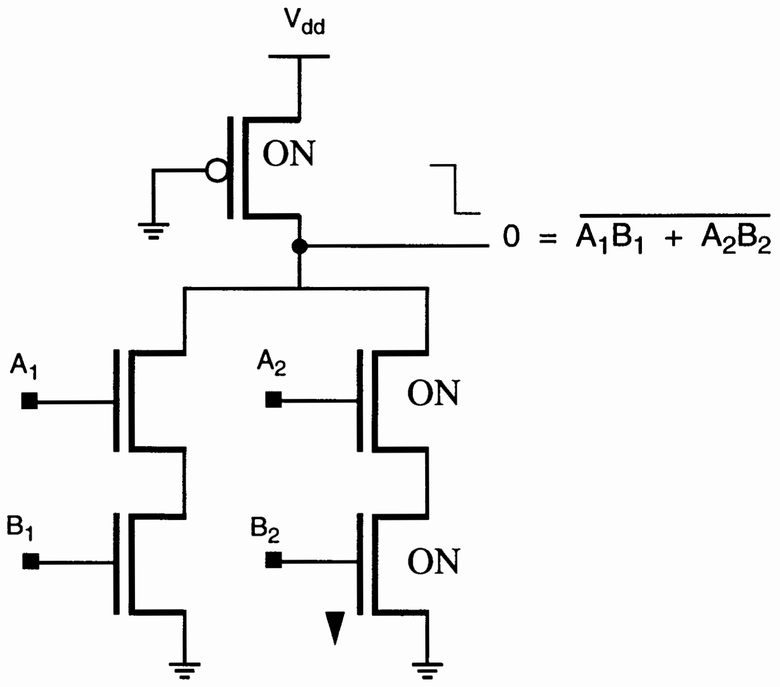

Figure 2. Pseudo-NMOS AND_OR_INVERT Gate

For an ASIC cell, static power resulted from static current can be modeled as equation (2), where c is the percentage of time the cell instance stays in a particular steady state.

$$P_{static} = c \cdot V_{dd} \cdot I_{static} \qquad (2)$$

Dynamic Power

The components of dynamic power consumption in CMOS ASIC are cell power and capacitive load switching power. The (internal) cell power is the power dissipated within the boundary of the cell. Cell power consists of short-circuit (or crossbar) power and internal capacitive switching power.

When a CMOS gate is switched by an input signal switching with a nonzero rise or fall time, both the n- and p-transistors will conduct simultaneously for a short time. During this time when input signal voltage is between Vtn and Vdd-Vtp, there will be a "short-circuit current", flowing directly between Vdd and ground and not participating in charging or discharging load capacitance.

Consider the CMOS inverter in Figure 3. With the input low, the n-channel is OFF and the p-channel is ON. No current flows as there is no path from Vdd to ground. As the inverter is switching due to the input switching, the n-channel is ON as the input voltage crosses Vtn while the p-channel remains ON until the input voltage rises to Vdd-Vtp. Thus there is a significant amount of time during which both devices are ON resulting in a resistive path from Vdd to ground.

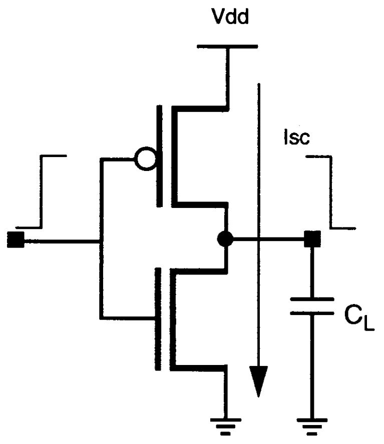

Figure 3. CMOS Inverter

Short-circuit power is dependent on input rise/fall times and output load capacitance[6]. Short-circuit power are significant when rise/fall time at the input of a gate is much longer than the output rise/fall time (which is dependent on output load capacitance). This is because the short-circuit path will be active for a longer period of time.

Internal cell capacitive power results from the charging and discharging of internal capacitances of the cell. Consider the CMOS OR gate in Figure 4. This two-level logic cell has an internal node between the NOR structure and the output inverter structure. As an input is switching, the internal capacitance will be charged or discharged depending on the state of the other input pin.

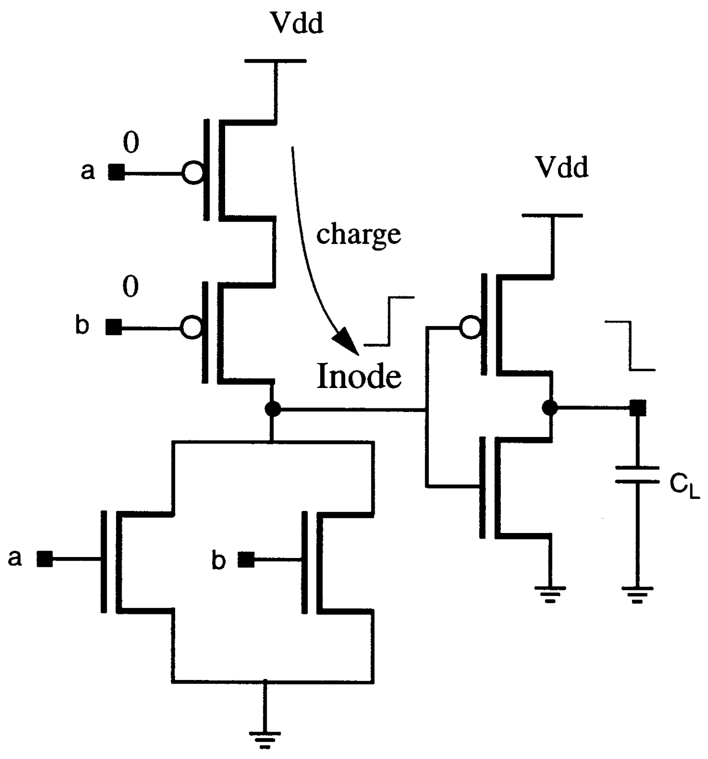

Figure 4. CMOS OR

In more complex cells, when an input is switching, the cell output may not change state but there is still a “hidden” power dissipated by those portions internal of the cell which change state. The predominant example of this hidden power is the buffered flip-flop. This hidden power is due to the switching of the clock buffering in each individual flip-flop, and has been found to account up to 50% of an ASIC's total dynamic power[19].

In this thesis, internal cell power can be conveniently modeled as a function of input slope, output load and cell internal state. This function can be represented as a set of piecewise linear equations or multi-dimensional tables. The piecewise linear equations and multi-dimensional tables capture data created by power characterization of the cell over a typical range of operating conditions.

The capacitive load switching power which is the dominant component (more than 90% for "properly" designed circuits[15]) of dynamic power arises when energy is drawn from the power supply to charge load capacitance or when the load capacitance is discharged to the ground. This component is the primary target of any power estimation technique.

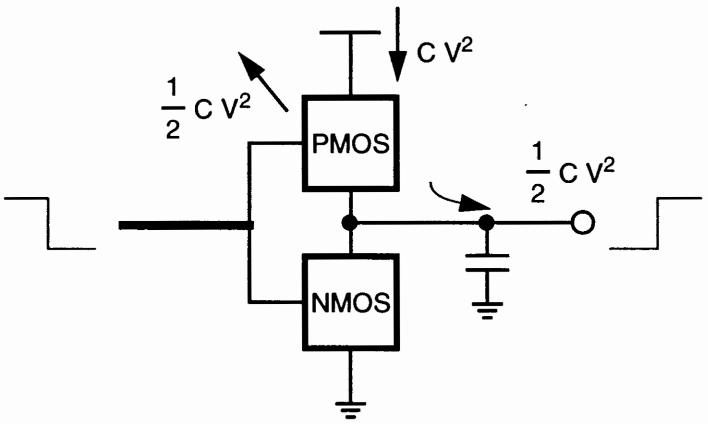

Figure 5. Capacitive load switching power of a CMOS gate

Consider any input transition which causes a 0 -> Vdd transition at the output of the generic CMOS gate. The instantaneous power and energy can be computed as follows:

$$p = \frac{dE}{dt} = i_{supply} \cdot V_{dd} \qquad (3)$$

Where isupply is the instantaneous current being drawn from the supply voltage Vdd and is equal to,

$$i_{supply} = C \cdot \frac{dV_{out}}{dt} \qquad (4)$$

Therefore, the energy drawn from the power supply for a 0->Vdd transition at the output node is given by:

$$E_{supply} = \int_0^T p dt = V_{dd} \int_0^T i_{supply} dt = V_{dd} \int_0^V C dV_{out} = C V_{dd}^2 \qquad (5)$$

The energy stored in the output load capacitor is given by:

$$E_{cap} = \int_0^T p_{cap} dt = \int_0^T V_{out} i_{cap} dt = \int_0^V C V_{out} dV_{out} = \frac{1}{2} C V_{dd}^2 \qquad (6)$$

Therefore, half of the energy drawn from the power supply is stored in the output capacitor and half of the energy is dissipated in the PMOS network. For the Vdd->0 transition at the output, no charge is drawn from the power supply and the energy stored in the capacitor is dissipated in the pull-down NMOS network.

In addition to all the power components described so far, from a ASIC gate point of view there are two more power concerns that must be addressed in order to improve accuracy: glitch power and bus-contention problem.

Glitch power

Glitches are identified as unwanted appearances of signal transitions. These logical hazards usually occur in connection with reconverting paths in logic, that is spurious signals are caused by different arrival times of input signals to a gate.

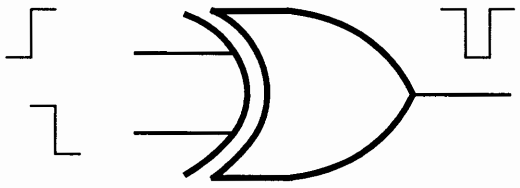

Figure 6. Glitch in XOR gate

If several parallel paths with different delays are merged into one gate, glitches may occur, e.g. as shown in Figure 6, two opposite-state input signals have different arrival times to an XOR gate. Their "briefly change to same state then back to different state" makes the output change twice, in spite of the fact that the final result do not change. This glitch give rise to unnecessary power consumption.

Some investigations of typical effect of glitches have been performed. Empirical results showed that glitch power is from 8% to 25% of dynamic power [9]. For more complex cell, glitch power is observed to reach 67% of dynamic power (for a 16X16 bit multiplier, logical depth about 30) [15].

During glitching, output transition is usually an incomplete transition, that is the output node voltage change (dV) is less than Vdd. Hence, glitch power can be expressed as:

$$Pglitch = \frac{1}{2} \cdot C \cdot V_{dd} \cdot dV \qquad (7)$$

Where dV is the Reduced Voltage Swing

Voltage swing is dependent on the ratio of glitch width (tg) and gate delay (td), hence can be approximated as:

$$dV = \frac{tg}{td} \cdot V_{dd} \qquad (8)$$

In theory, dV is dependent on input slope, output load (figured in td) and input pattern (figured in tg), however in engineering tradeoff between efficiency and accuracy of glitch power computation at gate level, the ratio (tg/td) can be further approximated to a reasonable constant value (e.g 50% as an average). This value can be an empirical result and may be specified as a constant value in the gate power model.

Bus contention

When multiple tristate output drivers driving the same external load (bus), error in power simulation may occur due to two reasons: bus contention short circuit and duplicated account of capacitive load switching power. Consider the following scenarios with two tristate drivers:

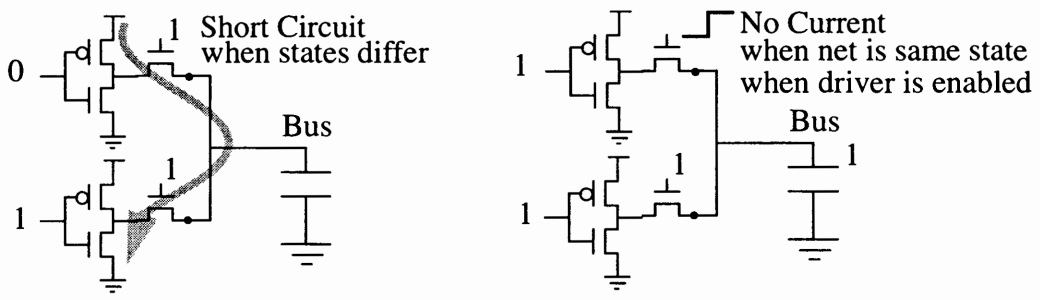

Figure 7. Bus Contention

One output transitions from Z to H, and the other output changes from L to Z. The signal delays may be slightly different so that the first output transition may complete some short amount of time before the second output changes to Z. During this brief period of time, a short circuit path exists from Vdd through PMOS plane of the first gate, through external bus, and through NMOS plane of the second gate to Gnd. In a typical design, this delay variation is small and the power dissipation is negligible. In fact, for some logic simulator, this grace period (e.g 1ns) is reasonably ignored. The first output drives the bus logic, and charges the load capacitance to Vdd.

If the bus logic is already H, that is the second output changes from H to Z, there is neither short circuit nor charging power.

If initially the bus is in logic state Z, that is both outputs are in Z state, then one of the output changes from Z to a valid state (H/or L). If the bus capacitance is free of charge/ or full of charge then there may be a charging/ or discharging power. In some logic simulator, the bus state can be represented as HZ or LZ, and based on this information we can determine whether there is a charging/discharging power. However in other logic simulator such as VHDL we must explicitly keep track of the previous bus state before its floating.

Now assume that one output is at H state, and the other output changes from Z to L. Bus contention occurs, and there is a short circuit path from Vdd through PMOS plane of the first gate, through external bus, and through NMOS plane of the second gate to Gnd. If the first driver is stronger than the second, the bus does not change state, there is no discharging power. However if the first driver is weaker than the second, the bus will changes state to L and there is discharging power. The short circuit is dependent on particular drivers and can be significant as long as the drivers stay unchanged. This case is a bad design for power and requires a special mechanism to recognize and calculate power effect.

In general, the bus contention occurs during a time period (tb) less than the gate delay (td) of the drivers. One approach to approximately model this bus short-circuit behavior is to assume that both drivers are the same strength, and one is charging the bus with a power effect of 1/2*C*Vdd**2, and the other is simultaneously discharging the bus with the same power effect 1/2*C*Vdd**2. The bus short-circuit power effect can be modeled as:

$$Pbus = \frac{tb}{td} \cdot 2\left(\frac{1}{2} \cdot C \cdot V_{dd}^{2}\right) \qquad (9)$$

Above we have analyzed power effects in details for different types of circuits and suggested how to appropriately model these sources of power dissipation. In the next section, we will present a format that can be used to describes the power effects of ASIC cells.

POWER MODEL FORMAT

The fundamentals of our simulation based approach to ASIC cell-level power analysis are to accumulate energy usage during a simulation by tracking the power activity of the cells in the circuit. The challenge is to approach the accuracy levels of transistor level power estimation while maintaining cell-level simulation performance and capacity. Success is dependent almost entirely on the quality of "power models" to accurately access power/energy dissipation given a state change in the circuit.

In this section, we present an overview of the ASCII model format that has the constructs necessary to describe all power effects in a ASIC cell. The complete description of the power model format will be described in appendix A. It is expected that model generation will be automated, using data generated by characterization tools operating on models at a lower level of abstraction, such as SPICE.

The power model format can be considered to be the model interface to power analysis tools. It is independent of any specific database or hardware modeling language.

Overview of the Power Model Format

Model Data And Expressions

- Signal Declarations
- Default Data
- Defined Constants
- Table Data
- Equations
- Internal Signal Definitions

Model Evaluation

Transition (dynamic) Energy Statements

Static Power Statements

In general, a power model consists of two sections: Model Data and Expressions section, and Model Evaluation section. The Model Data and Expressions section contains signal declarations, default data, defined constants, table data, defined equations, and internal signal definitions. The Model Evaluation section contains two types of power/energy statements: dynamic and static. Dynamic transition energy statements describe energy dissipated due to a power transition (signal change that results in power dissipation). Static power statements describe the static power component which includes leakage and static power. Both statement types can have additional condition to specify their state-dependency.

SUMMARY

We have analyzed in details all sources of power dissipation in an ASIC design which can be grouped into two main categories: static power and dynamic power. From there, we described methods to model the power behaviors from a gate-level standpoint. At gate-level, there exists two additional cases that must be considered in order to improve power estimation accuracy: glitch power and bus contention. Finally, we presented a format used to describe a power model using data pre-characterized with circuit simulator (SPICE) using transistor-level description of the ASIC cell. In the next chapter we will discuss technique to characterize power for an ASIC cell.

CHAPTER III

POWER CHARACTERIZATION

INTRODUCTION

The accuracy of cell-based power simulator depends upon the accuracy of cell power models. The accuracy of cell power model is further dependent upon the accuracy of the power characterization process that generates them.

Power characterization process is one that captures power behavior of a cell by simulating its lower-level description over a typical range of operating conditions. That lower-level simulation is usually circuit simulation using SPICE. During the circuit simulation, selected stimuli are applied to the circuit and measurement points are specified so that the resultant data will represent all or most of actual behaviors of the cell in real designs.

In order to obtain the requisite accuracy, the power characterization for cell must account for all power components (dynamic cell power, static power) within the cell boundary as well as their dependency on cell state and operating conditions (input slew and output load).

In this chapter, we first develop a principle to separate cell power from total power by examining power dissipation under individual cases. From this developed principle, a power characterization technique is presented to measure cell power using a pair of Amp meters. Then applying the described characterization technique, we characterize a sample

ASIC library based on 0.8 micron technology using an existing standard cell characterization environment. The characterized data obtained from the power characterization are processed to automatically generate accurate cell power models. In order to assure the quality of the generated cell power models, a verification method is presented. Using this presented verification method we have verified that the generated power models simulated with our cell-based power simulator produce consistent result within 5 percent of SPICE.

Figure 8 shows an overview of the power characterization process

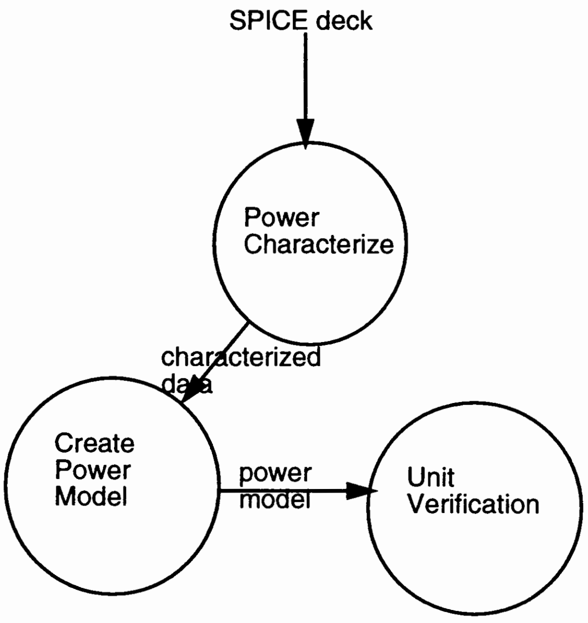

Figure 8. Power characterization process

PRINCIPLE

As discussed in previous chapter, the total power dissipated in a CMOS circuit includes dynamic and static powers. Dynamic power includes cell power and load switching power. As load switching power can be computed separately with the well-known formula (1/2*CL*Vdd**2), we need only to isolate and characterize cell power and static power.

In order to isolate dynamic cell power and static cell power we consider the case when the circuit is active and the case when the circuit is in idle state. Then for the case when the circuit is active, we further consider subclasses when output is switching and not switching.

