HotLeakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects

UNIV. OF VIRGINIA DEPT. OF COMPUTER SCIENCE TECH. REPORT CS-2003-05

Yan Zhang†, Dharmesh Parikh‡, Karthik Sankaranarayanan‡, Kevin Skadron‡, Mircea Stan†
†Dept. of Electrical and Computer Engineering, ‡Dept. of Computer Science
University of Virginia
Charlottesville, VA 22904

March 2003

Abstract

This report introduces *HotLeakage*, an architectural model for subthreshold and gate leakage that we have developed here at the University of Virginia. The most important features of HotLeakage are the explicit inclusion of temperature, voltage, gate leakage, and parameter variations, and the ability to recalculate leakage currents dynamically as temperature and voltage change due to operating conditions, DVS techniques, etc. HotLeakage provides default settings for 180nm through 70nm technologies for modeling cache and register files, and provides a simple interface for selecting alternate parameter values and for modeling alternative microarchitecture structures. It also provides models for several extant cache leakage control techniques, with an interface for adding further techniques. HotLeakage is currently a semi-independent module for use with SimpleScalar, but is sufficiently modular that it should be fairly easy to port to other simulators.

Because sub-threshold leakage currents are exponentially dependent on temperature and voltage, because gate leakage is growing so rapidly, and because parameter variations can have a profound effect on simulation accuracy, we hope that HotLeakage will serve as a useful tool for microarchitects to more accurately evaluate issues related leakage power. HotLeakage is available for download at `http://lava.cs.virginia.edu/HotLeakage`

1 Introduction

Power is rapidly become a design constraint not only in the domain of mobile devices but also in high performance processors. Dynamic power is caused by switching activity in CMOS circuits. It is the major source of total power dissipation in today's process generation. However static power, which is due to leakage current in the quiescent state of circuits, is gaining more importance. Technology scaling is increasing both the absolute and relative contribution of static power dissipation. The 2001 International Technology Roadmap for Semiconductors (ITRS) [24] predicts that in the next several processor generations, leakage may constitute as much as 50% of total power dissipation.

Recently, a great deal of research work in the architecture community has focused on reducing leakage power in the caches [11, 13, 14, 18, 22, 27, 28], branch predictor [15, 16], register file [2], issue queues [12, 21, 8, 9], and the ALUs [10]. Leakage control at the architecture level is attractive, because architectural techniques can control large groups of circuits (*e.g.* cache lines, banks, or the entire cache) at once. Yet most of these studies use only abstract models of leakage that do not fully account for all effects that may impact leakage, like supply voltage and temperature; others use circuit extracted parameters that are not easily incorporated into other researcher's models. Unlike for dynamic power, where widely-available simulators like Wattch [5] have enabled a widespread body of research, there is no widely available model for leakage power. This inhibits leakage research and leads to more approximate experiments. Although Butts and Sohi [7] propose a simple model for use at the architecture-simulation level of abstraction, no corresponding software is available. Most importantly, their model cannot easily model leakage when temperature, supply voltage, or threshold voltage vary dynamically: a new "normalized leakage" and k_{design}

must be calculated for every possible value. This is inconvenient although feasible for leakage-control schemes like drowsy cache that uses two supply voltages, but intractable for any leakage studies that account for dynamically varying temperature or involve dynamic voltage scaling.

This paper describes a software model of leakage—based on BSIM3 [3] technology data—that is publicly available on the web, computationally very simple, can easily be integrated into popular power-performance simulators like Wattch, can easily be extended to accommodate other technology models, and can easily be used to model leakage in a variety of structures (not just caches, which are the focus of this paper). It extends the Butts-Sohi model and corrects several important sources of inaccuracy. We call our model HotLeakage, because it includes the exponential effects of temperature on leakage. Temperature effects are important, because leakage current depends exponentially on temperature, and future operating temperatures may exceed 100° C [24]. In fact, HotLeakage also includes the heretofore unmodeled effects of supply voltage, gate leakage, and parameter variations.

HotLeakage has circuit-level accuracy because the parameters are derived from transistor-level simulation (Cadence tools). Yet like the Butts and Sohi model, simplicity is maintained by deriving the necessary circuit-level model for individual cells, like memory cells or decoder circuits, and then taking advantage of the regularity of major structures to develop abstract models that can be expressed in simple formulas similar to the Butts-Sohi model. All necessary components of this formula are encapsulated in lookup tables.

We hope that this new leakage model and its public availability will facilitate greater research on techniques for controlling leakage power at the architecture level.

2 An Accurate Model for Architects

Leakage current is influenced by the threshold voltage, channel physical dimensions, channel/surface doping profile, drain/source junction depth, gate oxide thickness, Vdd, temperature, and variations in all these parameters. For long channel devices, the leakage current is dominated by leakage from the drain-well and well-substrate reverse-bias pn junctions. For short channel transistors, because of the low threshold voltage, sub-threshold leakage is much higher. As gate oxides continue to scale, gate leakage is also becoming important. Keshavarzi, Roy, and Hawkins give an overview of these different leakage mechanisms in [19].

In [7], Butts and Sohi present a high-level model of sub-threshold leakage that neatly compartmentalizes some different issues affecting static power in a way that makes it easy to reason about leakage effects at the micro-architecture level. That model was never released as publicly-available software, and omits some other effects that are now recognized as essential. The Butts and Sohi framework, however, is ideal for use in architectural studies. We have therefore used the Butts and Sohi structure as a starting point for our model.

In this section, we briefly review the Butts and Sohi model and then describe how HotLeakage is derived. The next section describes how to use HotLeakage in a simulator like SimpleScalar/Wattch [5, 6].

2.1 The Butts and Sohi Model

In the elegant Butts and Sohi model, leakage is given by the following equation:

$$P_{static} = V_{CC} \cdot N \cdot k_{design} \cdot \hat{I}_{leak} \quad (1)$$

This formula must be computed for each circuit or block of interest, *e.g.* the data array or a cache or the cache's "edge logic" (decoders and sense amplifiers). V_{CC} is the supply voltage, and N is the number of transistors in the circuit, which could be estimated by comparing it with a circuit of known functionality. k$_{design}$ is a factor determined by the specific circuit topology and accounts for effects like transistor sizing, transistor stacking and the number and relationship of NMOS and PMOS transistors in a circuit. $\hat{I}_{leak}$ is a normalized leakage value for a single transistor-we call this a *unit leakage*-and includes all technology-specific effects like threshold voltage (V_T) and also factors in the operating temperature.

Using this model, it is easy to see the relationships of some major factors that a processor designer can control for leakage-power savings: given a unit leakage $\hat{I}_{leak}$, leakage power is proportional to operating voltage and the number of transistors in the unit of interest. For example, DVS affects leakage by reducing V_{CC}, and "turning off" some unit (a cache bank or part of an issue queue) by disconnecting its power supply effectively reduces N.

Other design choices affect k_{design} or $\hat{I}_{leak}$. Unfortunately, this model turns out not to be well suited for some types of leakage studies appearing in today's architecture literature. Recent work in low-leakage cache design [11, 13, 20, 23] as well as broader processor-design issues like thermal management [4, 17, 25, 26] manipulate parameters like V_T and temperature that are hidden in k_{design} or $\hat{I}_{leak}$ and this makes it more difficult both to reason about these effects (there is an exponential dependence of leakage on V_T, for example) and to actually simulate leakage power.

In summary, while [7] is a superb study of the various issues affecting sub-threshold leakage and related design issues, Equation (1) is not well-suited for actual simulation work.

2.2 Calculating Leakage—Single k_{design}

To develop a portable simulation module for use with various architecture-level simulators, we retain the notions of k_{design} and unit leakage but compute the unit leakage dynamically during the simulation using the BSIM3 [3] leakage-current equation. This lets us explicitly account for temperature, supply voltage, and threshold voltage as key parameters, and includes the important DIBL effect which is sensitive to supply voltage.

2.2.1 Modeling Methodology

Based on the BSIM3 v3.2 [3] equation for leakage in a MOSFET transistor, our leakage model of a single transistor is given by the following equation:

$$I_{leakage} = \mu_0 \cdot C_{OX} \cdot \frac{W}{L} \cdot e^{b(V_{dd}-V_{dd0})} \cdot v_t^2 \cdot \left(1 - e^{\frac{-V_{dd}}{v_t}}\right) \cdot e^{\frac{-|v_{th}|-V_{off}}{n \cdot v_t}} \tag{2}$$

Low-level parameters are derived using transistor-level simulations: u_0 is the zero bias mobility, C_{OX} is gate oxide capacitance per unit area, W/L is the aspect ratio of the transistor, $e^{b(V_{dd}-V_{dd0})}$ is the DIBL factor derived from the curve fitting method, V$_{dd0}$ is the default supply voltage for each technology (V$_{dd0}$=2.0 for 180nm, V$_{dd0}$=1.5 for 130nm, V$_{dd0}$=1.2 for 100nm and V$_{dd0}$=1.0 for 70nm), $v_t = kT/q$ is the thermal voltage, V$_{th}$ is threshold voltage which is also a function of temperature, n is the subthreshold swing coefficient, V$_{off}$ is an empirically determined BSIM3 parameter which is also a function of threshold voltage. In these parameters, u_0, C_{OX},W/L and V$_{dd0}$ are statically defined parameters; the DIBL factor b, subthreshold swing coefficient n and V$_{off}$ are derived from the curve fitting method based on the transistor level simulations; V$_{dd}$, V$_{th}$ and $v_t = kT/q$ are calculated dynamically in the simulations.

The above equation is based on two assumptions:

1. V$_{gs}$=0 — we only consider the leakage current when the transistor is off.
2. V$_{ds}$=V$_{dd}$ — we only consider a single transistor here; the stack effect and the interaction among multiple transistors are taken into account when we model the cell using Equation 3

Figure 1 show the comparison of leakage current calculated by our model and the transistor-level simulation. From Figure 1a, 1b, and 1c, we can see that for the ratio W/L, supply voltage V$_{dd}$ and temperature T, our results perfectly match the simulation results. Figure 1d shows that after threshold voltage increases to some value, the modeled leakage current does not decrease anymore. This is due to the simplicity of our model which only considers the subthreshold leakage and DIBL effect. It is only of concern if threshold voltage is beyond the normal value.

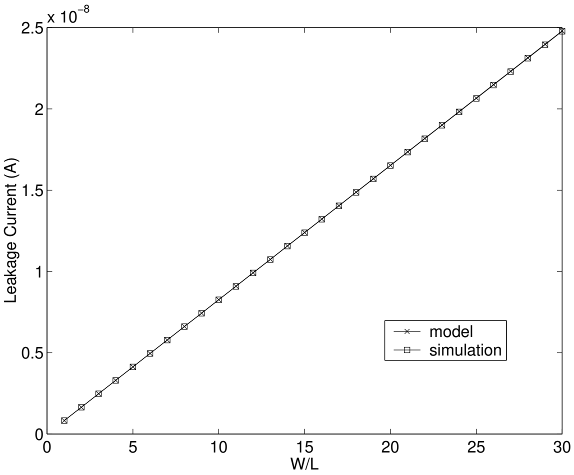
For a specific cell, the leakage current is given by the following equation:

$$I_{cell\_leakage} = (n_N \cdot I_N + n_p \cdot I_p) \cdot k_{design} \tag{3}$$

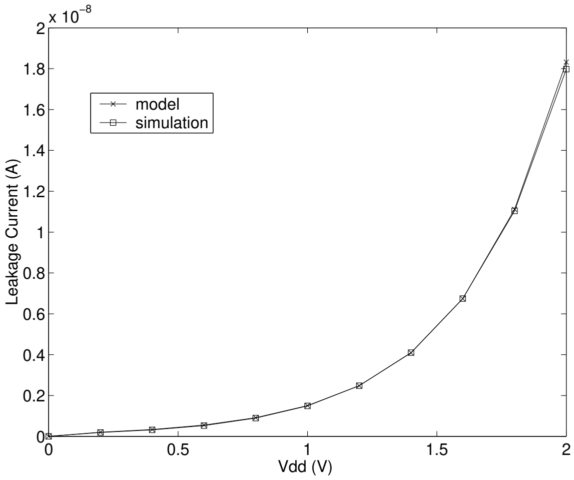
n_N and n_P are the number of NMOS and PMOS transistors in the cell, and I_N and I_P are the calculated leakage current of NMOS and PMOS according to equation 2; when aspect ratio $W/L = 1$ we call them *unit leakage*; k_{design} is the design factor determined by the stack effect and aspect ratio of transistors.

k_{design} is derived from transistor-level simulation of an actual design and layout of a cell of interest. Given a cell, the average leakage $I_{cell\_leakage}$ is derived from the transistor-level simulation with all possible input combinations. Equation 3 gives k_{design}.