Input Switching

When an input is switching, the circuit is active and the output may or may not be switching accordingly. However, there are energies dissipated in the circuit in both cases. The energy dissipated when output is switching is significantly different from the energy dissipated when output is not switching.

Output switching

Let's consider energy components during charging and discharging the load capacitance of a representative CMOS circuit.

Charging load capacitance (L->H transition)

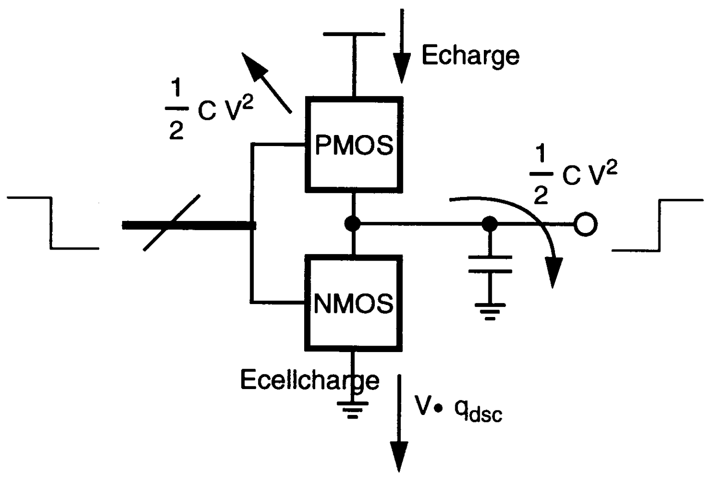

Figure 9. Energy components in charging load capacitance

When the output is switching from L to H, output capacitance is charged to Vdd as shown in Figure 9. The total energy consumption (Echarge) drawn from the power supply Vdd is divided into load charging energy (1/2*CL*Vdd**2), plus the same amount of energy dissipated in the PMOS plane, and internal cell energy (Ecell_charge). The internal cell energy includes short circuit energy and internal capacitive switching energy:

$$E_{charge} = \frac{1}{2}CV_{dd}^2 + \frac{1}{2}CV_{dd}^2 + E_{cellcharge} \qquad (10)$$

Discharging load capacitance (H->L transition)

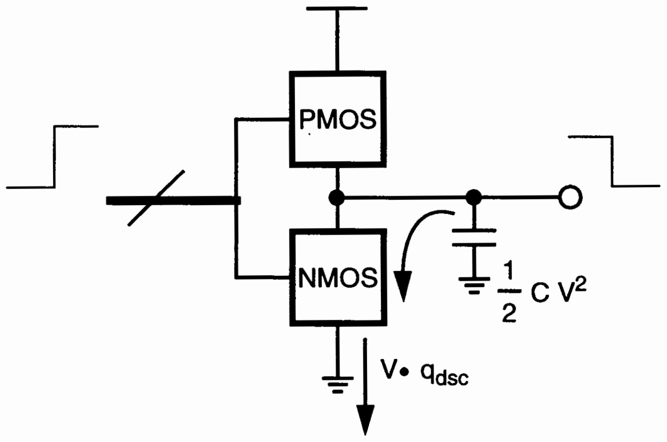

Figure 10. Energy components in discharging load capacitance

When output is switching from H to L, output capacitance is discharged as shown in Figure 10. The total discharging energy (Edischarge) includes load discharging energy and internal cell energy (Ecell_discharge). The load discharging energy which is the same as the energy previously stored in the load capacitance (1/2*CL*Vdd**2) is dissipated in the NMOS plane. The internal cell energy includes short circuit energy and internal capacitive switching energy:

$$E_{discharge} = \frac{1}{2}CV_{dd}^2 + E_{celldischarge} \qquad (11)$$

Output not switching

Although output is not switching (as in a DFF when clock is switching but output is stable), there is still internal cell energy due to dynamic short-circuit and internal cell capacitances switching:

$$E_{cell} = E_{shortcircuit} + E_{switching} \qquad (12)$$

Input not switching

When input is not switching, the circuit is in idle state; static power dissipated in the circuit is due to leakage current and state dependent static current:

$$Pcellstatic = Pleakage + Pstatic \qquad (13)$$

TECHNIQUE

To measure cell energy in a CMOS cell we use a pair of Amp meters as shown in figure 11. The first Amp meter Mdd is inserted between Vdd and PMOS plane. The second Amp meter Mss is inserted between NMOS plane and Vss. Mdd meter is used to measure total energy consumption drawn from Vdd when PMOS plane is open. Mss meter is used to measure total energy dissipated when NMOS plane is open.

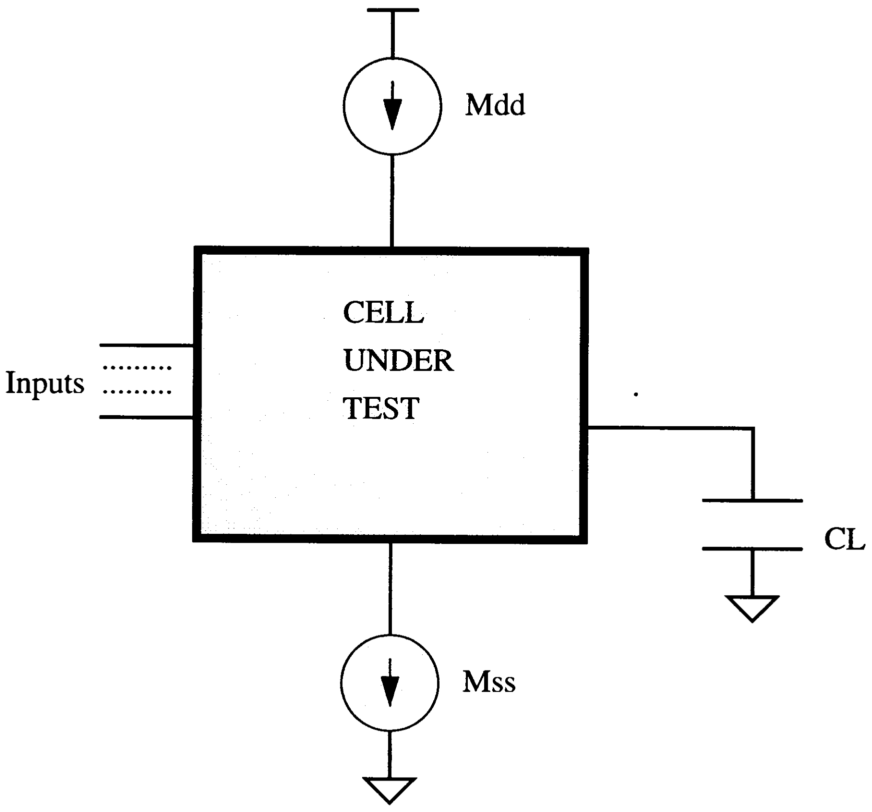

Figure 11. Two Amp meters used to measure cell energies

When input is switching and the PMOS plane is open to charge the load CL, total energy consumption (Echarge) is measured by Mdd. Therefore, cell energy can be derived from equation (10) as:

$$E_{cellcharge} = E_{charge} - CV_{dd}^2 \qquad (14)$$

Similarly, when the load is discharged, total energy dissipated (Edischarge) is measured by Mss, and the cell energy is simply computed from equation (11) as:

$$E_{celldischarge} = E_{discharge} - \frac{1}{2}CV_{dd}^2 \qquad (15)$$

For both cases cell energy is dependent on both input slew and output load. Hence measurements should be done with varying input slew and output load.

When output is not switching, cell energy is measured by Mdd or Mss when PMOS or NMOS is conducting. In this case, cell energy is not dependent on output load but is dependent on input slew. Therefore, measurements should be done with varying input slew. Note that the cell energy can also be dependent on the internal state of the cell.

When the cell is in idle state, Mdd is used to measure leakage power. For circuit that can have static power, the circuit should first be initialized to the state when static power occurs, and then Mdd will be used to measure the average static power.

Range of Typical Operating Conditions

When cell energy is dependent on load capacitance and/or input slew, we should measure it over a range of typical operating conditions. The range of typical operating conditions is technology dependent, and should be selected carefully so that it sufficiently covers the real operating conditions of the cell in design, but is not so wide that it requires long characterization time.

As an example, typical operating range can be selected as:

Load capacitance CL: [0, 4inv]

where inv is an inverter input capacitance used as a standard unit fanout load

Input slope: [0.5tp, 4tp]

where tp is typical delay of an standard inverter.

SPICE Time Step

In SPICE simulation, cell energy is measured by integrating average power over time. The accuracy is critically dependent on the precision, especially time step set during simulation. Smaller time step produces better result, however the simulation will also take longer to complete. Therefore, the time step should be set to a reasonable value. To obtain this time step value we continuously simulate average power of a chain of 5 inverters with the same periodic input pulse over many runs. In each simulation run, time step is gradually reduced until measured average power is within 1% from the previous run. This calibration process is done only once per ASIC library.

APPLICATION

Applying the above described power characterization technique, we characterize an ASIC cell library based on a 0.8 micron technology with voltage fixed at 5 Volts, and using the nominal values for the process parameters.

As a starting point, we use an existing standard cell characterization tool. This tool is primarily used for timing characterization. As data dependencies, specifically input slews and output load capacitances, for existing timing characterization processes are quite similar to those for power, little to moderate incremental effort is required to generate SPICE data to compute energy measurements. The resultant data for each cell is put into Power

Model Format (described in previous chapter), and compiled into a power library so that unit verification tests using the Power Analysis Tool (described in the next chapter) can be performed.

```
******** process corner: slow
.lib hspice_models.lib SLOW
.option brief nomod autostop
.option absv=50e-6

.SUBCKT dff
+ CKN
+ D
+ QN
+ VCC
+ VSS
...
<cell description removed>
...
.ENDS

X0 p0_0 p1_0 p2_0 P4 P5 dff

Cout_0 p2_0 0 LC

Vp0_0 p0_0 0 PULSE(5.0,0,40NS,TRF,TRF,190NS,460NS)

.TEMP 25

VP4 P4 0 5.0
VP5 P5 0 0

Vp1_0 p1_0 0 PWL 0N 0 460N 0  460.5N 5.0 920N 5.0 920.5N 0 1380N 0 1380.5N 5.0
   1840N 5.0

.MEAS p0_p2_TPHL[2] TRIG V(p0_0) VAL=2.50 TD=500N FALL=1
+ TARG V(p2_0) VAL=2.50 FALL=1

.MEAS p0_p2_TPLH[3] TRIG V(p0_0) VAL=2.50 TD=960N FALL=1
+  TARG V(p2_0) VAL=2.50 RISE=1

.TRAN 1N 1840N  SWEEP DATA=DF

.DATA DF TRF LC
...
<data values removed>
.ENDDATA
.END
```

Figure 12. Partial SPICE deck for DFF timing characterization

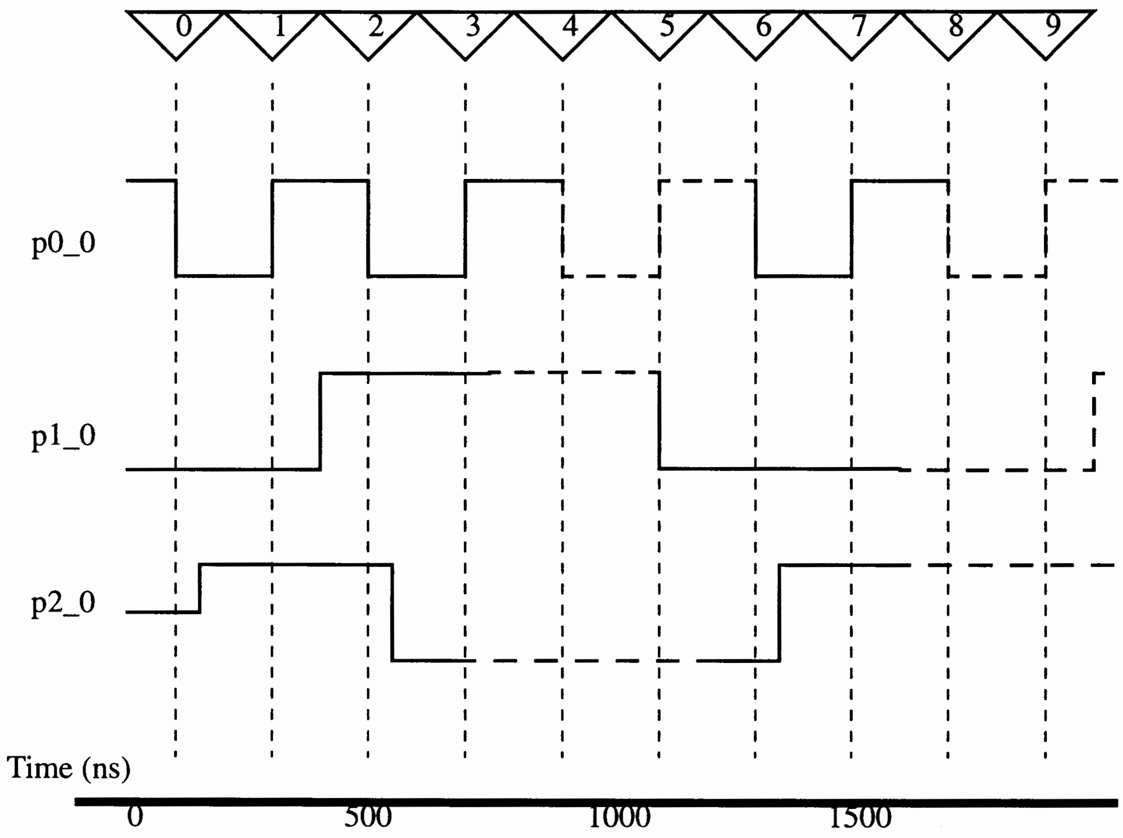

Figure 13. Timing relationship as specified from the DFF SPICE deck

Let's look in details at the power characterization process for the DFF. The DFF SPICE deck that is used for timing characterization as shown in Figure 12 is modified for power characterization. Stimulus for timing delay measurements were already generated for paths between CKN (p0_0) to QN (p2_0) when D was HIGH and LOW at Time Marker (2) and (6) respectively. Therefore, a measurement like the following would be an example to measure the energy due to those transitions:

```
.MEAS p2 AVG POWER from=<Time2> to=<Time2+50NS>
.MEAS e2 PARAM='p2*50e-9'
.MEAS p6 AVG POWER from=<Time6> to=<Time6+50NS>
.MEAS e6 PARAM='p6*50e-9'
```

But if we want to capture a more accurate power model, there are several other cases to consider too. It had been observed that some significant amount of energy was dissipated when clock is active but with no output change, for example at Time Marker (4) and (8). We must do some modifications to the stimulus set for this D Flip-Flop in order to make these 2 measurements:

```
.MEAS p4 AVG POWER from=<Time4> to=<Time4+50NS>
.MEAS e4 PARAM='p4*50e-9'
.MEAS p8 AVG POWER from=<Time8> to=<Time8+50NS>
.MEAS e8 PARAM='p8*50e-9'
```

And for completeness to this D Flip-Flop illustration, we also measured the energy values at Time Markers (3) and (7) when ckn was inactive and D was HIGH and LOW.

Therefore, unlike timing characterization where one measured the delays between an input pin change to an output pin change, power characterization for a cell should also consider those cases where the output did not change, if accuracy was of utmost importance.

For simple combinatorial cells like 2 or 3 inputs NANDs, NORs, ANDs, and ORs, it was still relatively easy to capture energy values for input vectors that had no output change. In this experiment, these input conditions were already in the original timing stimulus, and it was just a matter of putting a measurement statement at the correct time period. Considering that these energy values were observed to be 1 to 2 orders of magnitude less than those whose output changes state, we, however, might be willing to lose some accuracy if the effort in adding additional power test vectors was not justifiable when characterizing for a full and complete set of standard cell library.

Continuing to use the DFF as an example, Figure 13 illustrates a fragment of the Power Model that we wanted to achieved.

```
...
EQN node_pwr : 0.5 * GETCAP() * 1.0e-12 * 5 * 5;
BOOL cmp(sig1, sig2) : diffBITS(sig1, sig2) == 0;
// signal declarations
SIGNAL INPUT : ckn, d;
SIGNAL OUTPUT : qn;
SIGNAL GROUP : cur_st : d, qn ;
... <omitted>
dynamic {
  qn(*) :  node_pwr() ;
  ckn(\) & cmp(cur_st,'11') : tab_E1[GETFALL()][GETCAP(QN)] ;
  ckn(/) & d : tab_E3[GETRISE()] ;
  ckn(\) & cmp(cur_st, '10') : tab_E2[GETFALL()] ;
  ckn(\) & cmp(cur_st,'00') : tab_E4[GETFALL()][GETCAP(QN)] ;
  ckn(/) & ~d : tab_E6[GETRISE()] ;
  ckn(\) & cmp(cur_st, '01') : tab_E5[GETFALL()] ;
}
```

Figure 14. Fragment of DFF Power Model

To characterize the internal cell energy value for the second statement (tab_E1), this input vector corresponded to Time Marker 2.:

.MEAS *p2* AVG POWER from=*<Time2>* to=*<Time2+50NS>*
.MEAS *e2* PARAM=*'p2*50e-9'*

Actually, measurement variable e2 as illustrated should be corrected. Since there was already an energy equation to model output node switching, node_pwr(), e2 should not account for the "half CV squared" component. And finally, e2 should be factored with 1.0e9 since the units for energy expressions for dynamic power were in nanoJoules.

.MEAS *p2* AVG POWER from=*<Time2>* to=*<Time2+50NS>*
.MEAS *cknFA_11* PARAM=*'((p2*50e-9)-(0.5*LC*25))*1.0e9'*

Note that the measurement variable e2 had also been carefully changed to cknFA_11 so that such a label gave meaning with respect to the input vector transition and state. Thus, a filter program could be written to easily reconstruct these Power Models from SPICE output transcripts.

The third and fourth statements corresponded to Time Markers 3 and 4 respectively. In these cases, since the output did not change state, the "half CV squared" component was not subtracted from the energy measurement.

.MEAS *p3* AVG POWER from=*<Time3>* to=*<Time3+50NS>*
.MEAS *cknRI_10* PARAM=*'(p3*50e-9)*1.0e9'*
.MEAS *p4* AVG POWER from=*<Time4>* to=*<Time4+50NS>*
.MEAS *cknFA_10* PARAM=*'(p4*50e-9)*1.0e9'*

Based on the same reasoning, the fifth, sixth and seventh statements could be measured using the following:

.MEAS *p6* AVG POWER from=*<Time6>* to=*<Time6+50NS>*
.MEAS *cknFA_00* PARAM=*'((p6*50e-9)-(0.5*LC*25))*1.0e9'*
.MEAS *p7* AVG POWER from=*<Time7>* to=*<Time7+50NS>*

```
.MEAS cknRI_01 PARAM='(p7*50e-9)*1.0e9'
.MEAS p8 AVG POWER from=<Time8> to=<Time8+50NS>
.MEAS cknFA_11 PARAM='(p8*50e-9)*1.0e9'
```

Originally, each table of energy values was modelled using either a simple surface or linear equation of the form:

K0 + K1 * Capacitance + K2 * EdgeRate
or
K0 + K1 * EdgeRate
where K0, K1 and K2 were constants.

The reason this was done because the early prototype of the power analysis tool did not support tabular data. This turned out to be an issue as we could see from Figure 15, an example where the power distribution for the test vector (cknFA_11 of DFF) across a range of load capacitances and input edge rates were not linear in any predictable manner. Therefore, in this experiment, the energy values were put in a 2-dimension and 1-dimension tables for linear or spline interpolation lookup.