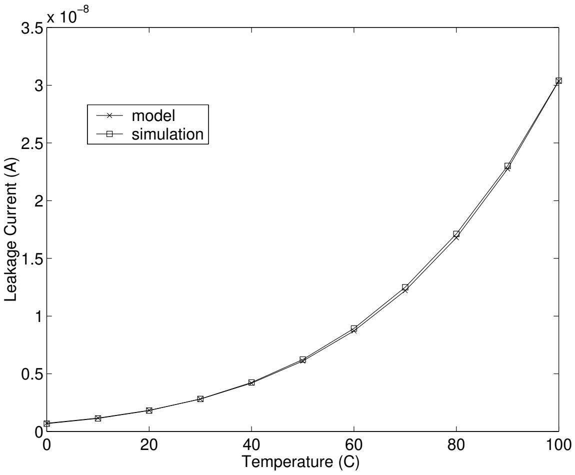
$$k_{design} = I_{cell\_leakage}/(n_N \cdot I_N + n_p \cdot I_p) \tag{4}$$



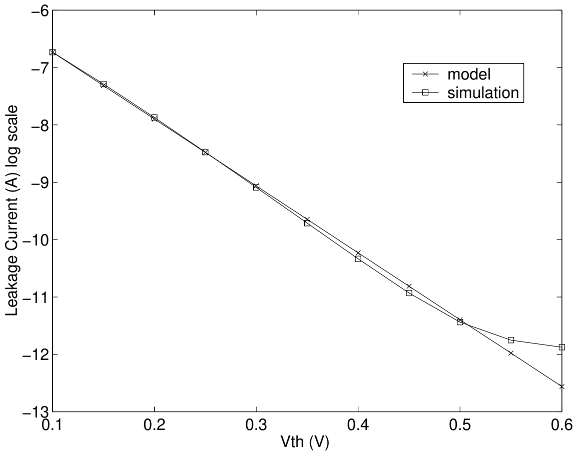
(a) Leakage vs. Aspect Ratio (W/L)



(b) Leakage vs. Supply Voltage (Vdd)



(c) Leakage vs. Temperature (T)

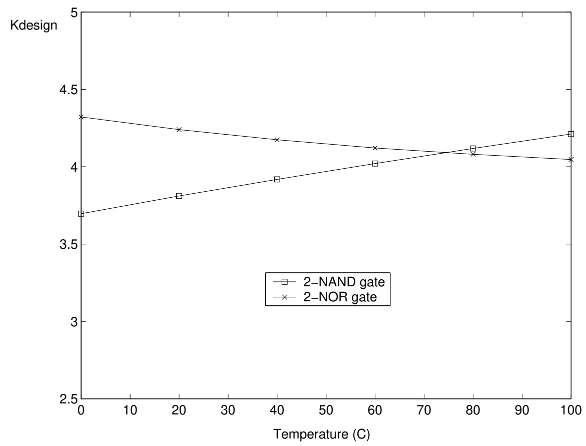


(d) Leakage vs. Threshold Voltage (Vth)

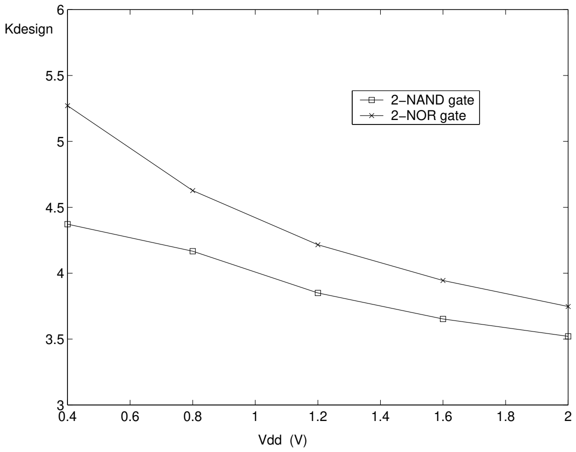
Figure 1: Comparisons of the proposed HotLeakage model against circuit-level simulation results.

2.2.2 Model Parameters

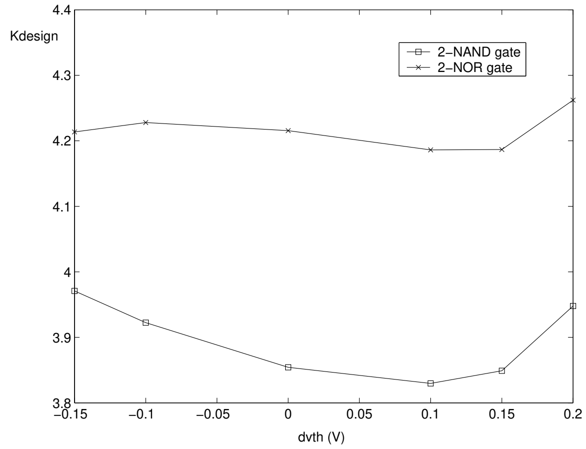
k_{design} is the factor which accounts for the transistor aspect ratio (W/L) and the stack effect. (The stack effect refers to the additional reduction in leakage when multiple series-connected transistors are off; for example, sleep transistors take advantage of this.) Unlike the Butts and Sohi model, we find that k_{design} does in fact vary with temperature, supply voltage, threshold voltage, and channel length. Figure 2 shows the k_{design} of a two input NAND gate and a two input NOR gate with respect to these values. HotLeakage therefore re-computes k_{design} every time it re-computes leakage, so that dynamically varying supply voltage, temperature, etc. can be accounted for. Note that X-axis of Figure 2c is dV_{th}; $d_V th$=0 means that the threshold voltage of N and P transistors stays at the default value. When dV_{th} increases, the threshold voltage of N and P transistors also increases with the same magnitude, and vice versa. The parameters of N and P transistors should scale in the same direction with the same value in order to keep k_{design} constant. This is a limitation of single-k_{design} model, discussed further in the next section.



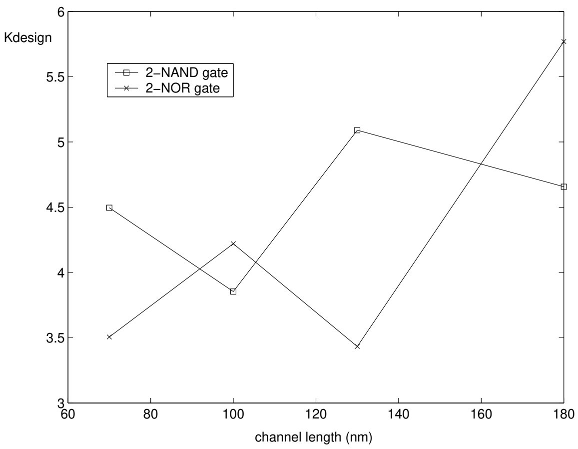
(a) k_{design} vs. Temperature



(b) k_{design} vs. Supply Voltage



(c) k_{design} vs. Threshold Voltage



(d) k_{design} vs. Channel Length

Figure 2: k_{design} for a two-input NAND and a two-input NOR as a function of various parameters.

2.3 Improved leakage model—Double k_{design}: k_n and k_p

The Single k_{design} model is suitable for cases where the parameters of N and P transistors are very close. If these two sets parameters of N and P transistors differ significantly, different k_{design} should be applied to these two type of transistors. This is also mentioned in [7]. We implemented this idea in the improved leakage model: double-k_{design} model. For a specific cell, the leakage current is now given by this different equation:

$$I_{cell\_leakage} = n_n \cdot K_n \cdot I_n + n_p \cdot K_p \cdot I_p \tag{5}$$

k_n and k_p are the design factors of N and P transistors and they can be derived by the same method as in the single-k_{design} model. For a given cell, we divide all possible inputs into two groups: one group inputs will turn off the pull-down network composed of N transistors. The other group will turn off the pull-up network composed of P transistors. Thus the leakage currents are also divided into two groups $I_{1n}, I_{2n}, \ldots, I_{kn}, \ldots$ and $I_{1p}, I_{2p}, \ldots, I_{kp}, \ldots$. I_{kn} is the leakage current when the pull-down network is turned off, while I_{kp} is the leakage current when the pull-up network is turned off. k_n and k_p are given by the following equation:

$$k_n = (I_{1n} + I_{2n} + \cdots + I_{kn} + \cdots)/(N * n_n * I_n) \tag{6}$$

$$k_p = (I_{1p} + I_{2p} + \cdots + I_{kp} + \cdots)/(N * n_p * I_p) \tag{7}$$

N is the number of all possible combinations. For example, Figure 3 is the diagram of two input NAND gate. There are

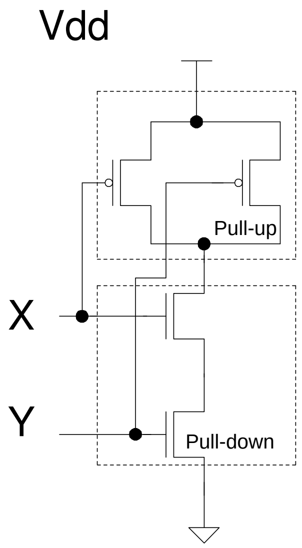


Figure 3: Two-input NAND gate.

two inputs X and Y, which make four possible combinations. There are three combinations: $(X = 0, Y = 0)$,$(X = 0, Y = 1)$ and $(X = 1, Y = 0)$ which turn off the pull-down network. I_{1n}, I_{2n} and I_{3n} are the leakage currents corresponding to these three inputs. The only combination that turns off the pull-up network is $(X = 1, Y = 1)$ and $I_1 p$ is the corresponding leakage current. k_n and k_p are given by:

$$k_n = (I_{1n} + I_{2n} + I_{3n})/(N * n_n * I_n) \tag{8}$$

$$k_p = I_{1p}/(N * n_p * I_p) \tag{9}$$

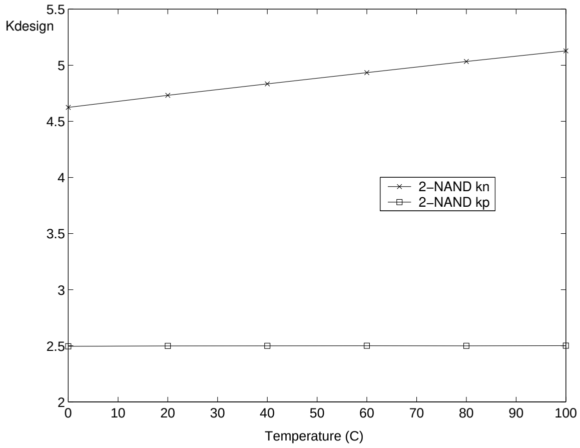
Here N equals 4.

Figure 4 shows the k_n and k_p of a two-input NAND gate with respect to temperature, supply voltage, threshold voltage, and channel length, while Figure 5 presents the same comparison for a two-input NOR gate.

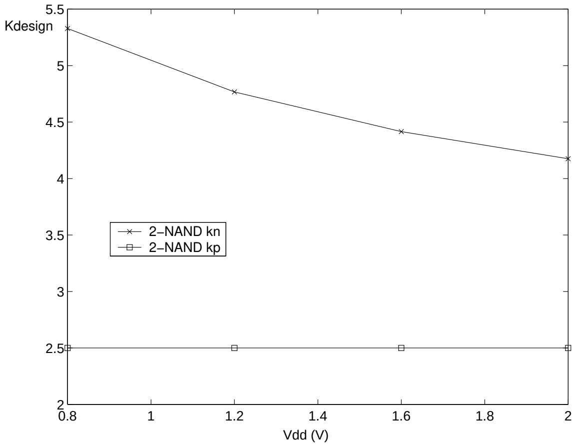
Note that k_n and k_p are still not constant with respect to these parameters. Figures 4c, 4d, 5c, and 5d show that k_n and k_p stay constant when we only vary the threshold voltage of N or P transistors. This means that the double k_{design} model has the important property that it is able to handle the differential scaling of N and P transistors that is widely used in the MTCMOS technologies. Figures 4c, 4d, 5c, and 5d also show that k_n and k_p have a linear relationship with temperature and supply voltage. We incorporate these features into our leakage model and k_n, k_p are calculated dynamically with respect to different temperatures and supply voltages. Figure 4e and 5e show that k_n and k_p do not have explicit relations with a different technology. In our leakage model, we derive different k_{design} for different technology via simulations.

2.4 Gate Leakage and GIDL effect

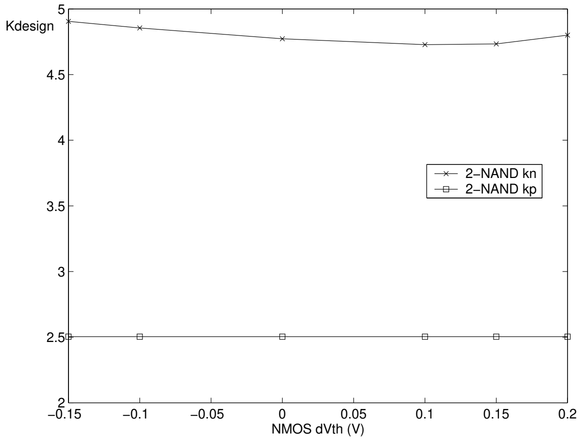
In order to improve device performance, gate oxide thickness is projected to scale aggressively for future technology nodes [24]. The result is that gate leakage through the gate oxide increases significantly due to the direct tunneling current. Our model includes gate leakage for 70nm technology where the gate leakage becomes dominant. To get an explicit equation for gate leakage calculations is very difficult and also unnecessary for an architectural-level model. We use AIM-spice [1] as the circuit simulator, which includes BSIM4 among the supported models for gate leakage.