This was just a DFF example. The timing characterization spice decks for the remaining cells were also enhanced in a similar manner.

Spice simulations were then performed on these power characterizing spice decks and the spice results were processed by a set of shell and Perl scripts. What these scripts did were to automatically generate a set of ASCII files (power model for each cell) written in the Power Model Format.

In these power models, those test vectors that resulted in an output change had energy values characterized as a function of the load capacitance and input slews. The test vectors that did not cause an output change were dependent on only the input slews.

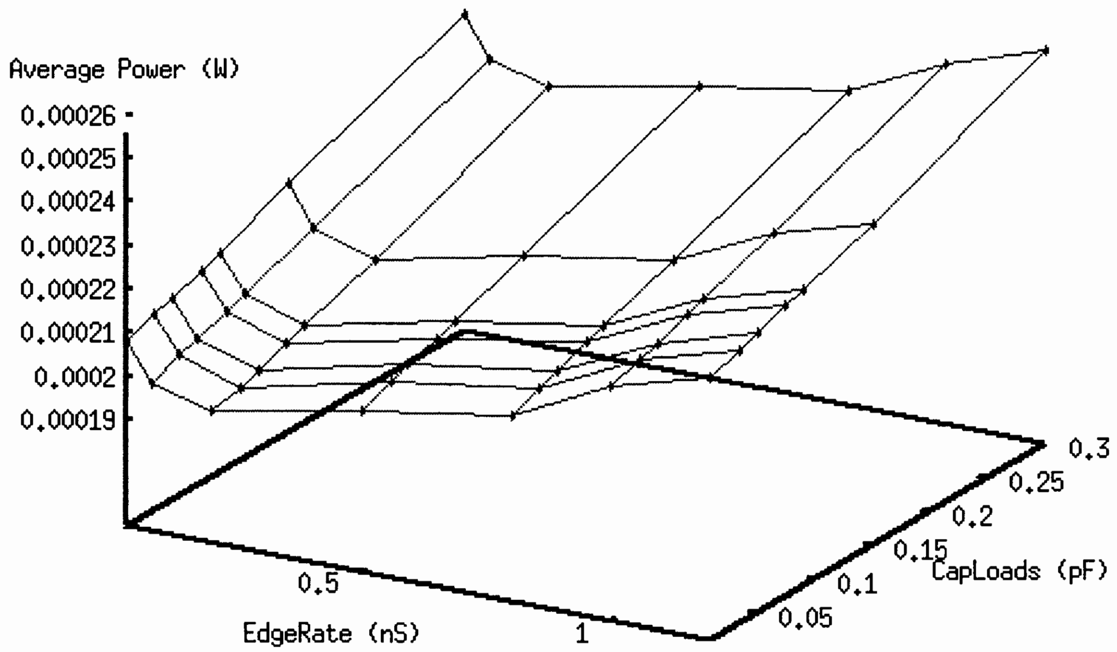

Figure 15. Power distribution for a particular input vector of a DFF

VERIFICATION

Unit Test

After all the cells have been characterized and their power models created, a test bench is created for each cell and tested using the Power Analysis Tool. A set of stimulus is chosen for each cell so that all or most of the characterized input conditions are exercised. The stimulus also has the characteristic where no 2 signals (for cells with 2 or more inputs) are transitioning at the same time or close to avoid glitch situations. The average power is computed and compared with SPICE result which also has unit test circuits generated in a similar manner. Each unit test circuit consisted of just a single instance of the cell, and tested under different load capacitances and input slews.

Results

In general, the error margins were within 5 percent if the characterization and modeling were done correctly. For cases where the error margins were significantly large like 10 percent or more, they were investigated and debugged.

The DFF was one example where the error margin was initially large. On closer look, there were several human errors.

First, the test vectors for the case of data D rising and falling (unshaded in Figure 16) when the clock CKN was either HIGH or LOW were not characterized. Actually, these vectors were overlooked in this experiment because the timing characterization process did not generate these conditions as no timing delay measurements were needed.

Figure 16 illustrates the energy values for the DFF captured at a load capacitance of 0.1pF and input (CKN, D) edge rates of 1ns. Notice that the energy values for the D transitions (unshaded), depending on the state of CKN, were either negligible or in the same order of magnitude as those with an active CKN transition and had an output change (darkest shade). So, ignoring these D transitions would introduce some degree of model inaccuracy.

CKN	D	QN (Pres St)	Energy (J)
\	1	1	5.5262E-12
\	0	0	7.0084E-12
\	0	1	2.6394E-12
\	1	0	3.4007E-12
/	1	1	5.8722E-12
/	0	0	6.0619E-12
/	0	1	3.4562E-12
/	1	0	3.0670E-12
0	/	1	5.3972E-15
0	/	0	4.6353E-15
0	\	1	7.5960E-15
0	\	0	1.5790E-15
1	/	1	3.3157E-12
1	/	0	3.2513E-12
1	\	1	3.2435E-12
1	\	0	3.2285E-12

Figure 16. Energy Values for DFF characterized at 0.1pF load and edge rates of 1ns

Second, recall in Figure 14 that an assumption was made when modeling an inactive CKN:

ckn(/) & d : tab_E3[GETRISE()] ;

ckn(/) & ~d : tab_E6[GETRISE()] ;

The above 2 model statements don't care about the state of QN but from Figure 16 (medium shade) we could see that different power consumption resulted depending on the present state of output QN.

Another model that needed careful consideration during modeling was the tristate buffer with pins en, in and out.

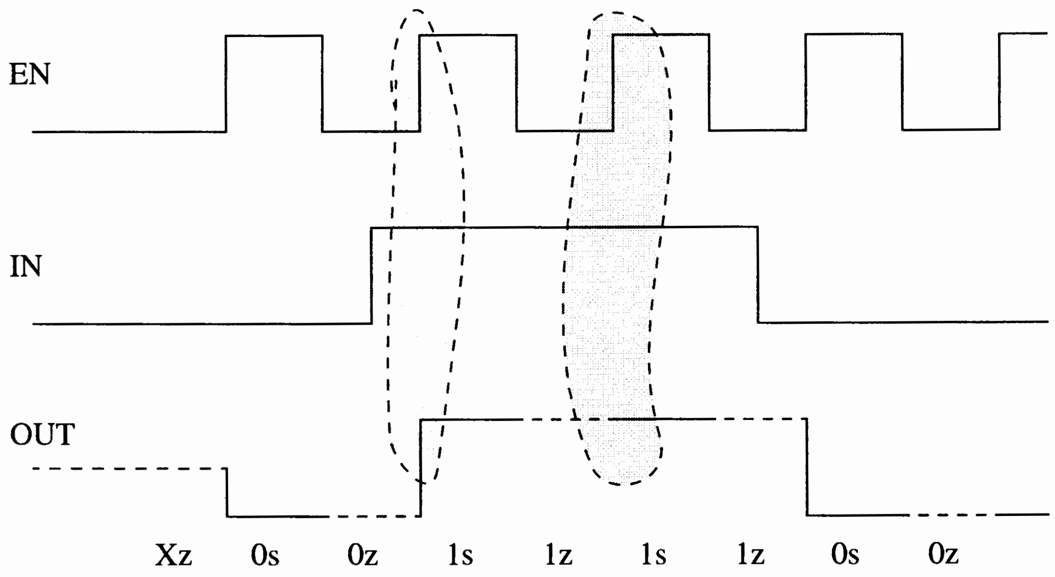

Figure 17. Tristate buffer transition diagram

The model statement concerned was:

en(/) & in : <...energy expression...>

On closer look, two situations fulfilled this statement as shown in Figure 17. The model was missing another state dependent variable to differentiate whether it was a case of output OUT transitioning from 0z to 1s, or 1z to 1s. The energy values between these two cases differed as much as 40 percent over the range of load capacitances and edge rates.

In summary, the experiment had provided some hints on how to create more accurate power models for a set of standard cells. Although the complexity of this cell library ranged from simple to slightly moderate, observations from the power characterization and modeling experiments, were sufficient to generalize a handful of important points worthy of consideration:

- input vectors that produced an output change had energy values quite significantly different from those that did not produce an output change. Timing characterization need not consider the latter, but to have accurate power models, they could not be ignored.
- state dependency of other input pins and internal states was crucial in creating accurate power models. As illustrated in the cases of the DFF and tristate buffer, making certain assumptions to simplify the power model could lead to unwanted inaccuracy error.

Therefore, in order for the power analysis tool to be effective, equal responsibility should also be undertaken to create accurate power models. And finally, it was hoped that this chapter would give an understanding on how to trade-off model accuracy with the amount of effort needed in the characterization process.

CHAPTER IV

ASIC POWER ANALYSIS SYSTEM

INTRODUCTION

This chapter describes the design of a ASIC Power Analysis System (APAS), a dynamic simulation-based power analysis tool for ASIC designs. The dynamic nature of the tool provides not only superior accuracy, but also diagnostics that the designers can use to determine when and where power is being consumed, or wasted.

The non-intrusive technique facilitates the tool to dynamically snap-on to majority of existing ASIC simulators. In fact, APAS has been integrated with three leading ASIC simulators: Quicksim II - schematic-based simulator, Quickhdl - dual language (VHDL-VERILOG) simulator, and Verilog-XL simulator. With the same simulator user interface, the integration is so transparent that the designers can comfortably stay in their familiar simulation environment while performing power analysis of their designs.

Targeting power analysis for ASIC designs at the gate level, APAS features interactive power analysis that will get consistent accuracy near that obtained by transistor level tools, and performance compatible to gate-level simulator speed without power analysis. The consistent accuracy over a wide range of design scenarios is warranted by the quality of power models, the quality of annotated design data, as well as special techniques used to handle glitches and bus contention conditions.

During simulation, simulator will be run in full timing mode to maintain the temporal and partial correlations of input signals to an instance, since the power dissipation behavior of an instance is strongly dependent on these correlations. It's this dynamic nature that makes dynamic simulation-based power estimation method superior in accuracy compared to the static probability-based power estimation method.

APAS estimates power dissipation based on the activity of a simulated circuit over some period of time. For the results of APAS to be meaningful, the input patterns must be representative of the expected usage of the circuit. If input patterns are well chosen and weighted for their expected rate of occurrence, the accuracy of APAS should exceed that of static, probabilistic analysis.

In the first version of APAS, the focus is on gate level modeling and analysis. Higher level (of abstraction) models of the type found in ASIC cell libraries, such as memories, macro cells, will also be supported. However, RTL models do not fit the modeling paradigm presented by this project. Further work will be needed to understand the requirements for RTL models.

OVERVIEW

In addition to the two primary goals: consistent accuracy and compatible (to gate-level simulator) performance, APAS is designed towards the following goals:

• Non-intrusive design: existing ASIC designs used for functional and timing verification do not require any modification for power analysis.

- Dynamic nature: APAS can be dynamically snapped-on to existing simulation environment regardless of design languages or databases.

- Diagnostics/advisory capability: APAS will provide an interactive power diagnostics, advisory, and analysis environment.

Figure 18 shows an APAS overview. Primarily, the host simulator provides functional and timing simulation of behavioral and structural designs. At the same time, APAS (shaded area) which works interactively with the host simulator by accessing the host's user interface (UI), as well as simulation events through the simulator interface, supplements the host simulator functionality by:

- Providing access to power analysis functionality through specialized power commands.

- Providing a power-specific user interface.

- Accessing a library of power models.

- Creating a list of power model instances and managing power-specific instance data.

- Optionally using post-layout, power-specific data values obtained through a Power Analysis Data (PAD) annotation file.

APAS uses a library of power models—one for each unique cell in a given design—to monitor switching activity and to determine the amount of energy required during each switching event. These power models can assess energy dissipation on both input and output changes, as well as monitor static power elements at the cell level.

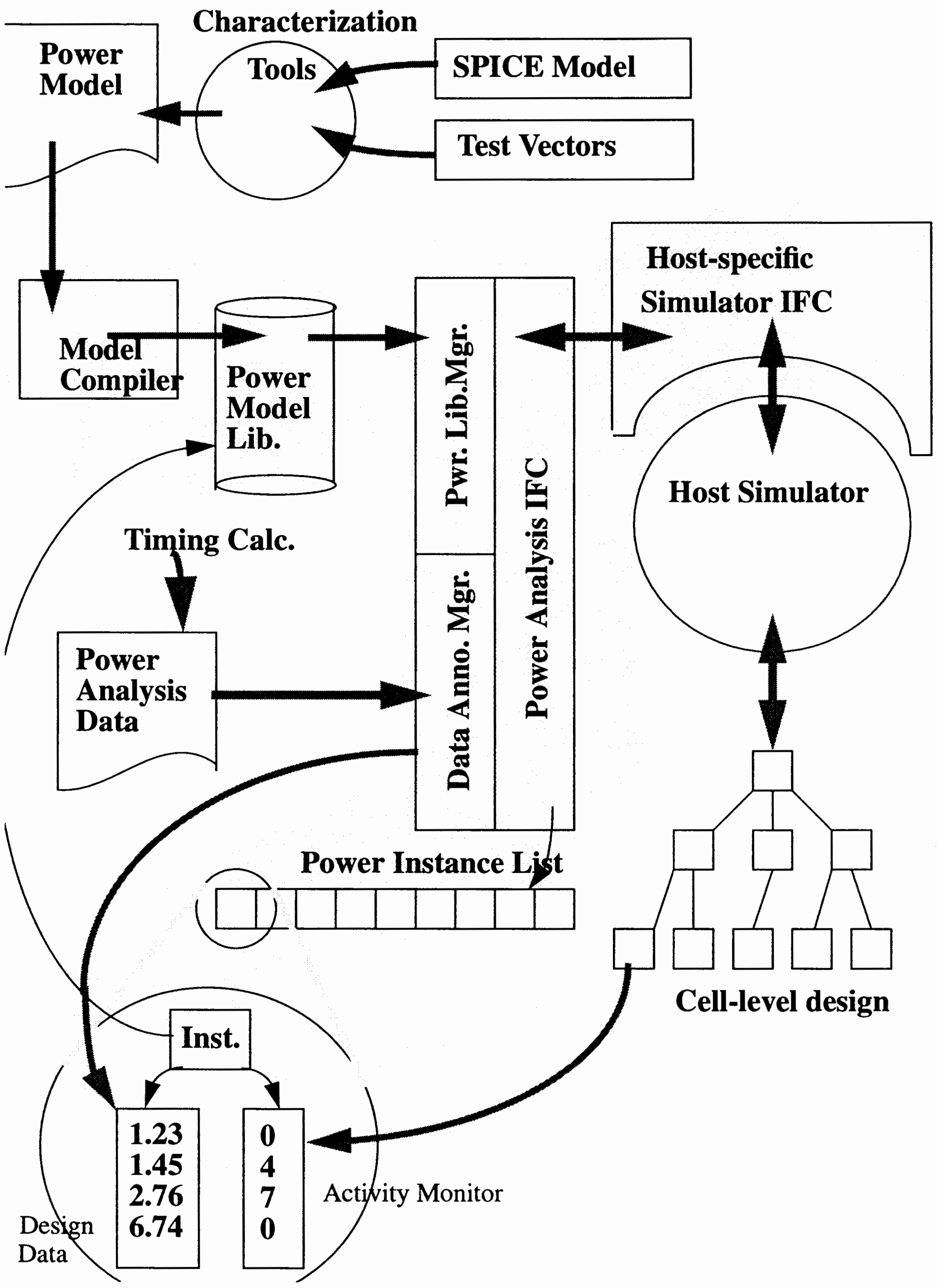

Figure 18. APAS overview

Power models are binary files compiled using the Power Model Compiler (PMC) from ASCII files written in specialized power syntax—the Power Model Format (PMF). Power Library contain compiled power models, and an ASCII catalog file to locate power model binary files and associate them with cells used in the design.

Initially, APAS can perform simulations on a design using default values for power data—specifically, load capacitance and slew rate. The default values can originate from defaults in the power model itself or from standard values provided to APAS. The results data obtained from using defaults could be sufficient to provide a general idea of whether or not the design falls within acceptable power limits.

For more accurate results later in the design cycle, power analysis models require data that are dependent on the context of the instances in the design and their interactions. Specifically, load capacitance on output pins and signal slew at the inputs are necessary for accurately determining the energy required by the cells when the states of the nodes in a design change.

In order to communicate load capacitance and input slews to power model instances, APAS uses an ASCII Power Analysis Data (PAD) file—similar to Standard Delay Format (SDF) used for timing data. The data values in this file provide post-layout values for load capacitance and slew, essentially overriding the default values used in earlier simulations. Data determined in these later simulations help the user to fine-tune the design in order to account for and correct inappropriate levels of power dissipation and waste.

PAD files can be generated by timing calculators—which are usually maintained by ASIC vendors—or synthesis tools. These PAD files make use of data that are extracted when generating delays for timing models.

APAS works interactively with the host digital simulator by accessing the host's user interface (UI), as well as simulation events through the simulator interface.

APAS GENERAL DESCRIPTION

Host Specific Simulator Interface (HSI)

The Host Specific Simulator Interface module is responsible for all the interactions between the host simulator and The Power Analysis Kernel. The HSI is mostly implemented using the host simulator's Application Procedural Interface (API). Most existing ASIC simulator supports a proprietary API, or an industry standard API such as Verilog PLI.

Power Analysis Kernel

The Power Analysis Kernel consists of three modules: Power Analysis Interface (PAI), Power Library Manager (PLM), and PAD Annotation Manager (PAM).

The PAI provides all power specific functions to be called by the HSI to perform power commands. It creates and maintains the power instance list, and the hierarchical power block list.

The PLM and PAM functionality will be described later.

Power Model and Compiler

In APAS, each cell instance is associated with a power model that maintains the information used to assess and track energy dissipation due state changes at the perimeter of the instance. The power models' job is to associate a change on the "pins" of a cell instance with a quantity of energy that is the amount characteristically dissipated when the specific change is made.

The creation and textual format of the power model has been discussed in chapter II. The Power Model Compiler (PMC) compiles a power model into its binary form to be used by APAS. To assist power modeler when creating and debugging power model, PMC can optionally compile with trace mode. With power model in trace mode, power modeler can observe all power events due to any input and or state change.

Power models should have all significant switching events accurately characterized for energy dissipation (see chapter III). Static power dissipation, if significant, should also be included as a function of the state of the cell and the elapsed time in the specified state.

Although State dependency is a cost of redundant information already existed in functional model, it is a feature of APAS power models that improves accuracy and differentiates them from known competitors. Assessing energy dissipation based on known internal state allows better accuracy. We have seen accuracy improvement of 20% in a simple tri-state buffer-driver when the internal node state is known.

See Appendix D for more on the Power Model Compiler. These compiled models are independent of the target simulator, its design language and/or data base. They are also platform independent. A single library should support all simulator environments.

In connection with each simulator functional instance in the design, a power instance is created. This shadow power instance is dynamically connected with its functional instance through power relevant pins. Power relevant pins are those that are used in the corresponding power model. This connection allows power instance to monitor any pin transition that occurs during simulation.