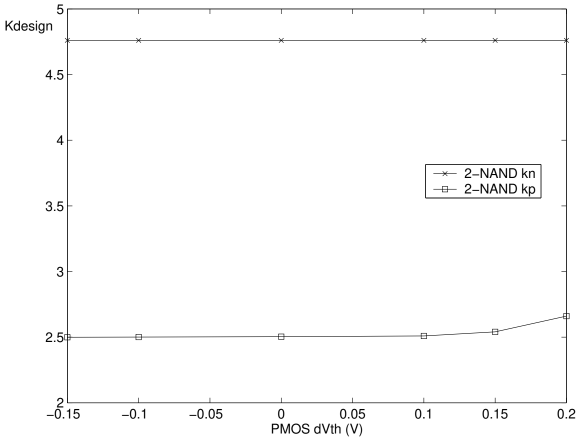
(a) k_{design} vs. Temperature



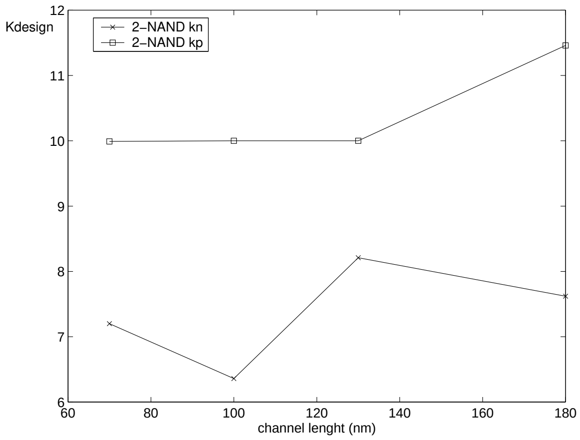
(b) k_{design} vs. Supply Voltage



(c) k_{design} vs. Threshold Voltage for N-type

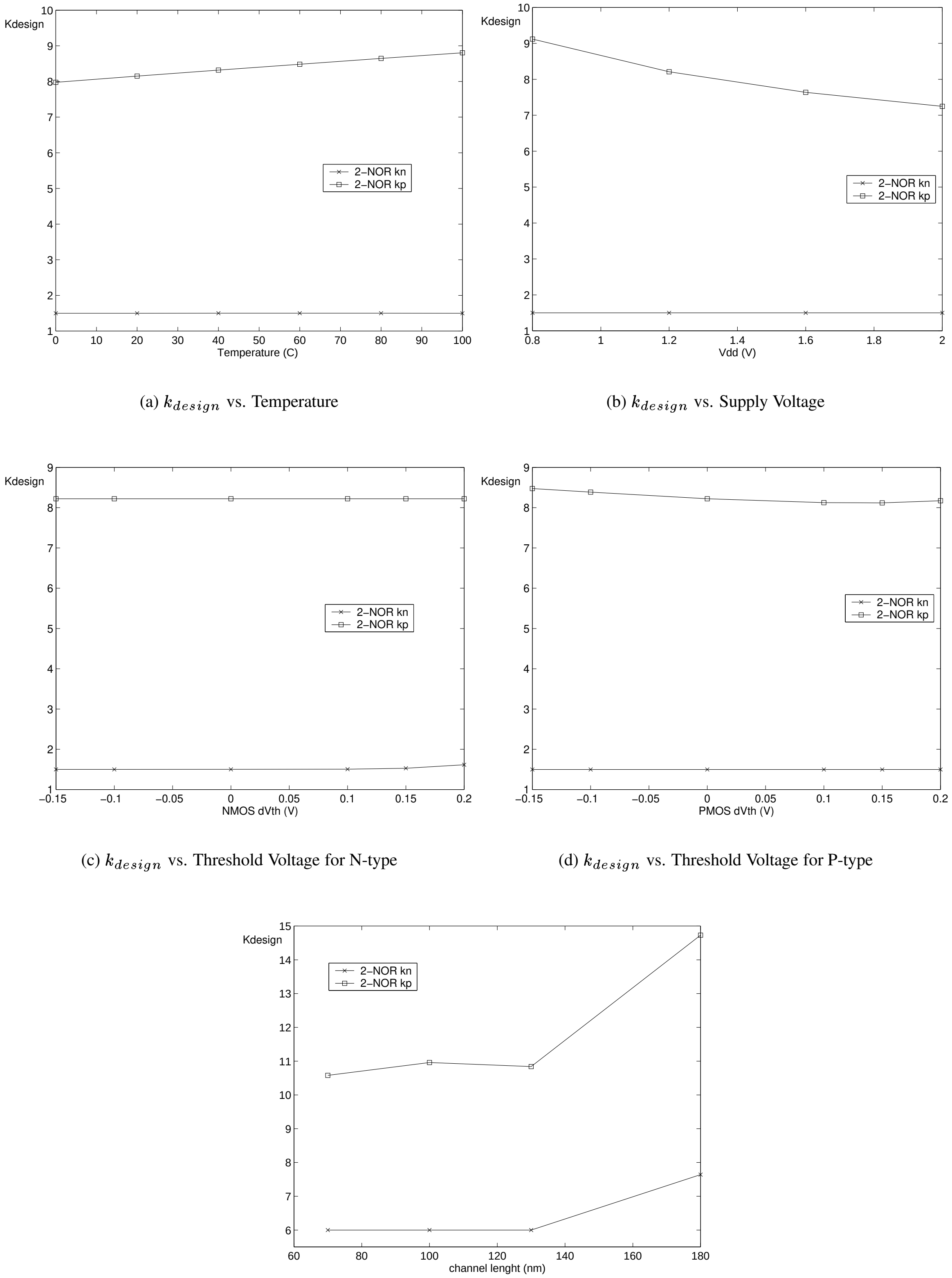


(d) k_{design} vs. Threshold Voltage for P-type



(e) k_{design} vs. Channel Length

Figure 4: k_n and k_p for a two-input NAND.