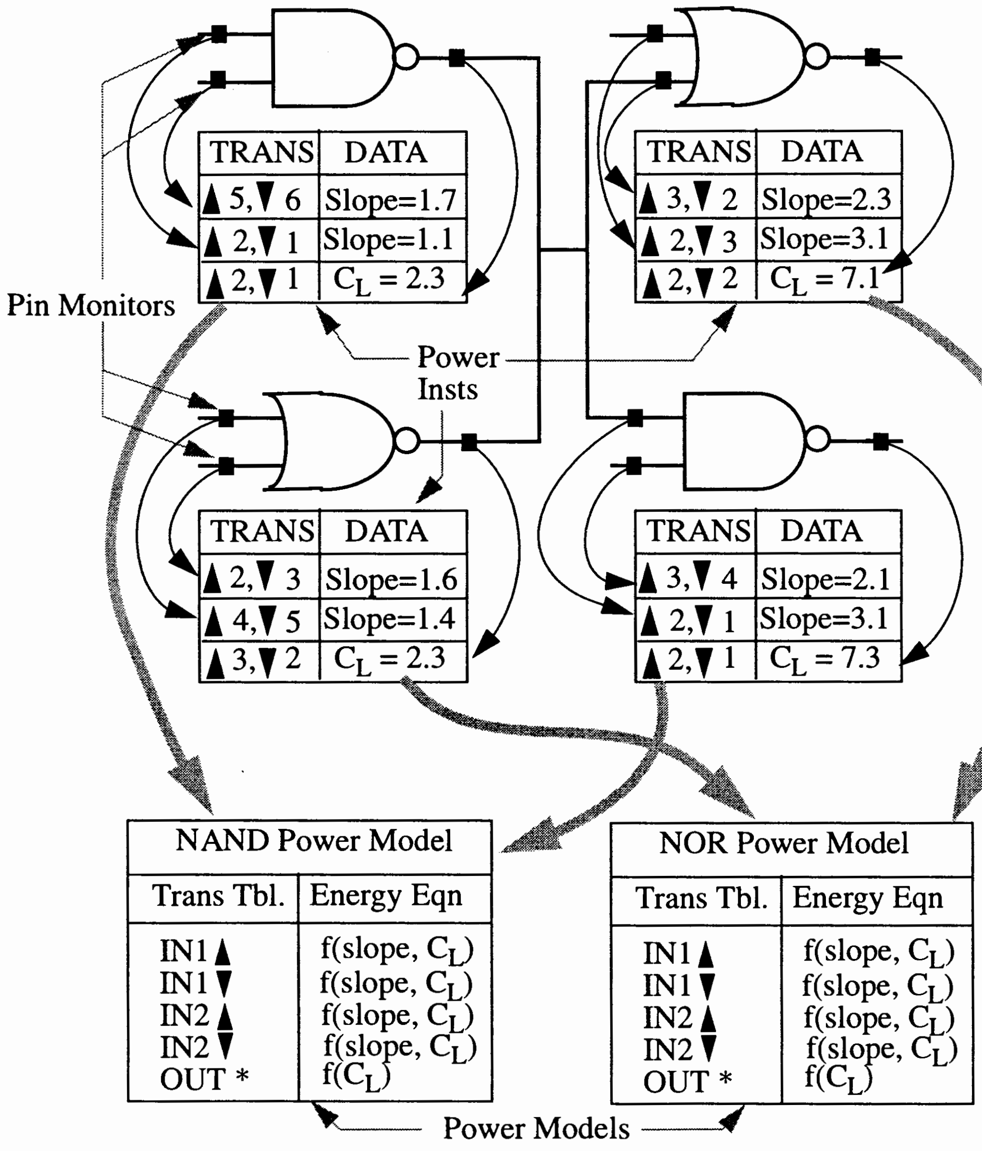

Figure 19. Power instances with their power models

Figure 19 shows how power instances and their power models are associated in a simple design being simulated. The integration point is at the perimeter of instances, since it is necessary to monitor pin activity on all instances in the design.

The power instance will track switching activity (shown in the TRANS columns) of the inputs and outputs. It also maintains any instance specific data called for in the energy equations of the model. This data is obtained in a session when APAS reads a Power Analysis Data (PAD) file.

The common model structures (bottom of figure) are shared by instances of like cell types. Using this structure, power consumption can be made available for each instance at any time during simulation.

To determine average power for an instance over a given time period, the energy dissipation of each transition is determined by the energy equation (usually a function of slew and/or load capacitance) multiplied by the number of the times the transition occurred. The energy quantities for all transitions are summed to give the total energy dissipated by the instance for the current period. If average power for the instance is reported, it is simply the total energy divided by the elapsed time.

Power Model Library

The success of APAS in ASIC design flows is highly dependent on the availability of accurate power model libraries. Normally, ASIC vendors are primary sources of these power model libraries.

The Power Library Manager (PLM) provides the interface to power model libraries. This section describes how a library of power models are cataloged so that the PLM can determine which model to use for a given cell instance. It is our intention that power model libraries support multiple simulators.

When a hierarchical block of a design is activated for power analysis, the PLM will use the Power Library Catalog file, described below, to find power models associated with instances in the block. This file is a part of power model library.

The catalog syntax is flexible enough to accommodate the (existing) library organization preferred by key vendors. Some modelers may treat process variables (voltage for example) as model variables, others may create separate library directories for expected process parameters (perhaps a 2.5v and a 3.3v library). Others may put all models in a single directory and use naming conventions to find the right one.

To find the power library catalog file, the environment variable PWR_LIB should be set to a value that is the path to the directory where the catalog file can be found. A common place to store the catalog file might be in the directory of the library: $PWR_LIB/pwr.cat.

Inside the catalog file, there are several elements needed to find the correct power model.

1) Celltype Key -- A set of strings, properties, parameters and environment variables that identify the cell (and its parameters) so that the power model is uniquely identified for an instance of a cell.

2) Library Path -- A system path that is common to all binary models in the catalog. (This is optional, mainly for readability.)

3) Catalog List -- A set of celltype/path pairs. Where the path is given to each binary power model for a given celltype value.

The syntax of the catalog file will allow the use of design and environment variables in the celltype key and the file paths.

Design variables will be evaluated in the context of the design for each cell.

The syntax of a variable is simply a variable name in parentheses preceded by a `$' symbol.

$(<name>)

When this system function is encountered in a catalog file the value of the named variable is returned. The value is found by searching in the following order:

A. A local variable (defined previously in the catalog file).

B. A design property (or generic in VHDL) found through the simulator interface. In the case of VHDL, the name "entity" will automatically return the entity name of the current instance.

C. An environment variable.

The resolved value is returned as a character string and is inserted in the catalog file in place of the system function.

Below is an example catalog file. See Appendix D, for complete syntax.

```
// define a local variable "celltype" as a concatenation of the
// property value of the MODEL and the value of the VOLTAGE
// environment variable:
CELLTYPE $(MODEL)_$(VOLTAGE)
```

```
// Define the path to the library by environment variables
LIBPATH $(PWR_LIB)/$(PACKAGE)
// define a catalog list that uses the CELLTYPE value, the
// resolved celltype value is then used to find the model.
CATALOG : $(CELLTYPE) {
and2 : $(LIBPATH)/and2.pow
nand2: $(LIBPATH)/nand2.pow
mux21: $(LIBPATH)/mux21.pow
dffr: $(LIBPATH)/dffr.pow ...
}
```

Given the example above, APAS might arrive at a cell, consult the catalog and select a model as follows:

- Determine a value for the local variable CELLTYPE by concatenating the value returned from MODEL, found as an instance property, and the value of environment variable VOLTAGE.
- Next LIBPATH must be resolved by using the values of environment variables PWR_LIB and PACKAGE.
- So, given an AND2 model, and assuming a commercial package at 3.3 volts. The model: $PWR_LIB/com/and2/a2_33.pow is returned.

Power Analysis Data (PAD)

To do accurate power analysis, simulation should be done with accurate timing delays.

For optimum accuracy, post layout parasitic should be used to calculate timing delays. In addition to accurate timing, there is also design dependent data specifically required by the power models: the capacitive load on outputs and input slew.

The PAD Annotation Manager (PAM) reads Power Analysis Data file to receive design dependent data. The PAD file is written in a format that is very similar to the OVI Standard Delay Format (SDF). See appendix B for detail PAD syntax.

SDF files are generated by timing calculators, and it is expected that PAD files will be generated in the same way. Most ASIC design kits use vendor supplied timing calculators. So, in addition to power modeling support, we will also need ASIC vendor's timing calculator to optionally output PAD files as well as SDF files. As with timing, post layout data will give the best results.

Once a model is found for an instance cell, a shadow power instance is constructed that will monitor activity and calculate power in the design context. When all the power instances for the desired analysis have been constructed the next step is to import the data needed by the power models.

The instance specific design data needed by the power models is obtained from a Power Analysis Data file. The example below shows a PAD file. Note that it is almost identical to the Standard Delay Format output by timing calculators. This is by design. This file will be generated in the same way timing delays are, and usually by the same timing calculation tool supplied by ASIC vendors.

The ability to generate the PAD file containing the specific data required by the models in the power library will be a condition of ASIC vendor support. (Pre-layout estimation of this data may be available from synthesis tools too.) Preliminary discussions with several vendors indicate that PAD file can be generated using the same technique, and in most cases the same tools, that generate SDF and other back annotation files.

In the first version of APAS, the PAM will parse the PAD file and send the data directly to specified power instances. This is much faster than annotating design properties (or generics in VHDL) but imposes some limitations on applying the data. No persistent cache of this data is created. So the file must be applied each time the tool is invoked. And since the data is contained in the power instance, the power instances must be constructed to receive their data prior to the time the file is read.

To set up a circuit for PAM this order is important:

1) Invoke the simulator.

2) Enable power analysis in ALL the regions to be annotated.

3) Read the PAD file.

Example PAD file:

```
(PADFILE
    (PADVERSION "0.1")
    (DESIGN "$DESIGN/top_hier_sh/default")
    (DATE "Thu Nov 7 11:19:40 1996")
    (DIVIDER /)
    (VOLTAGE 5.25:5.0:4.75)
    (PROCESS "0.8micron")
    (TEMPERATURE 25 )
    (TIMESCALE 1ns)
    (CAPSCALE 1pf)
    (CELL(CELLTYPE "NAND2")
        (INSTANCE I/$3)
        (POWERDATA
            (SLEW A (3.1:3.4:3.7)(3.2:3.5:3.9))
            (SLEW B (0.9:1.1:1.3)(3.2:3.4:3.5))
            (CAPLOAD Y (3.4))
        )
    )
```

```
        // remaining cells...
)
```

RUNTIME EVALUATION

The basic algorithm for calculating average power for a design block is simply to sum the energy dissipated by each instance in the block during the simulated period and divide by the time elapsed in that period. This section discusses several refinements to that general algorithm.

APAS strives to get more consistent results by recognizing these special cases:

· Nets with multiple drivers

· Bus Contention

· Glitches

Nets with Multiple Drivers

A special case exists when cells with tri-state drivers are driving a net. From the context of the cell model it is difficult to determine energy dissipation when drivers transition from driving state (0 or 1) to a high impedance (called "Z") state.

Assuming no bus contention, that is the design, and its software, are well behaved such that only one driver charges or discharges the bus net at any given time, we can assume that the act of "turning off", that is, going to a Z state does not dissipate any of the energy stored on the net. For this reason all transitions to Z will be ignored by APAS.

When a driver transitions from a “Z” to some logic state, the energy needed to “drive” the net depends on the current state of the net. If the “turning on” of the driver causes the net to change state then the amount of energy associated with a transition (usually 1/2 *C*V**2) should be assessed. But, if the net does not change state, no current will flow and the transition should be ignored.

APAS will do the needed analysis of the net to make the right decision if it is possible to get the net state from the design interface of the host simulator. If the net state is not available, we will assume that there is a 50% chance that the node will change state when a driver turns on, and 50% of the transition energy will be assessed.

Bus Contention

Another special case that occurs on bus nets is called “contention”. Contention exists when more than one driver is attempting to drive a net. Most host simulators are capable to report warnings for illegal contention.

Some forms of contention may not be considered bad design practice and may be allowed. For example, all active drivers driving the same state may be fine. Contention during transitions, for a very short period of time, may also be allowed. Still, bus contention may noticeably affect the power dissipation of a design.

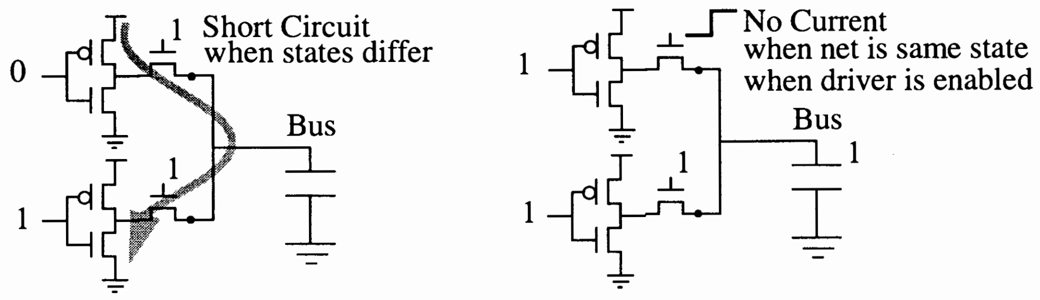

Figure 20. Bus Contention

Figure 20 shows some situations where bus contention affects power consumption. In the first figure, a short circuit is created when drivers connected to the same net are driving different values.

APAS will allow modelers to estimate the power dissipation as a function of the time elapsed while in this condition, similar to static power elements. This can be specified in the model language. In most cases, if this condition persists the chip will fail. But this condition may exist briefly in transient conditions.

The second figure shows the case when more than one driver drives a bus high, in this case it is inaccurate to assess the energy used to charge the net twice (once from each instance). On nets with multiple drivers, APAS will not assess energy when there is already a driver in the same state.

Glitches on Drivers

We have seen estimates that the power used by glitches (which generally do nothing for functionality) can approach 25% of the total. The appearance of a glitch in ASIC designs is often dependent on timing conditions. Glitches that appear with one set of conditions may disappear in another.

The effect of a glitch is usually to cause incomplete transitions on nodes. That is, the voltage change (dV) is less than Vdd. Glitch power can be expressed as:

$$P_{glitch} = \frac{1}{2} C V_{dd} dV \qquad (16)$$

Where:

dV : Reduced Voltage Swing

Voltage swing is dependent on the ratio of glitch width (tg) and gate delay (td), hence can be approximated as:

$$dV = \frac{tg}{td} V_{dd} \qquad (17)$$

In theory, dV is dependent on input slew, output load (figured in tp) and input pattern (figured in tg), however in engineering trade-off between efficiency and accuracy of glitch power computation at gate level, the ratio (tg/td) can be further approximated to a reasonable constant value.

In designs at the cell level, glitches may be hidden inside the cell (or macrocell). In this case APAS must depend on the cell model's characterization of internal energy dissipation to assess the energy of the internal glitch as it will not be seen at the cell's perimeter. This may not be possible when the glitch is due to multiple inputs changing at nearly the same time since this scenario is difficult to characterize, and so, is not modeled. But when internal glitches occur with specific input slew or intrinsic delay (based in load capacitance) their effect can be characterized.

Glitches on drivers (output pins) may be represented in digital simulation by turning on detection algorithms usually called “spike checking”.

APAS wants to recognize glitch occurrence during simulation so that glitch power is computed instead of normal switching power. In general, glitch behaviors are modeled in logic simulators with a “three band” model. Which band a glitch falls into is determined by its pulse width. When a glitch is narrow enough to fall into the first band it is “swallowed” and nothing is propagated. Pulses in the second band represent partial transitions and an X state is output to indicate the uncertainty that a threshold voltage was reached. Pulses in the third band are wide enough that threshold voltage is certain and propagated (transported) intact.

APAS will be unaware of first band (swallowed) glitches and ignore them. Since these glitches are small, accuracy will not be greatly affected. Third band (transported) pulses will be seen as full transitions, which they are by definition.

It is the second band glitches, that generate X pulses, that APAS will treat as a special case. Instead of assessing the energy of a full transition when a 1 or 0 to X transition is encountered, APAS will use 50% as the energy of an average partial transition. This quantity was determined by empirical means and appears to approximate the size of the average glitch.

KEY FEATURES

APAS implements its I/O tasks through the user interface of the host simulator. The UI features of APAS fall into the following categories:

• Command Set -- Basic commands to setup and report on a power analysis session. The command set is casted in the userware format of the host simulator. The Command Interface section later will describe in details each available command and its functionality.

• Textual Reports -- A textual report of average power calculated as specified by the user's setup process. This data may be sent to a file and may also be viewed from the simulators' UIs.

• Graphic Results -- A histogram of average power over a specified time periods is available for each block being monitored. This feature is useful when power consumption is being calculated over short periods of time.

Textual Reports

The primary output of APAS is the average power dissipation during the simulation run. Depending on the user's setup specifications, more detailed analysis may be presented:

· Total power broken down by type (i.e. dynamic and static as indicated in the model).

· Power dissipation of specific regions of the design.

· Power dissipation during fixed time periods.

A report may be generated at any time during the simulator session by issuing a POWER REPORT command. Using the POWER FORMAT command prior to POWER REPORT will allow the user to specify one of several formats so that it can easily be read or parsed by a post processing utility.

Reporting Dynamic and Static Power

The APAS model format allows the classification of energy dissipation to be either dynamic (due to state changes in the circuit) or static (due to leakage or static short circuit conditions). Power dissipation due to one or both of these factors can be reported as well as the sum of the two. The choice is made when the POWER FORMAT command is issued.

Linear Format

This format keeps each data element in an individual line for text parsing. Its syntax is:

<block name> <period start time> <period end time> <dyn. power> : <static power> : <total>

Where the block name is the hierarchical path to an element being monitored. (Any and all of those elements explicitly monitored prior to simulation will be reported.) The period times tell the simulation period over which power was calculated. And the fields for power results are filled in with the average power in the block for each category over the timing period indicated. If a power category is not being reported (see above) it is left blank.

Example:

```
/      0.0 20000.0 7.65mW:2.05mW:9.70mW
/ALU   0.0 20000.0 3.69mW:1.07mW:4.76mW
/ALU/A 0.0 20000.0 1.12mW:0.43mW:1.55mW
/ALU/B 0.0 20000.0 1.03mW:0.22mW:1.25mW
/ALU/C 0.0 20000.0 0.97mW:0.21mW:1.28mW
/ALU/D 0.0 20000.0 0.57mW:0.21mW:0.78mW
```

Hierarchical Format

This format is meant to be more readable. Data for all time periods of each hierarchical block being reported are grouped together. Average power for the total time is always the last entry for each block.

Example:

```
Average Power for: /ALU
Period:  0.0:100.0   Dyn: 1.12mW  Static: 0.43mW  Total: 1.55mW
Period:  100.0:200.0 Dyn: 1.13mW  Static: 0.42mW  Total: 1.55mW
Period:  200.0:300.0 Dyn: 1.14mW  Static: 0.41mW  Total: 1.55mW
*Period: 0.0:300.0   Dyn: 1.13mW  Static: 0.42mW  Total: 1.55mW

Average Power for: /ALU/A
Period: 0.0:100.0 Dyn: 0.12mWStatic: 0.13mWTotal: 0.25mW
Period: 100.0:200.0 Dyn: 0.13mWStatic: 0.12mWTotal: 0.25mW
Period: 200.0:300.0 Dyn: 0.14mWStatic: 0.11mWTotal: 0.25mW
*Period: 0.0:300.0:Dyn: 1.13mWStatic: 0.12mWTotal: 0.25mW
```

Time Period Format

This format organizes the data by time period, and hierarchically within each period, similar to the graphic display (See below):

```
Period: 0.0:100.0
/ALUDyn:    1.12mW  Static: 0.43mW  Total: 1.55mW
/ALU/ADyn: 0.12mW  Static: 0.13mW  Total: 0.25mW
/ALU/BDyn: 0.13mW  Static: 0.12mW  Total: 0.25mW
/ALU/CDyn: 0.14mW  Static: 0.11mW  Total: 0.25mW
/ALU/DDyn: 0.39mW  Static: 0.36mW  Total: 0.75mW
```

```
Period: 100.0:200.0
/ALUDyn:    5.12mW  Static: 0.43mW  Total: 5.55mW
/ALU/ADyn: 1.12mW  Static: 0.13mW  Total: 1.25mW
/ALU/BDyn: 1.13mW  Static: 0.12mW  Total: 1.25mW
/ALU/CDyn: 1.14mW  Static: 0.11mW  Total: 1.25mW
/ALU/DDyn: 1.39mW  Static: 0.36mW  Total: 1.75mW
```

Graphic Display

In order to display the time based data above in a graphical format, alongside the signal waveforms of the simulation, we have connected the periodical data to the list window in and to a charting window in the host simulator to give a graphic indication of power usage in the form of a histogram.

This feature is useful when power consumption is being calculated over short periods of time. A histogram of average power over each time period is available for each block being monitored.

The ability to print, save and review graphical data is dependent on the existing features of the host UI.

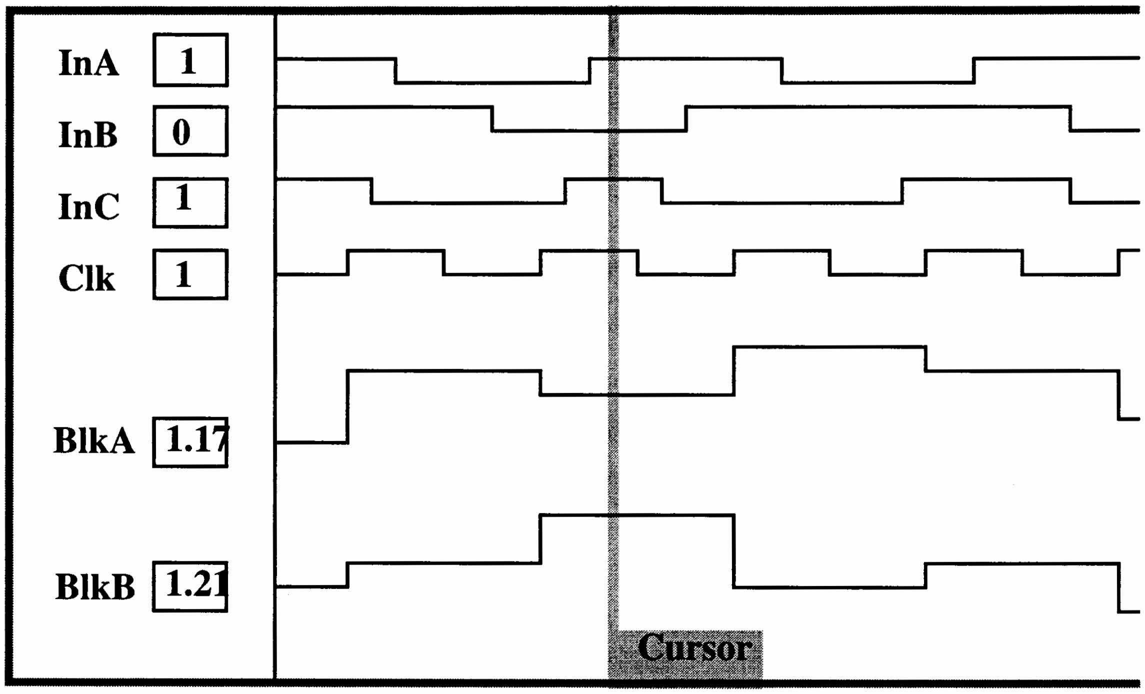

Figure 21. Histogram of power per cycle

Additional Diagnostics

Other diagnostics supported by APAS are listed below:

1) Tracking "greater that expected" activity, such as more than one transition per clock cycle on cells.

2) Keeping more detailed profiles of the most active periods. More detail might include more hierarchical information, split out by type, etc.

4) Tracking instances with high input to output toggle ratios in second logic levels and up.

Command Interface

The rest of this chapter is now used to describe the set of available power commands that are added to the command interface of the host simulator for controlling APAS. To insure uniqueness, all command are preceded with the keyword POWER. Note that, in the commands all significant letters are shown in UPPER case.

POWer ANAlyze *blk_list* [-ON | -OFF] [-Warn | -NOWarn]

Description

Turns on or off a hierarchical block for power analysis.

The Power Analyze command designates that one or more hierarchical blocks in a design will be analyzed for power during a simulation. The *blk_list* argument lists the design paths of the hierarchical blocks.

Arguments

blk_list

A list of named or selected hierarchical blocks to be analyzed.

-ON

An optional switch that enables power analysis for the specified hierarchical block. This is the default.

-OFF

An optional switch that disables power analysis for the specified hierarchical block.

-Warn

Displays a warning message indicating that one or more models cannot be found. This is the default.

-NOWarn

Disables warning messages regarding models that cannot be found.

POWer ANNotate *filename hier_path* [-Min | -Typ | -Max]

Description

Loads design data from the file designated by *filename.*

The Power Annotate command loads design data contained within a Power Analysis Data (PAD) reference file. Minimum, typical, or maximum data values can be specified for loading.

Arguments

filename

A required string argument specifying the name of the PAD file from which to load power data.

hier_path

Optional string specifying the relative path to a hierarchical block that needs power annotation data attached to it. Default: "/" level.

Pathnames in the PAD file must be relative to the "/" level in the design.

-Min

An optional switch specifying that minimum values should be used when triplets are present in PAD file data fields.

-Typ

An optional switch specifying that typical values should be used when triplets are present in PAD file data fields. This is the default for the initial simulation run.

-Max

An optional switch specifying that maximum values should be used when triplets are present in PAD file data fields.

POWer CLear

Description

Clears all power data accumulated up to the current simulation time.

The Power Clear command resets all the power-related data that has accumulated during the simulation and resets the time for further power calculations to the present time. If a Power Clear command is not issued, all the existing power data to the present time will remain during a subsequent simulation, and new power data will be averaged into it.

POWer FOrmat -<u>POWER</u> [-<u>DYNAMIC</u> | -NODYNAMIC] [-<u>STATIC</u> | -NOSTATIC] [-<u>TOTAL</u> | -NOTOTAL]

Description

Sets the format for textual reports.

The Power Format command determines the format that the simulator must use for textual reports when a Power Report or Power Sort command is issued. Average current can be designated instead of average power. If information on average power is required, the type of power can be specified—dynamic, static, and/or total.

Arguments

-<u>Power</u>

An optional switch that specifies average power as the type of data to report. Default.

-<u>Dynamic</u>

An optional switch specifying that dynamic power is to be reported. Default.

-Nodynamic

An optional switch specifying that dynamic power should *not* be reported.

-Static

An optional switch specifying that static power is to be reported. Default.

-Nostatic

An optional switch specifying that static power should *not* be reported.

-Total

An optional switch specifying that total power is to be reported; that is, power consisting of both dynamic and static power. Default.

-Nototal

An optional switch specifying that total power should *not* be reported.

POWer LIbrary *path*

Description

Specifies the pathname to a power library directory if PWR_LIB is not set.

The Power Library command designates where the power library manager can find a power library directory if the PWR_LIB environment variable was not set prior to invocation

Arguments

path

Designates a pathname to the power library that contains the catalog file.

POWer PEriod ***period***

Description

Establishes the time period for calculating average power.

The Power Period command is most commonly used with the Power Trace command and determines the time period in nanoseconds over which average power is to be periodically calculated during a simulation. If the time period is set to 0.0 ns, the time period is effectively infinite. A positive nonzero number specifies that the simulator should calculate average power for each period of time during a simulation.

If the Power Start command has been used to specify a particular start time, then the first time period will begin at that start time. If the simulation continues beyond the first time period, then average power will be calculated for each successive time period possible for the duration of the simulation. The time period can be changed during a simulation session.

Arguments

period

A required real number that specifies the period of time over which average power is to be calculated. Default: 0.0 ns.

POWer REport ***blk_list*** **[-Out*filename*] [-<u>NOUPDATE</u> | -UPDATE] [-<u>NOAVERAGE</u> | -AVERAGE]**

[-<u>PRESENT</u> | -NOPRESENT] [-<u>NOMAX</u> | -MAX] [-<u>NOALL</u> | -ALL]

Description

Displays power analysis results for a specified hierarchical block.

The Power Report command outputs power analysis results to the simulation user interface (UI) or to an output file; the default is to send it to the UI. The results are written out for a specific hierarchical block or blocks. The -Out switch specifies that the report is to be placed in a file located by the *filename* path.

You can choose to have the simulator automatically remove accumulated power data when starting a new simulation, to report average power data, to report present power when the simulation stops, to report maximum power during the simulation

Arguments

-Out *filename*

An optional label and string argument that specifies an output file to which the power results for the hierarchical block are to be written.

-<u>Noupdate</u>

Switch specifying that power data are not to be reset to zero (0) for further simulation; that is, power data continues to accumulate in subsequent simulations. Default.

-Update

Switch specifying that power data are to be reset to zero (0) for further simulation.

-<u>Noaverage</u>

Switch specifying that average power will not be reported. Default.

-Average

Switch specifying that average power will be reported.

-Present

Switch specifying that the present power will be reported. Default.

-Nopresent

Switch specifying that the present power will not be reported.

-Nomax

Switch specifying that the maximum power will not be reported. Default.

-Max

Switch specifying that the maximum power will be reported.

-Noall

Switch specifying that power is not to be reported for all the blocks in the design that are being analyzed for power; requires a selected or named instance. Default.

-All

Switch specifying that power is to be reported for all the blocks in the design that are being analyzed for power.

POWer SOrt *blk* [-PRESENT | -AVERAGE] [-Top *number*]

Description

Sorts the instances in a specified block by the amount of power dissipated.

The Power Sort command lists the instances in a block from the most power dissipated to the least power dissipated. Either average power or present power can be requested. The number indicates how many of the most power-intensive instances should be reported.

Arguments

blk

Specifies the name of the hierarchical block containing instances that are to be sorted from most to least power dissipation.

-Present

Switch specifying that only present power should be sorted. Default.

-Average

Switch specifying that only average power should be sorted.

-Top number

Label and number pair specifying how many instances should be reported for highest dissipation of power.

POWer STARt ***start***

Description

Sets power analysis to begin at a specified absolute time.

The Power Start command causes the power analysis to begin at a user-defined time, which is in nanoseconds and defaults to 0.0 ns. This command is especially helpful when used to avoid an initialization stage during which part or all of a circuit may begin in an unknown state (X). This enables the user to avoid inaccuracies caused by the unknown states.

Arguments

start

A required real number that specifies the absolute time at which to begin power analysis. Default: 0.0 ns.

POWer STATus

Description

Lists the status of APAS commands.

The Power Status command returns information regarding the present simulation time, the types of power to be reported, the start time, the period, and the level at which power analysis and power tracing is enabled.

POWer TRace *blk_list*

Description

Creates a waveform in the Trace window for each of the specified instances.

The Power Trace command enables for power analysis all instances designated in the *blk_list* argument, even if they have not been explicitly specified in the Power Analyze command. A waveform is then created and displayed in the Trace window for each instance.

If the end of the power trace coincides with the end of a power period, the trace will not be updated until the simulation time is advanced slightly. Thus, a Power Report -Present command will show the current values of the reported blocks, but a Power Trace command will show only the value that occurred during the last power period.

Arguments

blk_list

A list of named or selected hierarchical blocks to be traced.

POWer UPdate

Description

Asynchronously ends a period, calculates average power, and updates traces.

The Power Update command enables the period in a simulation to be reset to zero (0) without resetting the start time or the accumulated power data—the power data that has accumulated continues to be averaged in during subsequent runs. The power traces are also updated to include average power data. Power Update can be used instead of Power Clear to change the period in the middle of a simulation without losing previously-accumulated power data.

This command can also be accomplished by using the -Update switch in the Power Report command.

CHAPTER V

EXPERIMENTS AND RESULTS

INTRODUCTION

This chapter reports the results of our experiments to measure the power estimation accuracy, performance and capacity of the ASIC Power Analysis System.

Accuracy, the ultimate goal, is compared against the well known, traditional circuit simulator - SPICE, a proxy for real silicon measurement. Since performance, our secondary design focus, is comparable to gate-level simulator which is typically several orders of magnitude faster than circuit-level simulator, only small circuits with short test patterns are measured against SPICE. For larger circuits with long test patterns, performance will be measured against the gate-level simulator into which APAS engine is integrated. Besides being capable to handle mainstream ASIC design sizes which are typically in the range of tens of thousand gates [5], we also attempt to access the tool at higher capacity, 250K gate design [27].

In addition to the three primary design goals APAS, as a dynamic snap-on engine to existing ASIC simulators, will also be measured against the integrated simulators: Quicksim II and Quickhdl to assure that the accuracy is consistently maintained, and performance, capacity is comparably driven by its host simulator.

To obtain the high degree of consistency, top down design method which is the preferred method in ASIC design, especially when design complexity exceeds 10,000 gates, is applied. At the top level, design circuits, based on design specification and requirements, are functionally described and simulated. Then they are synthesized and technology-mapped into the 0.8 micron standard cell library. This selected standard cell library contains different models for circuit-level and gate-level simulators: SPICE (SPICE CMOS model), VHDL(VITAL model), and Quicksim II (AMP model). The synthesis environment allows writing resultant designs targeting different simulator's design descriptions. In companion with the standard cell library, there is a power model library which have been pre-characterized through our power characterization process. Pre-layout design parasitic described in PAD format is also generated from the synthesis process.

SETUP AND ENVIRONMENT

The experiments are conducted on a Sparc 10 with 256Mb of RAM and an available swap space of at least 500Mb. This environment represents today's typical design environment in the industry.

Below are the configuration profiles of the machines used in the experiments:

• Uname -a: SunOS curly 4.1.3 1 sun4m:

Hostname: curly
OS release: 4.1.3
CPU: SPARC_10
Memory: 256 MB
Swap: Total swap: 830 MB

• Uname -a: SunOS hercules 4.1.4 2 sun4m

Hostname: hercules
OS system: SunOS
OS release: 4.1.4
CPU: SPARC_10
Memory: 256 MB
Swap: Total swap: 625MB

Before the start of every new experiment, the RAM is "chilled". Disk usage is measured by making a UNIX call to /etc/pstat with option -s, and runtime measurement is done by calculating the elapsed time between calls to the UNIX function /bin/date.

EXPERIMENTS

Accuracy

The accuracy of our power estimation is mainly dependent on the following factors:

• The accuracy of event-driven logic simulation.

• Timing accuracy of logic simulation

• The accuracy of the power characterization procedure which has been fully discussed in chapter III.

In addition, special computation techniques in consideration of special conditions such as bus contention and glitching also improve the accuracy in exchange for some acceptable performance costs. We will test accuracy in these special cases as well as general cases.

Bus Contention

Description

The first test uses just a single tristate buffer to do accuracy calibration. Using an edgerate of 1.2NS and a load of 0.05pF

The second test is a simple design with 3 tristate drivers driving a common output net with load 0.05pF.

The first set of stimulus, 3drivers.pat, simulates a good condition where, *ena, enb, enc* take turns to drive the net alternating 0 and 1.

The second set of stimulus, contention_eqstate.pat, is similar to the first except that the *ena* and *enb* signals are skewed a little. *ena* and *enb* will both be active for 5ns, and the driven inputs are both the same either driving to *gnd* or both driving to *vdd.*

The third set of stimulus, contention_opstate.pat, is similar to contention_eqstate.pat except that the inputs *ia* and *ib* are also skewed so that during the contention period, *ia* and *ib* are at opposite states.

The last set of stimulus, contention_ueqdelays.pat simulate the X region created due to unequal delays when 1 driver is turning off say H to Z and this delay is longer than the delay of another driver turning on the bus from Z to L. In SPICE, this momentary X region caused a VDD to GND path, static power. The sdf file contention_ueqdelays.sdf was used to simulate this effect.

This test case uses tribufL, a VITAL model in c8lib_vital.power, with it's own power model, tribufL.pwr. This power model is characterized during the development stage of c8 lib, 0.8 micron power model library.

Result

Table 1: Bus Contention

	APAS	SPICE	%Error
calibrate	8.962e-05	8.9747e-05	1.2%
3drivers	1.030e-04	1.0809e-04	5.3%
contention_ eqstate	1.177e-04	1.1661e-04	0.9%
contention_ opstate	1.166e-04	1.277e-04	9.5%
contention_ ueqdelays	1.186e-04	1.564e-04	5.2%

Table 1 shows that even in the bad design case, contention_opstate, when a short-circuit current flows in the net, the accuracy is still within 10% of SPICE.

Glitch Power

Description

This test case depends on the inv1x, nand2H, and and2H power models. The design is a simple glitch generator circuit, mux2. Pin *sel* is toggled with inputs *a* and *b* tied at 0. Glitch will occur during sel rising. Experiment is carried out 4 times for different loads of *Y*.

Table 2: Glitch Power

	APAS	SPICE	%Error
glitch1	4.382e-04	4.220e-04	3.8%
glitch0.5	3.646e-04	2.960e-04	3.1%
glitch2	6.158e-04	6.803e-04	10.5%
glitch4	9.668e-04	1.1232e-03	14%

Glitch power, a partial transition switching power, is dependent on load capacitance. The accuracy shown in table 2 is mostly within 10% of SPICE except case when load is significantly large (glitch4, 14% Error).

General cases

Description

In these general cases which represent general operating conditions in design circuits, most selected circuits are synthesized using Autologic II, except small and simple circuits which are hand-crafted. Some tests are simulated using post-layout data when it is available from AutoCells. Otherwise pre-layout data estimated from synthesis environment will be used. No matter what design data, pre or post layout, is used, SPICE representation of the circuits must be equivalent to the gate-level descriptions.

ca - accuracy_chainbuff.qs.power

The circuit is a chain of 16 buffers and inverters. Power models consist of inv12x, inv1X, buf1. The PAD file data, test.padf, is consistent with the design data specified in the spice deck chainbuff.sp. Capload in PAD file are computed by adding net capacitance and inpin capacitance:

- inv12x inpin cap is 0.0716pF
- inv1x inpin cap is 0.0112pF
- buf1 inpin cap is 0.0186pF

All net capacitances used in interconnecting nets are 0.25pF. Each Input slews are approximated from driving output slope. Output slopes are taken from c8 data book, at characterized points:

- inv12x (0.25pF) 0.27 0.17 (0.50pF) 0.46 0.28
- inv1x (0.25pF) 1.48 0.98
- buf1 (0.25pF) 0.37 0.41

cb - accuracy_dec24.qs.power

The circuit is a 2 to 4 decoder synthesized to gates using Autologic II, AL2. The result is a 5 cell combinatorial circuit. Power models consist of nand2H, inv1X, or2H. The PAD file data in padf is consistent with the design data specified in dec24 spice deck dec24.sp. All outputs of cells are modelled with 0.25pf, all primary input slews are modelled with 0.8ns except /g1/a and /g2/a where they are driven by /g2/O has slews 1.48, 0.98 based on c8 data sheet of 0.25pf load. In this test, we used a random but periodic test patterns and did not figure out the toggle counts of each cell. The periodic test sequence is (ab) 00->01-

>11->10->00

cc - accuracy_simple.qs.power

d1 is a single inverter IV1, and d5 is 5 instantiations of the inverter with separate inputs and outputs. The PAD file data in padf1, padf5 are consistent with the design data specified in the spice deck design1.sp and design5.sp. Input slopes are 0.4ns and load is 0.12pF, a characterized point in the power model of IV1.

cd - basicalu4.power

The circuit is a 4bit ALU synthesized to gates using AL2. The synthesized circuit contains 255 gates of the following cells: and2H, and3H, buf1, inv1X, inv12X, mux2L, nand2H, nor2H, or2H, or3H, xor2L.