(a) k_{design} vs. Temperature

(b) k_{design} vs. Supply Voltage

(c) k_{design} vs. Threshold Voltage for N-type

(d) k_{design} vs. Threshold Voltage for P-type

(e) k_{design} vs. Channel Length

Figure 5: k_n and k_p for a two-input NOR.

Gate current parameters have been adjusted to target 40 nA/um gate leakage in 70nm technology at 1.2nm oxide thickness and 0.9 V supply voltage at room temperature (300K) as predicted in [24]. Gate leakage is strongly dependent on the gate oxide thickness t_{ox} and supply voltage. It is weakly dependent on the temperature. From the transistor-level simulations, we derived these factors with curve-fitting method and incorporated it into our models.

GIDL effect occurs at low gate voltage and high drain voltage bias. This effect will raise the leakage current when gate voltage goes negative. It becomes worse when biasing the substrate to negative voltage for N transistors and to positive voltage for P transistors. This will limit the reverse body-biasing (RBB) technique.

2.5 Parameter Variations

Device parameter variations can be divided into two categories: inter-die (die-to-die) variation and intra-die (within-die) variation.

Inter-die variation is the difference in the value of a parameter across nominally identical die and is typically accounted as a shift in the mean of some parameter value equally across all device or structures on any one chip. For purposes of circuit design, it is usually sufficient to lump all the contributions in the inter-die variation into a single variation component with a mean and variance.

Intra-die variation is the deviation occurring spatially within any one die. It may have a variety of sources depending on the physics of the manufacturing steps. In contrast to inter-die variation (affecting all devices on any one chip equally), intra-die variation contributes to the mismatch behavior between structures on the same chip.

Due to both inter-die and intra-die parameter variations, there is significant variation in leakage power. Thus parameter variations must be taken into account in the new leakage model. Inter-die variation can be characterized as a global mean and variance while intra-die variation is more complicated. In this version our model only includes the inter-die variation.

There are four parameters which we are interested in. They are L: length of the transistor; t_{ox}: thickness of the gate oxide; V_{dd}: supply voltage; and V_{th}: threshold voltage of the transistor. For each parameter, user can give the specific mean μ, variance σ, and the number of samples N. In the initializing phase of the simulation, N gaussian distribution samples are generated and the leakage currents are also calculated accordingly. The mean of those leakage currents is used in the following simulations in order to show the effects of the parameter variations.

3 Using HotLeakage

The HotLeakage simulation tool is currently based on Wattch [5] and also uses some code from the Princeton/Agere cache-decay [18] simulator. It implements various cache leakage-control techniques in cache.c, cache_leak_ctrl.c and sim-outorder.c, with calls to the suitable routines within HotLeakage (leakage.c and leakinit.c). Using this system as a starting point, the simulator can be extended easily to model new leakage-control techniques for the cache or for other regular structures in the processor like the register file, issue queues etc. Because HotLeakage is a separate module with its own interface, it should be fairly easy to port to other simulators.

3.1 How to Use the HotLeakage Software Within an Architecture Simulator

The HotLeakage simulator is a configurable module. The various parameters related to the leakage power modeling and the leakage control techniques are specified at the command line (See Section 3.3, and please also read, README.HotLeakage.Howto.FAQ and README_HotLeakage_Model.doc, supplied with the tool for further details). HotLeakage dynamically tracks leakage for each cell of interest (*e.g.*, an SRAM cell) and this information is then translated into leakage at the architecture level. The functions that calculate leakage for each structure of the micro-architecture are in the main leakage module (leakage.c). These need to be called whenever any of the parameters—like temperature, supply voltage etc.—that affect leakage is changed. These functions will recalculate the leakage currents using the HotLeakage model. HotLeakage and the accompanying simulation infrastructure currently model leakage of caches and register files; adding models for other cache-like structures is very simple. Please follow the instructions given in the README.howto file supplied with the tool to model a different structure.

The power-performance simulator, *e.g.* Wattch, is responsible for implementing the leakage-control technique and using the HotLeakage values accordingly. We have implemented a generic abstraction for modeling leakage control techniques based on the dynamic resizing of structures, allowing us to study techniques like gated-V$_{ss}$ [18], drowsy cache [11], and reverse-body-bias [20]. Leakage control by resizing can be typically classified into two categories: state-preserving and state-losing methods. Most dynamic leakage-control techniques partition a structure into active and passive portions. This can be done at various granularities; most recent work has done this at the granularity of rows in the SRAM array, which correspond to cache lines.

These leakage control techniques also require some extra hardware that adds to the area of the structure. Hence, these methods have the following costs

- Dynamic power due to the extra hardware
- Leakage power due to the extra hardware
- Dynamic power due to the mode transitions (active to passive and vice-versa)
- Dynamic power due to extra latency (or state loss) in accessing the structure

The benefit of these techniques is the leakage power saved due to the portion of the structure that is in the passive mode. This saving is proportional to the percentage area of the structure in passive mode to the total area averaged over the run of the program (this is called the *turnoff ratio*). In modeling the effectiveness of the individual control techniques, the HotLeakage simulator measures the costs against the benefits. Dynamic power due to the extra latency in accessing the structures is not explicitly modeled by our simulator but is subsumed under Wattch's modeling of dynamic power.

The simulator currently models leakage control in caches using the above costs and benefits. The dynamic power calculations are performed using Wattch routines. The leakage power is calculated using our model as configured by the command line options. The simulator can be easily extended to other regular structures in the processor. The additional code required will be very similar to the modeling of cache leakage control, with few structure-specific modifications.

As mentioned, the implementation is currently designed for use with Wattch. However, the leakage model itself is implemented in a modular and parameterized form so that the user can explore the effects of threshold voltages, temperature, and supply voltages for individual components. This means that it should also be straightforward to port the leakage model for use with other simulators.

3.2 Major Techniques for Leakage Control in Caches

3.2.1 Lowering the Quiescent V_{DD} (Gated-V_{ss})

Leakage currents decrease as the supply voltage (V_{DD}) is lowered. The *gated-V_{dd}* structure was introduced in [22] as a way to reduce leakage power by using a high threshold "header" transistor to disconnect a cell, row, or way in the cache from V_{dd}. This high-threshold transistor drastically reduces the leakage of the circuit because the high-threshold transistor effectively breaks the connection to the power supply. While this technique is very efficient in saving leakage, there is the disadvantage that the cell looses its state (information). Thus this is a *state-losing* technique. This means that there will be some performance penalty when the data in the cell is accessed and needs to be fetched from a farther level of the cache. This technique was used in [18] to shut down lines in a cache to save leakage. Because the sleep transistor is more effective as a "footer" on the connection to ground–it is easier to prevent bitline leakage this way–this technique is better called *gated-V_{ss}*.