All caploads are post-layout data generated using AutoCells.

ce - basiccrc32.power

The 32-bit CRC with its RTL model at crc32.vhd is synthesized to gates using AL2, and the gate representation contains 1060 gates of C8_LIB cells: and2H, and3H, buf1, dffr, inv1X, inv12X, mux2L, nand2H, nor2H, or2H, or3H, xor2L.

cf - basicshift4.power

The circuit is a 4bit shift register synthesized to gates using AL2. It contains 52 gates of C8_LIB cells: and3H, buf1, dffr, inv1X, inv12X, nand2H, nor2H, or2H

All caploads are post-layout data generated using AutoCells.

Result

Table 3: Power Estimation Accuracy

Circuit	gate count	APAS (mW)	SPICE (mW)	%Error
ca	16	6.90	6.65	3.6%
cb	5	1.03	1.07	4.9%
cc	5	1.08	1.04	3.2%
cd	255	34.97	36.66	4.6%
ce	1060	41.08	37.54	9.4%
cf	52	1.31	1.43	8.1%

Table 3 shows that for the selected circuits which span from 5 to 1060 gates, combinational or sequential, the accuracy of APAS is within 10% of SPICE.

Performance and Capacity

APAS compared against SPICE

In this experiment the following two circuits are selected:

cd - basicalu4.power

cf - basicshift4.power

Table 4 shows the speed advantage over SPICE and the degradation in simulation speed over original logic simulator without power.

Table 4: APAS vs. SPICE Performance

Circuit	APAS	Quicksim II	SPICE
cd	1.0	0.67	26.4e3
cf	1.0	0.68	9.17e3

APAS is 4 order of magnitude faster than SPICE and suffers only 33% speed degradation compared to the base logic simulator, Quicksim II.

perf15K_alu32.power

Description

The design is a 32bit ALU synthesized to gates using Autologic II and targeted to the 0.8 micron technology library that was internally characterized for power. A design data annotations file is size of 5Mbytes. The run is for 100,000ns. This circuit contains 15050 gates of C8_LIB cells: and2H, and3H, buf1, inv1X, inv12X, mux2L, nand2H, nor2H or2H, xor2L. The structural vhdl description contains three blocks: addsub block, multiplier block, division block.

Performance and capacity tests are done in several areas:

- measures the disk usage and runtime performance for the power annotation operation
- measures the disk usage and runtime performance for the default power sort operation
- measures the disk usage and runtime performance for the power load operation

• measures the disk usage and runtime performance for a RUN (long time) operation, and compare it with the same RUN without the power option. Also compares with RUN's for different power period values of "half a clock" (implies 6000 power calculations), "a clock cycle" (implies 3000 power calculations), and "full simulation run time" (1 power calculation).

• measures and compares QuickSim II / QuickHDL invocation with and without power option.

Result

Table 5: alu32 Performance

Tasks	QuickSim II	QuickHDL
w/o power option		
Invoke	00h 10m 30s::73.3Mb	00h 01m 14s::50.8Mb
Run	00h 01m 53s::0Mb	00h 18m 47s::1.4Mb
with power option		
Invoke without Power	00h 10m 33s::86.6Mb	
Invoke with Power	00h 10m 47s::75.8Mb	00h 01m 17s 53.5Mb
Load Power	00h 00m 04s::2.5Mb	NA
Sort	00h 00m 14s::148Kb	00h 00m 03s::912Kb
Load Annotations	00h 00m 26s::12Kb	00h 00m 26s::920Kb
Power Analyze 15K	00h 00m 30s::3.8Mb	00h 00m 10s::7.8Mb
Run (period = phase)	00h 24m 06s::0Mb	00h 40m 35s::9.2Mb
Run (period = clock)	00h 13m 47s::0Mb	00h 38m 31s::9.2Mb
Run (period = Full)	00h 02m 42s::0Mb	00h 33m 43s::9.2Mb

In general, the power estimation accuracy from Quicksim II and QuickHDL are the same since both simulators are integrated with the same power kernel using the same power model library, the same power design data, and equivalent timing and functional models. However, the performance and capacity (memory overhead) are dependent on the host simulator. Some host simulator such as Quicksim II allows a tight integration which results in better performance, less memory overhead, but less flexibility to maintain. Table 5 shows that a degradation of 30% in power simulation with Quicksim II, and 44% in Quickhdl. Design data annotation operation is at the speed of 11.5Mbytes per minute.

CHAPTER VI

CONCLUSION AND FUTURE WORK

We have developed an ASIC Power Analysis System (APAS) in this thesis. APAS is capable of supporting the following major features:

. Snaps on to the leading ASIC simulators: - QuicksSim II - QuickHDL - Verilog-XL with a common library.

. Easy to learn: Start analyzing your design for power in a single day.

. Interactive power diagnostics tell when and where - not just how much.

. Performance and capacity for today's complex designs (33% degradation).

. Flexible models to support gates, RAMs, and higher-level macros.

. State-dependent dynamic and static power for superior accuracy (10% of SPICE).

APAS takes the guess work out of power analysis. It is the industry's first power analysis tool that combines accuracy with easy of use. It is also the first to support the three leading ASIC simulators: QuickSim II, QuickHDL, and Verilog-XL, all with a common library.

APAS "snaps on" to the designer's chosen simulator. This allows the designer to use the same environments for power simulation as is used for functional and timing simulation. Since the host simulator uses the same netlist, vectors, models, and back-annotated parasitic, the same behavior is guaranteed. The familiar simulation environment also makes power analysis easy to learn and use. APAS is also easy to learn that it can be installed in the morning and used productively in the afternoon.

APAS is an interactive power analysis tool that helps the designer reduce power consumption. In addition to reporting how much power is being consumed, it helps the designer determine where and when. APAS's hierarchical capabilities can be used to find out which blocks in the design are consuming the most power. The designer can then focus on reducing power in these blocks. Analyzing the dynamic behavior of the circuit is also key to power reduction. APAS can dynamically trace the power consumption of any block in the design, giving the user a graphical representation of the periods of high and low activity. Moreover, the real strength of interactivity comes from combining these two capabilities. By stopping the simulator at a time of high activity and then listing the blocks consuming the most power at that moment, the designer can identify candidate blocks for power down. Powering down blocks can often result in dramatic power savings, but the savings need to be verified. APAS's interactive simulation environment allows the designer to perform "what if" experiments to verify power reduction.

Not only are the diagnostics important, but the designer must have confidence in the accuracy of the results. The key to accuracy is the power models. The power models must represent the ways that current flows in the design. This includes dynamic switching current, dynamic short-circuit current and static current. Achieving optimum accuracy requires models which account for input slew, load and internal state. APAS uses power models written in the Power Modeling Format (PMF). PMF has the robustness to handle state-dependent dynamic and static power. Its combination of equations and interpolated tables provide the means to capture the most detailed cell characterization data under varying slew, load and state conditions. PMF not only supports gates but it can also handle I/O cells, RAMs, and higher-level macros. These higher-level models become increasing important as the amount of Intellectual Property (IP) used in designs continues to grow.

Closing market windows are putting tremendous pressure on design throughput. How a tool fits in the design flow is critical. The ability to perform power analysis in the chosen environment streamlines the iteration cycle because no additional steps are required in the design flow. Simply turn APAS on to measure power during the usual functional or timing simulations, making power analysis part of the simulation flow - not an afterthought.

Designers need to meet market demands by increasing functionality while reducing system costs. APAS snaps on to the designer's chosen simulator to reduce design iteration and improve productivity. APAS's unique interactive power diagnostics enable designers to meet these challenges. APAS also delivers the required accuracy for both dynamic and static power consumption.

FUTURE WORK

Early in the design verification, especially when power models are not yet available, designers want a first look at the power consumption in their design. A fast power estimation algorithm is required to be added to APAS. This fast estimation is based only on the load switching power. The load capacitance can be estimated from cell fanout or from back annotation. Other extensions to APAS includes a mechanism or backplane that can connect power estimation algorithms from lower-level of abstraction (switch-level power estimation) to higher-level of abstraction (RTL power estimation), and make APAS a multilevel power estimation system.

REFERENCES

[1] E. Melancon, M. Rosneck, “Low-Power IC Design”, Communication Systems Design, pp. 30-34, December 1995.

[2] L. Maliniak, “IC Analysis Tools Help Manage Power”, Electronic Design, pp. 71-80, January 1996.

[3] L. Cooper, “IC Power: a Hot Problem”, Integrated System Design, pp. 63, July 1996.

[4] B. Nadel, “The Green Machine”, PC Magazine, Vol.12, no. 10, pp.110, May 1993.

[5] W. Roethig, “Full Chip Gate Level Power Analysis Requirementes”, LSI Logic Power Methodology Group, Rev. 5, May 1996.

[6] H.J.M. Veendrick, “Short-circuit Dissipation of Static CMOS Circuitry and Its Impact on The Design of Buffer Circuits”, IEEE Journal of Solid-State Circuits, Vol. SC-19, pp. 468-473, August 1984.

[7] R. Burch, F. Najm, P. Yang, I. Hajj, “Probabilistic Simulation for reliability analysis of CMOS VLSI circuits”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 9(4), pp. 439-450, April 1990.

[8] B.J. George, D. Gossain, S.C. Tyler, M.G. Wloka, G.K.H Yeap, “Power analysis and characterization for semi-custom design”, In Proceedings of the 1994 International Workshop on Low-Power Design, pp. 215-218, April 1994.

[9] L. Benini, M. Favalli, B. Ricco, “Analysis of hazard contribution to power dissipation in CMOS IC’s”, In Proceedings of the 1994 International Workshop on Low-Power Design, pp. 27-32, April 1994.

[10] R. Burch, F. Najm, P. Yang, D. Hocevar, “Pattern independent current estimation for reliability analysis of CMOS circuits”, In Proceedings of the 25th Design Automation Conference, pp. 294-299, June 1988.

[11] A. Ghosh, S. Devadas, K. Keutzer, J. White, “Estimate of average switching activity in combinational and sequential circuits”, In Proceddings of the 29th Design Automation Conference, pp. 253-259, June 1992.

[12] S.M King, “Accurate simulation of power dissipation in VLSI circuits”, IEEE Journal of Solid State Circuits, 21(5), pp. 889-891, October 1986.

[13] S. Rajgopal, G. Mehta, "Experiences with simulation-based schematic level current estimation", In Proceedings of the international Workshop on Low Power Design", pp. 9-14, April 1994.

[14] A. Tyagi, "Hercules: A power analyzer of MOS VLSI circuits", In Proceedings of the IEEE International Conference on Computer Aided Design, pp. 530-533, November 1987.

[15] A. Chandrakasan, R. Brodersen, "Low Power Digital CMOS Design", Kluwer Academic Publishers, pp. 56-95, 259-306, 1995.

[16] J. Rabaey, M. Pedram, "Low Power Design Methodologies", Kluwer Academic Publishers, pp.1-15, 21-33, 129-156, 1996.

[17] A. Bellaouar, M. Elmasry, "Low Power Digital VLSI Design Circuits and Systems", Kluwer Academic Publishers, pp. 1-10, 115-251, 510-522, 1995.

[18] F. Najm, "A Survey of Power Estimation Techniques in VLSI Circuits", IEEE Transactions on VLSI Systems, Vol. 2, No 4, pp.446-454, December 1994

[19] G. Frenkil, "Power Dissipation of CMOS ASICS", VLSI Technology Inc. 1991.

[20] F. Dresig, P. Lanches, O. Rettig, U. Baitinger, "Simulation and reduction of CMOS power dissipation at logic level", European Design Automation Conference, pp. 341-346, 1993.

[21] M. Cirit, "Estimating dynamic power consumption of CMOS circuits", IEEE Int. Conf. Computer Aided Design, pp. 534-537, November 1987.

[22] C. Tsui, M. Pedram, A. Despain, "Efficient estimation of dynamic power consumption under a real delay model", IEEE Int. Conf. Computer Aided Design, pp. 224-228, November 1993.

[23] F. Najm, "Transition density: a new measure of activity in digital circuits", IEEE Trans. Computer Aided Design, Vol. 12, No 2, pp. 310-323, February 1993.

[24] C. Huizer, "Power dissipation analysis of CMOS VLSI circuits by means of switch-level simulation", in IEEE Europ. Solid State Circ. Conf. pp. 61-64, 1990.

[25] R. Burch, F. Najm, P. Yang, T. Trick, "McPOWER: A Monte Carlo approach to power estimation", IEEE/ACM Int. Conf. Computer Aided Design, pp. 90-97, November 1992.

[26] J. Lipman, "EDA tools let you track and control CMOS power dissipation", EDN

Design Feature, pp. 65 75, November 1995.

[27] ---, “Power Analysis and Optimization”, TI ASIC CAD Flow, Internal product requirement survey, November 1993.

[28] Harish Sarin, Andrew McNelly, “A Power Modelling and Characterization Method For Logic Simulation”, IEEE CICC, pp363-366, 1995.

[29] B. George, G. Yeap, M. Wloka, S. Tyler, D. Gossain, “Power Analysis For Semi-custom Design”, Proc. CICC-94, pp. 249-252, 1994.

[30] M.A. Cirit, “Characterizing a VLSI standard cell library”, Proc. CICC, pp.25.7.1, 1991.

[31] M. N. Misheloff, “Improved Modeling and Characterization System for Logic Simulation”, Proc. IEEE Asic Intl. Conference, 1992.

APPENDIX A

POWER MODEL FORMAT

We start with an overview of the model format. Details and syntax of each section follows.

In the examples of this chapter, compiler keywords will appear in capital letters and user defined names will be lower case. However, the compiler is not case sensitive.

Model Data and Expressions:

- Signal Declarations
- Default Data
- Defined Constants
- Table Data
- Equations
- Internal Signal Definitions

Model Evaluation:

- Transition Energy Statements
- Static Power Statements

Figure 22. Power Model Format

Comments

Comment text can be put in models using either C comment style, /* <lines of text> */:

```
/*
This is a C style comment
*/
```

Or, comments can be proceeded by a double slash (as with C++) where a carriage return terminates the comment:

```
    // This is also a comment
```

Signal Definitions / Declarations

Cell Signal Declarations

Any signals used in the power model that exist as “pins” or “ports” on the cell interface need to be declared prior to appearing in an equation or table. This simplifies interface validation and indicates pin direction to the model compiler. Signals are declared as inputs, outputs or bidirectional. This is done with the keyword SIGNAL followed by INPUT, OUTPUT or BIDIR. It is not necessary to declare signals on the functional model that are not used in the power model.

Wide pins (buses) must include their range so that their width is determined. Once the busses are declared (or defined), if only the bus name appears in an expression, then the operation applies to all of its bits. Alternatively a specific bit can be referenced, e.g. ‘DATA[0]’.

Examples:

```
SIGNAL INPUT bist, rd, cs, ada[0:31], adb[0:31];
SIGNAL OUTPUT data[0:8];
SIGNAL BIDIR datb[0:8];
```

Grouping Signals

For convenience, a signal may be defined in the power model as a group of pins. To create a group use the following syntax:

```
SIGNAL GROUP buspins : ad1, ad2, ad3, ad4;
```

Boolean signals may be included as a contributing signal to a group. Internal states, defined below, may also be included. Group signals get their width implicitly from their definition. All contributing signals in the group must be declared previously in the model.

The state of the group is always the combined current state of their contributors. This differs from an internal state (see below) which is initialized at model construction then updated only by the power model following evaluation.

Defined Internal State

The definition of 'internal' signals within the 'black-box' cell boundary may be needed, to accurately represent the state and operation of a cell for a given power consumption data point.

The internal state may be a single bit or a bus, it inherits the width of the signal used to initialize it. An internal signal may used in any GROUP defined after the internal state definition.

Internal states are explicitly updated during model evaluation. An internal signal must be initialized in its definition with

a port signal, a GROUP, or an explicit state. The initial signal must be defined prior to the internal state definition.

Example:

This example shows an internal state used and updated in a model. First, define internal state “lastd” and initialize it with signal ‘d’:

```
SIGNAL INTERNAL lastd <= d;
```

In the transition energy statements (described in detail below), the statement shown will be active if the enable pin “en” goes low-to-high and ”lastd” is a logical ‘1’ when evaluation occurs. The state of “lastd” is updated with the current value of ‘d’ AFTER it is evaluated in the statement:

```
DYNAMIC {
      en(/) & lastd : energy_eqn1($PAD(CAPLOAD)), lastd <= d;
};
```

Boolean Signals

There may be some cases where it is necessary to compare signals or buses to determine the state or mode of a cell. A binary boolean expression is available for this purpose. It tests for equivalence, “==”, (for non-equivalence use “!=”) of two evaluated expressions. The result may be used like a single bit binary signal. The return value will be a logical ‘1’ if the equivalence tests true and zero ‘0’ otherwise.

The arguments to the boolean operators may be two signals or two arithmetic expressions. (A mix of signals and expressions are not supported.)

The boolean “signal” can be used in a transition statement (see below) as a condition, to impose state dependency. A boolean may also be a “signal” in a GROUP.

Example:

The mode of a dual port RAM cell is dependent on comparing two address buses. The following expression will return non-zero if the condition is true and zero if it is not.

An arithmetic expression:

```
SIGNAL BOOL bus_comp { $DIFFBITS(A1,A2) == 0 };
```

A direct signal compare:

```
SIGNAL BOOL bus_comp { A1 == A2};
```

Next the “bus_comp” and the control lines of a RAM are combined as a GROUP:

```
SIGNAL GROUP mode : bus_comp, rd, wr, ce;
```

The resulting group may be used to test for “read through” mode using the $STATE system function (defined below). The mode is defined as ‘1111’, the state where the address buses match, both read, “rd”, and write, “wr”, control line are asserted, and the cell is enabled:

```
$STATE( mode, ‘1111’)
```

System Functions

The system functions defined below will be used in the energy expressions associated with the transitions and state of the cell.

Data Access Function

The $PAD system function is used to access instance specific data from the client application. For example, the data will usually come from a Power Analysis Data (PAD) file. Default values may be set in the model format or design wide in the library file.

Syntax:

"$PAD(" <data type> [, <port name>] ")"

Where:

<data type> -- is one of a set of data elements found in the PAD file. These elements are defined in the syntax of the PAD file. Data in the PAD file will be either design wide - VOLTAGE and TEMPERATURE, or instance and port specific -SLEW and CAPLOAD.

<port name> -- (optional) Used for instance/port specific data to indicate which port's data to access. (The instance context is set in the PAD file.) This field is optional, the active pin is used as a default. (Dynamic pin equations are evaluated in the context of an instance and active pin transition, more on this later.)

Examples:

$PAD(VOLTAGE) -- Get voltage.
$PAD(CAPLOAD) -- Get load capacitance of active pin.
$PAD(CAPLOAD, QBAR) -- Get load capacitance of pin "QBAR".
$PAD(SLEW) -- Get the slew of the active pin.
$PAD(SLEW, IN1) -- Get the slew of pin IN1.

Special case for SLEW type data:

SLEW statements in the PAD file have positions for both rising and falling data. If the context of the equation specifies a specific transition of an active pin, ('/' for rising or '\'

for falling) then the data for the active transition will be used. If no transition is specified the average of the rising and falling slews will be used.

Units:
$PAD returns a real number in primary units, as follows:

VOLTAGE (Volts)
TEMPERATURE (Degrees Celsius)
SLEW (seconds)
CAPLOAD (farads)

State Check Function

The $STATE function compares a signal to an explicit state and returns "true" (some non-zero value) if they do not match and "false" (0) if they do not.

Syntax:
"$STATE(" <sig> "," <explicit state> ")"

The signal, <sig>, may be a bus (wide), a GROUP (see above), or single pin. The <explicit state> is a string of bit states, '0,' '1' or '-' (dash is "don't care") that must be the same width as the signal. The two arguments are compared bitwise, if any bits in the current state of <sig> do not match the <explicit state> ("don't care" always matches) a value of zero "0" is returned.

Signal Comparison Function

The $DIFFBITS function does a bitwise comparison and returns the number of bits that differ between two signals as an integer. If a bit is in an unknown state (X) it will always be counted as different, unless matched with a "don't care" in an explicit state. The arguments are optional and supply the signals to be compared.

Syntax:

```
"$DIFFBITS(" [<sig1>] [, <sig2>] ")"
```

The signals, <sig1> and <sig2>, may be buses (wide) or single pins.

Examples:

When given a single argument, the signal, DATA, is compared with its own previous state. This allows the current and previous state of a bus to be compared to determine how many bit changed.

```
$DIFFBITS(DATA)
```

When used with no arguments the function must be used in the context of a transition statement (defined below). The bus is assumed to be the active pin and is compared with it's previous value.

```
$DIFFBITS()
```

It is possible to use an explicit state (in quotes) as the second argument. A dash ('-') in the explicit state means "don't care" and will always count as a match, even with a bit in the X state:

```
$DIFFBITS( DATA, '010101--' )
```

When given two arguments the function simply compares the two bus pins (or internal states) specified.

```
$DIFFBITS(ADA, ADB)
```

If the arguments are not the same width, then the smallest bus will be extended with the value of its MSB (the left-most bit). Example:

```
$DIFFBITS (enables, '0')
```

The function call above may be used to determine the number of active (high) read or write enable pins on the RAM, since the bits in the state '1' will differ from the "all zeros" argument '0'.

Bus Width Function

This $WIDTH function will return the width of a bus given as the argument as an integer. When used with no argument the width of the active pin is returned.

```
$WIDTH(<bus>)
```

Where <bus> is a declared port signal or defined internal state or group.

Natural Log

The $LN function returns the natural log of the argument.

```
$LN(<operand>)
```

Where operand is a real number, constant or a reference to an expression that has been previously defined.

Model Data and Expressions

Constants

Constant variable can be defined for use in generic equations. This simplifies incorporating characterized data into expressions.

Examples:

```
CONST int_cap 4.53;
CONST nano 1e-9;
```

Default Data

Equations that use $PAD system functions to access data expect design specific data to be annotated into the power instances. The DEFAULT statement sets a model-wide default for these data elements.

Example:

```
DEFAULT VOLTAGE 5.0;
DEFAULT CAPLOAD 0.02(nano);
```

The conversion operator “(nano)” is used to normalize data values and is discussed further in the “Tables” section below.

Tables

One, two or three dimensional table can be used to enter piece-wise linear data. They are designed to allow real numbers in their subscripts.

Conversion Operators:

For readability, the indices "cap" and "slew" in the example below, are given conversion operators, which follow them in the table definition. The table's data also has a conversion operator, shown following the table indices: "(1e-14)". These are optional but allow the table to be more compressed and readable.

When used, the conversion operator scales its associated data by the operator's value. This is useful since data received through the $PAD system function is always in primary units, and the energy or power quantities returned must also be in primary units (joules and watts).

The syntax is simply a number or defined CONST enclosed in parenthesis.

Table Example:

A three dimensional table.

```
TABLE dim3 (
      cap (1e-12) : 0.05, 0.1, 0.15, 0.2;
      slew (nano) : 1.2, 2.3, 3.7;
      voltage : 2.5, 3.5, 4.5;
) (1e-14) {
      {{0.1, 1.1, 2.1}, {0.2, 1.2, 2.2}, {0.3, 1.3, 2.3}},
      {{0.4, 1.4, 2.4}, {0.5, 1.5, 2.5}, {0.6, 1.6, 2.6}},
      {{0.7, 1.7, 2.7}, {0.8, 1.8, 2.8}, {0.9, 1.9, 2.9}},
      {{1.0, 2.0, 3.0}, {1.1, 2.1, 3.1}, {1.2, 2.2, 3.2}}
};
```

Nesting order of data elements is done from outside in (as in C language). The real number indices are mapped one-to-one with their corresponding array or data points. Linear interpolation (and extrapolation) is used to return a value when the real number indices are between the data points explicitly specified in the table.

Given the example and a table reference:

```
dim3[$PAD(CAPLOAD,out)][$PAD(SLEW)][3.0]
```

Assume the load capacitance is 0.15pF and the slew is 1.2ns. The conversion operators of "cap" and "slew" anticipates that the index values passed in from the $PAD system function, or any other source, such as an equation, will be in primary units, farads and seconds, and convert the index values accordingly. Note that slew uses the defined constant "nano" that we defined in the CONST example above.

A resulting 0.15 value for "cap" would indicate the third row. "Slew" has value 1.2 so the first set of three values will be used from that row and a "voltage" value of 3.0 would require interpolation between the first (0.7) and second (1.7) data point. A value of 1.2 is found in the table and 1.2e-14 is returned due to the data conversion operator: "(1e-14)".

Table Data Tags

It may be desirable to include several sets of characterized data into a single table when the table's dimensions are the same for all sets of data. The use of "data tags" will allow this as shown below. A one dimensional table using data tags:

```
TABLE dim1 (
      slew (nano): 0.5, 1.0, 1.5, 2.0;
) (1e-12) {
rise : {0.1, 1.1, 2.1, 3.1};
fall : {0.2, 1.2, 2.2, 3.3};
};
```

To access the desired data when referencing a table, follow the table reference with a dot '.' and the tag. Example of a table reference: to the table above, "dim1", we want to pass in a pin's slew and get data for a falling transition:

```
dim1.fall[$PAD(SLEW)]
```

Expressions

Equations

Energy dissipations may be modeled using mathematical equations. An equation (EQN) returns a real number.

Operators:

'+'	: add
'-'	: subtract
'*'	: multiply
'/'	: divide
'**'	: exponent

Operands:

Equation operands may be real numbers, constants, passed in arguments, system functions, or references to tables or other equations.

Tables and equations referenced in an equation definition must be defined in the model prior to being referenced. Forward declarations are not supported.

Example:
To define switching energy for a typical output pin transition:

$$Energy\,(trans) = \frac{1}{2} C \times V^2$$

In model syntax:

```
EQN out_erg() { 0.5 * $PAD(CAPLOAD) * ($PAD(VOLTAGE) ** 2) };
```

Alternatively the modeler may want to pass in the variables using an argument list:

```
EQN out_erg(cap, volts) { 0.5 * cap * (volts ** 2) };
```

Model Evaluation Section

This model section must follow data and expression definitions. It is where transition energy and static power dissipation is specified based on a state change on the perimeter of the cell.

Transition Statements

A transition statement is used to associate a specific pin transition with a quantity of energy, or update the internal state(s) of the model, or both.

A transition statement should exist in the model for each pin transition that is known to produce energy dissipation. The quantity of energy is specified by an energy expression. The energy expression can be either a constant, a reference to a defined expression, or a table reference.

Internal states can be updated in transition statements following an energy expression, if one exists.

State Dependence

If the amount of energy dissipated depends on the (static) state of other signals, an entry for unique transition/state combinations may be included in the model. To specify a state dependence, follow the active pin with syntax: "& <expression>"; where the expression is a boolean signal, a $STATE system function, or a single bit signal. The result of the expression can be inverted by preceding it with a '~' (tilde) character. A non-zero result of expression evaluation will satisfy state dependence.

It is important to note that output pin changes, caused by an input pin change, will not appear at the output until the propagation delay time elapses. So, power models may need to use internal state to help determine the effect an input change will have on output state to accurately determine internal energy dissipation.

Specifying Transitions

Statements may, if necessary, explicitly specify the direction of the transition by following the active signal with "(/)", for rising, or "(\)" for falling. A "(*)" indicates "any change" and is the default. Transitions will NOT be allowed if the subject signal is a bus (i.e. more than one bit wide). Transitions to and from X and Z are handled implicitly by the evaluator, (more on this below).

Examples:

For any change on out driver "out":

```
        out           : out_erg($PAD(CAPLOAD)) ;
```

For any change on input "in" IF en is true (a 1 state):

```
        in(*) & en    : intbl.en[$PAD(CAPLOAD, out)][$PAD(SLEW)];
```

For falling input "in" IF en is 0. Note that internal state "last_st" is also updated:

```
        in(\) & ~en  : intbl.off[$PAD(CAPLOAD, out)][$PAD(SLEW)], last_st <=
    in;
```

For a rising transition on pin "en" IF signals "in" and "last_st" are the same. First, define a BOOL signal, then reference it in the transition condition.

```
    SIGNAL BOOL same_state { $DIFFBITS(in, last_st) == 0 };
    en(/) & same_state : entbl.rise.same[$PAD(CAPLOAD)][$PAD(SLEW)];
```

Statement Organization: Blocks

Transition energy statements are organized in "blocks" starting with the keyword DYNAMIC and enclosed in braces.

Example:

```
DYNAMIC {
        in(*) & en    :intbl.en[$PAD(CAPLOAD, out)][$PAD(SLEW)];
        in(*) & ~en  :intbl.off[$PAD(CAPLOAD, out)][$PAD(SLEW)], last_st <= in;
};
```

Block Evaluation:

When more that one transition and condition is "true" within a block, only the first transition statement, by order of appearance, will be evaluated. This may be used to avoid double counting internal energy dissipation. (e.g. All inputs of a 3 input and gate going 0 to 1 at the same time.)

Blocks allow better control over the situation when more that one pin transition occurs. It is also useful with complex state dependence as a "catch-all" row may be used at the end of a block after the transitions with specific state dependencies.

Example:

```
SIGNAL GROUP ins : i0, i1, i2;

DYNAMIC{ // rising input pin causing output change
      i0(/) & outrises(ins) : in_erg_ri();
      i1(/) & outrises(ins) : in_erg_ri();
      i2(/) & outrises(ins) : in_erg_ri();
      ins : in_erg_noop();
};
```

Modelers can use as many blocks as necessary to describe the power model. Each block will be evaluated separately. Internal state updates may be more conveniently handled in separate blocks but this is not required.

Wide Pins and Buses

When a wide pin or bus is used as the active signal the following rules apply:

- Specific transitions may not be specified.
- The associated equation is evaluated only once regardless of the number of bits changing.
- An alternative to whole-bus statements is to specify each bit (e.g.: BUS[0]) so that transitions may be used.

Modelers may use system functions, such as $DIFFBITS and $WIDTH, to scale energy consumption for the case of multiple bits changing.

Example:

```
EQN addr_trans(bus) { $DIFFBITS(bus) * column_erg[$PAD(SLEW)] };
DYNAMIC {
        addr & cs    :addr_trans(addr);
};
```

Static Power Statements

Static power entries should be added for any states for which static power has been characterized. Static power may be state dependent using the same expression syntax described in transition energy statements above. Static power expressions will contribute to total average power by a ratio of the time they were in the specified state to total analysis time. Static power statements are organized in blocks titled STATIC.

Examples:
Static power for this model is dependent on the state of the "en" pin.

```
STATIC {
        en             : st_pwr_en1();
        ~en            : st_pwr_en0();
};
```

Static power characterized for variable voltage, not state dependent.

```
STATIC { : leakage[$PAD(VOLTAGE)] };
```

Units

The units of the data for energy expressions for dynamic power should be returned in joules (J). Static power expressions should be returned in watts (W).

State Abstraction

This power model format will be based on a standard four state abstraction. The following symbol definitions will be used in the model syntax:

‘1’	-- A logical high state (Vdd)
‘0’	-- A logical low state (Ground)
‘Z’	-- High impedance state (Hi-Z)
‘-’	-- A “don’t care”, used when defining state dependence
‘/’	-- A rising transition
‘\’	-- A falling transition
‘*’	-- Either a rising or falling transition

A reasonable mapping from the state abstraction of the host simulator to the Power Model’s four state abstraction will be created for each host.

The Unknown “X” State

Unknown states occur in digital simulation in several situations:

- Initialization (power up)
- Multi-driver node resolution (bus contention).
- Uncertainty during transition (such memory data bus transitions)
- Partial transitions (spikes glitches)
-

Note that it is not possible to explicitly specify transitions to or from an unknown “X” state. The model evaluator will automatically handle these events based in the energy specifications for “good state” events. Transition from 0 to X or from X to 1 will assess 50% of the energy specified for a 0 to 1 (a “/” symbol) transitions. X to 0 and 1 to X will assess 50% of 1 to 0 (‘\’) energy.

Users will be encouraged to avoid power analysis during initialization periods when the state of the circuit is unknown. In the other cases the assessment of 50% of a full transition has shown to work well in most cases when X is encountered. Consider the 0>X>1 case in which two 50% transitions will give the same result as a single 0>1 transition that certainly has occurred. Only if the real silicon undergoes multiple partial transitions in the X region is inaccuracy introduced.

Partial transitions, 0>X>0 or 1>X>1, which usually indicate glitches will assess half of the power of a full pulse.

The High Impedance “Z” State

The model evaluator will also implicitly handle tri-state transitions. It will be assumed that Hi-Z drivers are attached to multiply driven (bus) nets. The analysis tool will need to handle situations where bus contention causes anomalies such as a short circuit caused by two or more drivers driving opposing states.

Assuming no bus contention, we can assume that the act of “turning off” - that is, going to a Z state - does not dissipate any of the energy stored on the net (or internal node). For this reason transitions to Z will be ignored on both inputs and outputs.

When a driver transitions from a “Z” to some logic state, the energy needed to “drive” the net depends on the current state of the net. If the “turning on” of the driver causes the net to change state then the amount of energy associated with a transition should be assessed. But, if the net does not change state, no current will flow and the transition should be ignored.

If it is possible to get the net state from the design interface of the host simulator, the scenario above can be modeled. If the net state is not available, we will assume that there is a 50% chance that the node will change state when a driver turns on, and 50% of the transition energy will be assessed.

Inputs will assess internal energy based on the last valid driven state and the current driven state, with intermittent Hi-Z state ignored.

Model Syntax

- Keywords are shown in capital letters. They will be parsed case insensitive.
- Syntax punctuation is in single quotes.
- Element may be grouped in brackets '{' '}' to clarify definition
- Elements in braces -- [] -- are optional.
- A star -- '*' -- means "zero or more" of the preceding bracketed or braced element.
- A plus sign '+' means one or more.
- Elements separated by bars (i.e. A|B|C) mean "one of A or B or C".
- **name** -- can be any string of alphanumeric characters, or '_', starting with a non-numeric.
- **real** -- is a floating point number, may use exponential format (ex: 3.456e-2).
- **posint** -- a positive integer (or zero)

Note: Text preceded by double slash -- "//" -- are comments, not part of syntax.

Model Syntax:

<model> ::= <model name> <defs_and_data>* <eval_model>

<model name> ::= MODELNAME **name** ';'

<defs_and_data>::= <signal_def>
      |= <expression_def>

Signal Definition Syntax:

```
<signal_def>::= SIGNAL <pin_dir> <sig_name> [ ‘,’ <sig_name> ]* ‘;’
      ||= SIGNAL GROUP <grp_name> ‘:‘ <grp_list> ‘ ;’
      ||= SIGNAL INTERNAL <int_name> ‘<=’ <int_assign> ‘;’
      ||= SIGNAL BOOL <sig_name> ‘{’<test_expression>‘}’ ‘;’

<pin_dir>::= INPUT
      ||= OUTPUT
      ||= BIDIR

<grp_list>::= <gen_signal> [ ‘,’ <gen_signal> ]*

<gen_signal>::= <pin_name>
      ||= <int_name>
      ||= <bool_ref>// Reference to a boolean signal
      ||= <state_dep>// $STATE system function

<sig_name>::= <pin_name>
      ||= <bus_name> <range>

<grp_name> ::= name // Name of state to be used in model table.

<int_name>::= name // Name of state to be used in model table. Cannot conflict
                   // with pin name.

<int_assign>::= <pin_name>
      ||= <bus_name>

<range> ::= ‘[‘ posint ‘:’ posint ‘]’

<pin name> ::= name // Name of pin found on instance of model.
      ||= name <bit>// bus name followed by a bit specification

<bit>::= ‘[‘ posint ‘]’

<bus_name>::= name // Name of bus found on instance of model, full
                   // range implied

<test_expression> ::= <simple_expr> <equality_op> <simple_expr>
      ||= <sig_ref> <equality_op> <sig_ref>

<equality_op> ::= ‘==’// expressions are equal
      ||= ‘!=’        // expressions not equal

<bool_ref>::= <sig_name>
```

Energy Expressions:

```
<expression_def> ::= <constant>
      ||= <default_data>
      ||= <data_table>
      ||= <equation_def>

<constant> ::= CONST <const_name> <value>';'

<const name> ::= name // to be substituted for value in equation.

<default_data> ::= DEFAULT <data_type> <value> [<conversion>]';'

<data_type>::= VOLTAGE// data elements of PAD file
      ||= TEMPERATURE
      ||= CAPLOAD
      ||= SLEW

<value> ::= real

<conversion>     ::= '(' <const name> ')'
                 ||= '(' real ')'
```

Table Syntax:

```
<data_table> ::= TABLE <table_name> '(' <tab_var_def>+ ')'
      [<conversion>] '{' <table_data> '}' ';'

<tab_var_def> ::= <var_name> [<conversion>]':' <data list> ';'
      / one for each table dimension

<table_data>::= <labeled_array>
      ||= <nested_array>
      ||= <data_list>

<labeled_array>::= <data_tag> ':' <nested_array> ';'

<nested_array> ::= '{' <nested_array> [',' <nested_array> ]+ '}'
      ||= '{' <data_list> '}'

<data_list> ::= real [',' real ]*

<table_name> ::= name // the name by which the table is called in an
expression.

<var_name> ::= name // the name given to a range variable. Expected to be
passed            // as a real number.

<data_tag> ::= name // To store data from several characterizations in a single
                  // table, the optional "data tag" can be used. It is required if
                  // more than one set of TABLE DATA is present.
```

<table_ref> ::= <table_name> { '[' <table_index> ']' }+
// as many as 3 dimensions allowed. Number must match definition.
||= <table name>'.'<data_tag> { '[' <table_index> ']' }+
// use data tags if used in definition

<table_index> ::= **real**
||= <constant>
||= <expr_ref>
||= <system_function>

Equation Syntax:

<equation_def> ::= EQN <expr_name> '(' [<args>] ')' '{' <math_expr> '}' ';'

<expr_name> ::= **name** // used to call expression evaluator.

<args> ::= **name** [, **name**]* // A list of passed-in variables. Expected to be a
// real number valued variable.