3.2.2 Multiple Threshold CMOS (MTCMOS)

It is clear from above discussion that threshold voltage is one of the most important parameters to influence the leakage current. Multiple threshold CMOS technique was proposed in [20]. For active mode, the low threshold voltage is preferred because of the higher performance. However, for the standby mode of operation, high threshold voltage is useful for reduction of leakage power. Hence, if transistors can be set to different threshold voltages, most likely using reverse-body-bias (RBB), the threshold voltage can be set according to different modes of operation. This does not lose the data stored in a cell, so this is a *state-preserving* technique. There is still some overhead, however, when accessing a unit that is in standby mode, because the threshold voltage must be returned to the proper level before the value can be read.

3.2.3 Drowsy Caches

This method, proposed in [11], utilizes dynamic voltage scaling to reduce the supply voltage of the cell to approximately 1.5 times V_t. This reduces leakage current dramatically due to short-channel effects and preserves the value that is stored, making this another *state-preserving* technique. Like MTCMOS, there is still some overhead because V_{dd} must be returned to the proper level before the value can be read.

For all of the above techniques during the initialization phase of the simulation the leakage currents for the cache, with and without the specified technique turned on, is calculated using the HotLeakage model. Every cycle the leakage energy of the cache is calculated using the turn-off ratio (the fraction of cache using one of the above leakage saving technique) and the leakage currents calculated as above.

3.3 HotLeakage Parameters

To use HotLeakage with currents based on BSIM3 models and our pre-determined values of k_design, it is only necessary to specify the technology parameter; this specifies the technology, e.g. 70nm.

Other paramters can also be configured, but all have reasonable default values except that the default for all the parameter-variation items is zero. The available range of values for the parameter-variation items are number of standard deviations between 0 and 1.

- T_{ox}: Thickness of oxide. If not specified, a default value will be taken.
- V_{dd}: Supply Voltage. If not specified, a default will be taken according to the specified technology.
- Temperature: Temperature of the specific structure. If not specified, the default will be taken as room temperature.
- Threshold voltages: Threshold voltages of the transistors of specific structures. Default values are taken if they are not specified.
- T_{oxide} variation: Variation in oxide thickness due to inter-die variations. This should vary with technology node.
- V_{dd} variation: Variation in V_{dd} due to inter-die variations. This should vary with technology node.
- Channel-length variation: Variation in channel length due to inter-die variations. This should vary with technology node.
- Threshold-Voltage variation (p-type): Variation in threshold voltage of a p-type transistor due to inter-die variations. This should vary with technology node.
- Threshold-Voltage variation (p-type): Variation in threshold voltage of an n-type transistor due to inter-die variations. This should vary with technology node.

4 Conclusions and Future Work

HotLeakage provides the first publicly-available microarchitecture-level leakage-modeling software of which we are aware. Its most important features are the explicit inclusion of temperature, voltage, gate leakage, and parameter variations.

HotLeakage provides default settings for 180nm through 70nm technologies (based upon BSIM3 models) for modeling cache and register files, and provides a simple interface for selecting alternate parameter values and for modeling alternative microarchitecture structures. HotLeakage also provides models for several extant cache leakage-control techniques, with an interface for adding further techniques.

Because sub-threshold leakage currents are exponentially dependent on temperature and voltage, because gate leakage is growing so rapidly, and because parameter variations can have a profound effect on simulation accuracy, we hope that HotLeakage will serve as a useful tool for microarchitects to more accurately evaluate issues related leakage power.

HotLeakage is not yet fully independent of the SimpleScalar/Wattch framework upon which it was developed. In future versions of HotLeakage, we hope to make the leakage model an entirely self-contained library; make the code more readable and self-documenting; and provide better support for modeling leakage in other microarchitectural structures. We welcome user contributions in these regards.

The HotLeakage tool, with all the supporting documents, is available at *http://lava.cs.virginia.edu/HotLeakage*

Acknowledgments

This work was funded in part by the National Science Foundation under grant nos. CCR-0133634, CCR-0105626, and MIP-9703440, a grant from Intel MRL, and an Excellence Award from the Univ. of Virginia Fund for Excellence in Science and Technology. We would also like to thank Amit K. Naidu for his help on an early version of this document, and Margaret Maronosi and Stefanos Kaxiras for their advice on a variety of issues related to leakage power.

References

[1] Aim-Spice Home Page. http://www.aimspice.com.

[2] A. Alvandpour, R. Krishnamurthy, K. Soumyanath, and S. Borkar. A low-leakage dynamic multi-ported register file in 0.13um CMOS. In *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, pages 68–71, Aug. 2001.

[3] U. C. Berkeley. BSIM3v3.1 SPICE MOS device models, 1997. `http://www-device.EECS.Berkeley.EDU/ bsim3/`.

[4] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, pages 171–82, Jan. 2001.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, June 2000.

[6] D. C. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. *Computer Architecture News*, 25(3):13–25, June 1997.

[7] J. A. Butts and G. S. Sohi. A static power model for architects. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 191–201, Dec. 2000.

[8] A. Buyuktosunoglu, D. H. Albonesi, P. Bose, P. W. Cook, , and S. E. Schuster. Tradeoffs in power-efficient issue queue design. In *Proceedings of the 2002 International Symposium on Low Power Electronics and Design*, Aug. 2002.

[9] A. Buyuktosunoglu, S. E. Schuster, D. Brooks, P. Bose, P. W. Cook, and D. H. Albonesi. An adaptive issue queue for reduced power at high performance. In *Workshop on Power-Aware Computer Systems*, Nov. 2000.

[10] S. Dropsho, V. Kursun, D. H. Albonesi, S. Dwarkadas, and E. G. Friedman. Managing static leakage energy in microprocessor functional units. In *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 321–32, Nov. 2002.

[11] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: Simple techniques for reducing leakage power. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 147–57, May 2002.

[12] D. Folegnani and A. Gonzalez. Energy-effective issue logic. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 248–59, June. 2001.

[13] H. Hanson et al. Static energy reduction techniques for microprocessor caches. In *Proceedings of the 2001 International Conference on Computer Design*, pages 276–83, Sept. 2001.

[14] S. Heo, K. Barr, M. Hampton, and K. Asanović. Dynamic fine-grain leakage reduction using leakage-biased bitlines. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 137–47, May 2002.

[15] Z. Hu, P. Juang, P. Diodato, S. Kaxiras, K. Skadron, M. Martonosi, and D. W. Clark. Managing leakage for transient data: Decay and quasi-static memory cells. In *Proceedings of the 2002 International Symposium on Low Power Electronics and Design*, Aug. 2002. To appear.