<math_expr> ::= <simple_expr>
||= '(' <math_expr> ')'
||= <math_expr> <binary_op> <math_expr>

<binary op> ::= '+'
||= '-'
||= '*'
||= '/'
||= '**'   // exponential

<simple_expr> ::= <operand>
||= '(' <simple_expr> ')'

<operand> ::= **real**
||= **name**   // name of passed arg
||= <constant>
||= <expr_ref>
||= <table_ref>
||= <system_function>

<expr_ref> ::= <expr_name> '(' [<expr_args>] ')'

<expr_args> ::= <table_index>
||= <sig_name>

```
<system function> ::= $PAD '(' <data_type> [',' <pin_name> ] ')'

        ||= $DIFFBITS '(' [ <diff_args> ] ')'
        ||= $WIDTH '(' [ <bus_name> ] ')'// name of bus
        ||= $LN '(' <math_expr> ')'

<state_dep>::= $STATE '(' <sig_ref> ',' <state_vec> ')'

<diff_args>::= <sig_ref> ',' <sig_ref>
        ||= <sig_ref> ',' <state_vec>
        ||= <sig_ref>// compare pin with it previous state
                        // (if no pin name use active pin)

<sig_ref>::= <pin_name>
        ||= <bus_name>
        ||= <grp_name>
        ||= <int_name>

<state_vec>::= ''' [ '0' | '1' | '-' ]+ ''' // number of characters relates to number
and
                        // order of pins in pin list of state's definition. '-'
                        // means "don't care". String in single quotes.
```

Model Evaluation Syntax:

```
<model_stmt> ::= <dyn_blk>
                        ||= <static_blk>

<dyn_blk>::= DYNAMIC '{' dyn_stmt> '}' ';'

<dyn_stmt> ::= <trans_sig> ['&' <st_dep>] ':' <energy_ref> [state_update]';'

<trans_sig>::= <pin name> [<trans>]
        ||= <bus_name>
        ||= <grp_name>

<trans>::= '(' '\' ')'// a falling transition
        ||= '(' '/' ')'   // a rising transition
        ||= '(' '*' ')'   // any transition (this is default if none given).

<st_dep>::= ['~'] <gen_signal > // reference to a defined state, must be 1 bit
                        // '~' inverts signal

<state_update>::= int_name '<=' <int_assign> [',' int_name '<=' <int_assign>
]*

<static_blk>::= STATIC '{' <static_stmt> '}' ';'
```

```
<static stmt>::= <st_dep> ':' <energy_ref> ';'
      ||= ':' <energy_ref> ';' // no signal indicates "all states"

<energy_ref>::= <expr_ref>
      ||= <table_ref>
      ||= real
      ||= <constant>
```

Power Model Examples

A Three-input AND

```
MODELNAME and3;

CONST ns 1.0e-9;
CONST pF1.0e-12;

// declare signals

SIGNAL INPUT i0, i1, i2;
SIGNAL OUTPUT z;

// define a specific cell state based on input pin states

SIGNAL GROUP ins : i0, i1, i2;

// A 3D piece-wise linear table for short circuit power
// (Data from characterization with variable load capacitance, input slew and
voltage)
```

```
TABLE dim3 (
    cap (pF): 0.5, 1.0, 1.5, 2.0;
    slew (ns): 1, 2, 3;
    volts : 2.5, 3.3, 5.0;
)(1e-13){
    in_rise :
    {{{0.1,1.1,2.1}, {0.2,1.2,2.2}, {0.3,1.3,2.3}},
    {{0.4,1.4,2.4}, {0.5,1.5,2.5},{0.6,1.6,2.6}},
    {{0.7,1.7,2.7}, {0.8,1.8,2.8},{0.9,1.9,2.9}},
    {{1.0,2.0,3.0}, {1.1,2.1,3.1},{1.2,2.2,3.2}}};

    in_fall :
    {{{0.1,1.1,2.1}, {0.2,1.2,2.2}, {0.3,1.3,2.3}},
    {{0.4,1.4,2.4}, {0.5,1.5,2.5},{0.6,1.6,2.6}},
    {{0.7,1.7,2.7}, {0.8,1.8,2.8},{0.9,1.9,2.9}},
    {{1.0,2.0,3.0}, {1.1,2.1,3.1},{1.2,2.2,3.2}}};

};

    // A table for leakage power dissipation

TABLE leakage(
    volts : 2.5, 3.3, 5.0;
) (1e-15) { 1.1, 1.2, 1.3 };

    // set default data

DEFAULT CAPLOAD 0.03 (pF);
DEFAULT VOLTAGE 5;
DEFAULT SLEW 1.2 (ns);

    // a typical equation for switch power dissipation of a driver

EQN out_erg(cap, volts) { 0.5 * cap * (volts **2) };

    // local sub-expressions

EQN cload() { $PAD(CAPLOAD, z) + 0.013 };

    // a lookup for internal energy when inputs rise

EQN in_erg_ri() { dim3.in_rise[cload()][$PAD(VOLTAGE)][$PAD(SLEW)] };

    // a lookup for internal energy when the output goes to 0

EQN in_erg_fa() { dim3.in_fall[cload()][$PAD(VOLTAGE)][$PAD(SLEW)] };

// Transition/state statements
```

```
DYNAMIC {
      z              : out_erg($PAD(CAPLOAD), $PAD(VOLTAGE)) ;
};
DYNAMIC{ // rising input pin causing output change
      i0(/) & $STATE(ins, '-11') : in_erg_ri();
      i1(/) & $STATE(ins, '1-1') : in_erg_ri();
      i2(/) & $STATE(ins, '11-') : in_erg_ri();
      // falling signal resulting in output change.
      i0(\) & $STATE(ins, '-11') : in_erg_fa();
      i1(\) & $STATE(ins, '1-1') : in_erg_fa();
      i2(\) & $STATE(ins, '11-') : in_erg_fa();
};
// an expression for leakage current (no state dependency)
STATIC { : leakage[$PAD(VOLTAGE)]; };
```

A Tri-State Buffer

This table for a tri-state buffer shows how internal states are updated. Note that the internal state is updated when the enable falls and when in changes while enabled, thus capturing the last state driven. Also note that the output, input and enable are treated in separate blocks so that if more than one pin changes at once power will be assessed for all events. (This is done at the discretion of the modeler):

```
MODELNAME buf_3st;

CONST ns 1.0e-9;
CONST pF 1.0e-12;

SIGNAL INPUT a, en;
SIGNAL OUTPUT z;

// define an internal state to keep last driven state
SIGNAL INTERNAL last_st <= a;

     // a test for signal equivalence
SIGNAL BOOL same_st { $DIFFBITS(a, last_st) == 0 };

// data for static power
CONST st_en 0.0063;
CONST st_nen 0.0012;

TABLE intbl ( cap : 0.5, 1.0, 1.5, 2.0; ) {0.01, 1.01, 2.01, 3.01};

TABLE entbl (
     cap(pF) : 0.5, 1.0, 1.5;
     slew(ns) : 1, 2, 3;
) (1e-12) {
     rise_nc:   {{0.03,0.42,0.71}, {0.06,0.47,0.74}, {0.10,0.38,0.76}};
     rise_ch:   {{0.12,1.13,2.11}, {0.27,1.28,2.29}, {0.38,1.35,2.31}};
     fall:      {{0.01,0.11,0.21}, {0.02,0..21,0.22}, {0.03,0.13,0.23}};
};

     // equation for driver power dissipation
EQN out_erg(cap, volts) { 0.5 * cap * (volts **2) };

DYNAMIC { z : out_erg($PAD(CAPLOAD)) ; };
```

```
DYNAMIC{
      a(*) & en    : intbl[$PAD(CAPLOAD)], last_st <= a;
};
DYNAMIC{
en(/) & same_st  : entbl.rise_nc[$PAD(CAPLOAD, z)][$PAD(SLEW)];
en(/) & ~same_st : entbl.rise_ch[$PAD(CAPLOAD,z)][$PAD(SLEW)];
en(\)            : entbl.fall[$PAD(CAPLOAD, z)][$PAD(SLEW)],last_st<= a;
};
STATIC{
      ~en          :st_nen;
      en           :st_en;
};
```

A D-Flip-Flop

```
// This negative edge D-Flip-Flop model is characterized for a voltage of 4.5v.

// signal declarations

MODELNAME dff;

CONST ns 1.0e-9;
CONST pF 1.0e-12;
CONST pJ 1.0e-12;

SIGNAL INPUT ckn, d;
SIGNAL OUTPUT qn;
SIGNAL GROUP cur_st : d, qn;

// output switching power equation

EQN node_pwr() { 4.5 ** 2 * $PAD(CAPLOAD) * 1.0e-2 * 0.5 };

TABLE tab_clk_2dim(
      cload(pF) : 0, 0.25, 0.5, 1, 2.5, 5 ;
      slew(ns) : 0, 1.2, 2, 4, 6 ;
) (pJ) {
      fall_d0_q1:
      {{3.974e-2, 3.0994e-2, 3.3769e-2, 3.4613e-2, 3.5252e-2, 3.5081e-2},
       {3.9006e-2, 3.4302e-2, 3.4898e-2, 3.0213e-2, 2.9842e-2, 2.9796e-2},
       {3.4232e-2, 4.0576e-2, 4.39e-2, 4.1577e-2, 4.0579e-2, 4.0459e-2},
       {5.8654e-2, 4.0317e-2, 3.5113e-2, 3.3995e-2, 3.4068e-2, 3.3739e-2},
       {5.0407e-2, 4.2158e-2, 4.0427e-2, 3.4845e-2, 3.5392e-2, 3.4513e-2}};

      fall_d1_q0:
      {{4.8449e-2, 3.8375e-2, 3.003e-2, 3.0024e-2, 3.9709e-2, 4.1301e-2},
       {3.5001e-2, 3.0209e-2, 3.3085e-2, 3.4681e-2, 3.4235e-2, 3.3853e-2},
       {5.1888e-2, 4.2568e-2, 4.2027e-2, 4.4994e-2, 4.5027e-2, 4.5834e-2},
       {4.8066e-2, 4.2745e-2, 3.1542e-2, 3.147e-2, 3.3391e-2, 3.2414e-2},
       {3.3796e-2, 3.6753e-2, 3.1979e-2, 3.3546e-2, 3.4167e-2, 3.3534e-2}};
};
```

```
TABLE tab_clk_1dim(
      slew(ns) : 0, 1.2, 2, 4, 6 ;
) (pJ) {
      fall_d1_q1  :{1.6837e-2, 1.2341e-2, 1.7523e-2, 1.3552e-2, 1.4915e-2};
      rise_d1     : {1.9667e-4, 2.8740e-3, 6.3183e-4, 3.2267e-3, 3.5620e-3};
      fall_d0_q0  :{1.1194e-2, 1.7557e-2, 1.1486e-2, 1.2422e-2, 1.3525e-2};
      rise_d0     : {3.8123e-3, 2.1425e-3, 3.5092e-3, 3.5030e-3, 3.2410e-3};
};
TABLE tab_d(
      slew(ns) : 0, 1.2, 2, 4, 6 ;
) (pJ) {
      fall_ckn1:  {1.7184e-2, 1.3748e-2, 2.1493e-2, 1.4666e-2, 1.4828e-2};
      fall_ckn0:  {3.0177e-3, 3.1572e-3, 3.3195e-3, 3.3850e-3, 3.5388e-3};
      rise_ckn1:  {1.5459e-3, 1.5407e-3, 1.4681e-3, 1.6305e-3, 1.7655e-3};
      rise_ckn0:  {4.4968e-4, 3.8350e-4, 3.9492e-4, 3.7668e-4, 5.2090e-4};
};
EQN cload() { $PAD(CAPLOAD, qn) };
// evaluation statements
```

```
        // change on output
    DYNAMIC{
        q : node_pwr() ;
    };
        // clock change, data high, w/ various output states
    DYNAMIC{
    ckn(\) & $STATE(cur_st,'10') :
    tab_clk_2dim.fall_d0_q1[$PAD(SLEW)][cload()];
    ckn(\) & $STATE(cur_st, '11') : tab_clk_1dim.fall_d1_q1[$PAD(SLEW)] ;
    ckn(/) & d          : tab_clk_1dim.rise_d1[$PAD(SLEW)] ;
        // clock change, data low
    ckn(\) & $STATE(cur_st, '01') :
    tab_clk_2dim.fall_d0_q1[$PAD(SLEW)][cload()];
    ckn(\) & $STATE(cur_st, '00') : tab_clk_1dim.fall_d0_q0[$PAD(SLEW)] ;
    ckn(/) & ~d         : tab_clk_1dim.rise_d0[$PAD(SLEW)] ;
        // change on data pin
    d(\) & ckn          : tab_d.fall_ckn1[$PAD(SLEW)];
    d(\) & ~ckn         : tab_d.fall_ckn0[$PAD(SLEW)];
    d(/) & ckn          : tab_d.rise_ckn1[$PAD(SLEW)];
    d(/) & ~ckn         : tab_d.rise_ckn0[$PAD(SLEW)];
    };
```

A Multi-port RAM

This requires an internally defined boolean equation to determine whether two read address pins are accessing the same location. (See the "bus_comp" expression.) If the result is true then an offset to the read power can then be applied to the calculation for this case.

```
MODELNAME ram32;
CONST pF 1.0e-12;
SIGNAL INPUT AA[0:31], AB[0:31];
SIGNAL INPUT RWA, RWB; CS;
SIGNAL OUTPUT : QA[0:31], QB[0:31)];
SIGNAL INTERNAL lAr <= QA;        // Defines a state for the last word read
from the
SIGNAL BOOL bus_comp { AA == AB};
SIGNAL GROUP mode : bus_comp, RWA, RWB;
TABLE rdtab( //  A table for internal read power (data omitted)
        cap (pF) :  0.5, 1.0, 1.5;
        Nbits :  0, 4, 8, 12, 16;
)
        {{.......
        ..........}};
                        // 1/2CV2N
EQN node_pwr(nbits) { $pad(VOLTAGE) ** 2 * $pad(CAPLOAD) * 0.5 * nbits };
EQN cload() { ($pad(CAPLOAD, QA) + $pad(CAPLOAD, QB)) / 2 };
// note: some energy statements omitted
dynamic {
    // Power for Port A when Address A = Address B and both are READ
enabled
    QA & $state(mode, '111')  : 0.75 * rdtab [cload()][$diffbits(QA,lAr)]
                    + node_pwr($diffbits()), lAr <= QA;
                    // Power for Port A when Address A differs from Address B
    QA & $state(mode, '01-') : rdtab [cload()][$diffbits(QA,lAr)]
                    + node_pwr($diffbits()), lAr <= QA;
```

```
};
dynamic {
    // Power for Port B when Address A = Address B and both are READ
enabled
    QB & $state(mode, '111') : 0.75 * rdtab [cload()][$diffbits(QB,IBr)]
                    + node_pwr($diffbits()), IBr <= QB;

                    // Power for Port B when Address A differs from Address B
    QB & $state(mode, '0-1') : rdtab [cload()][$diffbits(QB,IBr)]
                    + node_pwr($diffbits()), IBr <= Q;
};
```

APPENDIX B

POWER ANALYSIS DATA FORMAT

PAD syntax

The syntax of the PAD file is identical to SDF with the following exceptions:

CAPSCALE is added to specify capacitive units. Unit can be 1, 10, or 100 of either pf (picofarads) or ff (femtofarads).

The POWERDATA section replaces the DELAY and TIMINGCHECK sections.

POWERDATA contains instance specific data for input slew in SLEW, and output load capacitance, given by CAPLOAD.

Min, Typ, Max Data fields

As with SDF, data can be generated in sets of three, called triplets. The field to be used is often determined by the process corner being simulated. The three fields are usually called, from left to right, "min", "typ" and "max" when referring to timing data. When annotating power data with APAS, the same mode should be used as is being used for timing data. The choice of min, typ, max is made for APAS as an argument to the annotate command.

Note: The terminology here may be confusing. The notion of "min" or "max" may imply "minimum power" and "maximum power", respectively. This is not necessarily the case. It is more likely to be meant to be consistent with timing min, typ, max values to be consistent with SDF. That is, “min” will indicate the conditions that minimizes input slew, which may, in fact, result in greater power consumption than “max” timing data.

Slew Data

SLEW is usually defined as the time it takes from 10% to 90% of Vdd. This definition may differ in various technologies. What is most important is that the definition used first, in characterizing the model, and second, in calculating slew in the design, be consistent to maintain accurate energy sums. The syntax is:

(SLEW <pin name> (<rising slope>)(falling slope>))

Where the pin name identifies an input pin on the instance and the rising and falling slew values are triplets of real numbers.

Load Capacitance Data

(CAPLOAD <output pin> (<capacitance>))

CAPLOAD is defined as the external capacitive load seen by an output driver. This should include the capacitance of the net, the driven input(s).

As with slew, it is very important to the accuracy of APAS to make sure that total load is defined and calculated with a definition that is equivalent to that used in characterization. Specifically, the capacitance of the driver (usually attributed to drain-source junction capacitance) may or may not have been considered part of external capacitance when calibrating the energy due to internal switching and dynamic short circuit. The definitions used in modelling and annotation should be consistent.

Voltage Data

The VOLTAGE triplet in the PAD file is made available to power models through the APAS power model interface. This will allow fine tuning based on voltage level. In the first version of APAS, only one voltage triplet will be allowed in a PAD file. If instance specific voltage is needed, it may be added in the future.

Temperature Data

The TEMPERATURE triplet is also made available to the power models as with voltage.

APPENDIX C

POWER MODEL COMPILER

This section describes the APAS power model compiler, a stand alone utility which converts the Power Model language into a binary file that is the actual model used by APAS.

To invoke the compiler, named pwrcom:

pwrcom <path to source> [-out <path to destination>] [-r]

Where:

<path to source> A system path to an ASCII description of the model in the specified syntax.

-out This optional switch allows the user to specify the destination of the resulting binary model.

<path to destination> A system path to the binary model. The file suffix ".pow" will automatically be appended if not included in the leaf name. The default will be to use the leaf name of the source with any extension removed and the ".pow" extension appended. Default directory is the current working directory.

-r This switch is required to overwrite an existing file.

Invoking this tool will parse the model language and check for syntax and semantic errors. Errors will be reported to standard output. A binary file will not be created if errors are found.

Error checking will NOT include verification of the port names with the functional models' ports. This will take place in the simulator environment.

APPENDIX D

CATALOG FILE SYNTAX

- Keywords are shown in capital letters. They will be parsed case insensitive.
- Syntax punctuation is in single quotes.
- Elements in symbols < > are defined further in this syntax.
- Elements in brackets -- [] -- are optional.
- A star -- '*' -- means "zero or more" of the element.
- A plus sign '+' means one or more of the element.
- Elements separated by bars (i.e. A|B|C) mean "one of A or B or C".
- **name** -- can be any string of alphanumeric characters, starting with a non-numeric.
- **chars** -- a string of any characters, except white space
- **number** - is a floating point number, may use exponential format (ex: 3.456e-2).

Note: Text preceded by double slash -- "//" -- are comments, not part of syntax.

```
<catalog file>      ::=   <local variable>* <catalog list>
<local variable>    ::=   name ':' <string>+ ';' // concatenated strings
<string>            ::=   <variable>
                    ||=   chars
<variable>          ::=   '$' '(' name ')' // name is a previously defined variable,
                                   // property or env variable
<catalog list>      ::=   CATALOG ':' <celltype var> '{' <cell model pair>+ '}'
<celltype var>      ::=   <variable> // this variable is evaluated for list lookup
<cell model pair>   ::=   name ':' <model path> ';' // names are expected
```

```
                                        // celltype values
<model path>    ::=    [<path branch> '/']* <model file>
<path branch>   ::=    <string>
<model file>::=>name  // leaf name of model file
```