[16] Z. Hu, P. Juang, K. Skadron, D. Clark, and M. Martonosi. Applying decay strategies to branch predictors for leakage energy savings. In *Proceedings of the 2002 International Conference on Computer Design*, pages 442–45, Sept. 2002.

[17] W. Huang, J. Renau, S.-M. Yoo, and J. Torellas. A framework for dynamic energy efficiency and temperature management. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 202–13, Dec. 2000.

[18] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, July 2001.

[19] A. Keshavarzi, K. Roy, and C. F. Hawkins. Intrinsic leakage in low power deep submicron CMOS ICs. In *Proc. of the 1997 International Test Conference*, pages 146–55, Nov. 1997.

[20] K. Nii et al. A low power SRAM using auto-backgate-controlled MT-CMOS. In *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*, pages 293–98, Aug. 1998.

[21] D. Ponomarev, G. Kucuk, and K. Ghose. Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources. In *Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 248–59, Dec. 2001.

[22] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, pages 90–95, July 2000.

[23] K. Roy. Leakage power reduction in low-voltage CMOS designs. In *Proceedings of the International Conference on Electronics, Circuits, and Systems*, pages 167–73, 1998.

[24] SIA. *International Technology Roadmap for Semiconductors*, 2001.

[25] K. Skadron, T. Abdelzaher, and M. R. Stan. Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture*, pages 17–28, Feb. 2002.

[26] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, D. Tarjan, and K. Sankaranarayanan. Temperature-aware microarchitecture. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, June 2003. To appear.

[27] S. Velusamy, K. Sankaranarayanan, D. Parikh, T. Abdelzaher, and K. Skadron. Adaptive cache decay using formal feedback control. In *Proceedings of the 2002 Workshop on Memory Performance Issues*, May 2002.

[28] H. Zhou, M. Toburen, E. Rotenberg, and T. Conte. Adaptive mode control: A static-power-efficient cache design. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2001.

License Terms

The following is the license agreement for all components in the HotLeakage archive with exceptions listed in Section II.

I. License

II. Exceptions

SimpleScalar. SimpleScalar simulators, tools, and functions are held under license by SimpleScalar LLC: SimpleScalar Tool Suite, (c)1994-2001 Todd M. Austin, Ph.D. and SimpleScalar, LLC All Rights Reserved.

THIS IS A LEGAL DOCUMENT BY USING SIMPLESCALAR, YOU ARE AGREEING TO THESE TERMS AND CONDITIONS.

No portion of this work may be used by any commercial entity, or for any commercial purpose, without the prior, written permission of SimpleScalar, LLC (info@simplescalar.com). Nonprofit and noncommercial use is permitted as described below.

- SimpleScalar is provided AS IS, with no warranty of any kind, express or implied. The user of the program accepts full responsibility for the application of the program and the use of any results.
- Nonprofit and noncommercial use is encouraged. SimpleScalar may be downloaded, compiled, executed, copied, and modified solely for nonprofit, educational, noncommercial research, and noncommercial scholarship purposes provided that this notice in its entirety accompanies all copies. Copies of the modified software can be delivered to persons who use it solely for nonprofit, educational, noncommercial research, and noncommercial scholarship purposes provided that this notice in its entirety accompanies all copies.
- ALL COMMERCIAL USE, AND ALL USE BY FOR PROFIT ENTITIES, IS EXPRESSLY PROHIBITED WITHOUT A LICENSE FROM SIMPLESCALAR, LLC (info@simplescalar.com).
- No nonprofit user may place any restrictions on the use of this software, including as modified by the user, by any other authorized user.
- Noncommercial and nonprofit users may distribute copies of SimpleScalar in compiled or executable form as set forth in Section 2, provided that either: (A) it is accompanied by the corresponding machine-readable source code, or (B) it is accompanied by a written offer, with no time limit, to give anyone a machine-readable copy of the corresponding source code in return for reimbursement of the cost of distribution. This written offer must permit verbatim duplication by anyone, or (C) it is distributed by someone who received only the executable form, and is accompanied by a copy of the written offer of source code.
- SimpleScalar was developed by Todd M. Austin, Ph.D. The tool suite is currently maintained by SimpleScalar LLC (info@simplescalar.com). US Mail: 2395 Timbercrest Court, Ann Arbor, MI 48105.

Copyright (c)1994-2001 by Todd M. Austin, Ph.D. and SimpleScalar, LLC.

Wattch Wattch is a microarchitectural power model built on top of simplescalar. At the time this document was written, there were no licensing restrictions on Wattch itself, but users should check the Wattch homepage at http://www.ee.princeton.edu/ dbrooks/wattch-form.html

Wattch is built on SimpleScalar and CACTI and hence is still subject to the licensing terms associated with those software releases.

CACTI. CACTI is subject to the following licensing terms.

Copyright 1994 Digital Equipment Corporation and Steve Wilton, All Rights Reserved

Permission to use, copy, and modify this software and its documentation is hereby granted only under the following terms and conditions. Both the above copyright notice and this permission notice must appear in all copies of the software, derivative works or modified versions, and any portions thereof, and both notices must appear in supporting documentation.

Users of this software agree to the terms and conditions set forth herein, and hereby grant back to Digital a non-exclusive, unrestricted, royalty- free right and license under any changes, enhancements or extensions made to the core functions of the software, including but not limited to those affording compatibility with other hardware or software environments, but excluding applications which incorporate this software. Users further agree to use their best efforts to return to Digital any such changes, enhancements or extensions that they make and inform Digital of noteworthy uses of this software. Correspondence should be provided to Digital at:

Director of Licensing
Western Research Laboratory
Digital Equipment Corporation
100 Hamilton Avenue
Palo Alto, California 94301

This software may be distributed (but not offered for sale or transferred for compensation) to third parties, provided such third parties agree to abide by the terms and conditions of this notice.

THE SOFTWARE IS PROVIDED "AS IS" AND DIGITAL EQUIPMENT CORP. DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS.

Cache-Decay Code. Our functions to implement various forms of cache leakage control are based on the original cache-decay codebase provided by Zhigang Hu and Margaret Martonosi of Princeton University and Stefanos Kaxiras of Agere Corp. This code consists chiefly of the functions update_cache_stats(), update_cache_decay(), update_cache_block_stats_when_hit(), update_cache_block_stats_when_miss(), and various other changes in sim-outorder.c and cache.c. These Princeton/Agere modifications are subject to the following license terms: