Managing Static Leakage Energy in Microprocessor Functional Units *

Steven Dropsho†, Volkan Kursun, David H. Albonesi,
Sandhya Dwarkadas†, and Eby G. Friedman

†Department of Computer Science
Department of Electrical and Computer Engineering
University of Rochester
Rochester, NY 14627

Abstract

Static energy due to subthreshold leakage current is projected to become a major component of the total energy in high performance microprocessors. Many studies so far have examined and proposed techniques to reduce leakage in on-chip storage structures. In this study, static energy is reduced in the integer functional units by leveraging the unique qualities of dual threshold voltage domino logic.

*Domino logic has desirable properties that greatly reduce leakage current while providing fast propagation times. However, due to the energy cost of entering the low leakage current state (*sleep *mode), domino logic has thus far been used only for leakage reduction in the long-term standby mode. We examine the utility of the* sleep *mode (while considering the aforementioned costs) when idle times are relatively short, one to a few hundred cycles, as is often the case for functional units.*

Using an analytical energy model suitable for architecture-level analysis, we explore the interaction of the application and technology, and the effect on energy and performance as the underlying parameters are varied, on a set of benchmarks. Our results show that if the leakage approaches the magnitude as projected in the literature, even for short idle intervals as few as ten cycles, an aggressive policy of activating the sleep mode at every idle period performs well and a more complex control strategy may not be warranted. We also propose a simple design, called Gradual Sleep*, to reduce the energy impact of using the* sleep *mode for smaller idle periods.*

*This work was supported in part by NSF grants EIA-9972881, EIA-0080124, CCR–9702466, CCR–9701915, CCR–9811929, CCR-9988361, and CCR–9705594; by DARPA/ITO under AFRL contract F29601-00-K-0182; by New York State Office of Science, Technology & Academic Research to the Center for Advanced Technology – Electronic Imaging Systems and the Microelectronics Design Center; by an IBM Faculty Partnership Award; and by external research grants from the corporations of Intel, DEC/Compaq, Xerox, Eastman Kodak, Lucent Technologies, and Photon Vision Systems, Inc.

1 Introduction

Energy dissipation has become a critical design constraint in high performance microprocessors. Until recently, the focus has been on the *dynamic energy* dissipated in CMOS circuits. In older technologies, the majority of the energy is dissipated when transistors switch (transient power dissipation). When the circuits are not active the current is extremely low relative to switching, and thus, the *static energy* consumed is negligible. This exaggerated relationship between dynamic and static energy will experience a marked shift in the near future.

Static energy dissipation is a result of *leakage current* due to the finite-resistance of the off transistors between power and ground that exist whenever power is applied to a CMOS circuit. The magnitude of the leakage current is highly dependent on the threshold voltage V_t characteristics. As integrated circuit technology scales to ever smaller dimensions, supply voltage levels are likewise scaled. To improve circuit speed, the threshold voltages are also decreased. This decrease in threshold voltage results in an exponential increase in the subthreshold leakage current [3]. The International Technology Roadmap for Semiconductors [2] projects dimensions of 70 *nm* to be in production by the year 2006. At these dimensions, the leakage energy is estimated to be *on par* with the dynamic switching energy if novel circuit techniques are not developed [3].

Since most of the transistors in a microprocessor reside in the storage structures (the caches and buffers), the RAMs are responsible for a large portion of the leakage power [7, 9, 11, 14, 20]. The functional units, alternatively, consist of a much smaller fraction of the transistors. However, the model developed by Butts and Sohi [6] for estimating leakage current in various logic structures reveals an order of magnitude larger leakage current for combinational logic relative to cache RAM transistors. While precise estimates for static power require detailed circuit knowledge of the processor, which is not readily available, this model indicates the integer and floating point functional units contribute a noticeable fraction of the overall static power despite the smaller transistor count relative to the caches.

In this paper, we present the benefits of employing a dual threshold voltage domino logic circuit technique [16] (dual-V_t) to reduce subthreshold leakage current in the integer functional units (FUs) of a general-purpose processor. We focus on domino logic dual-V_t circuits because domino logic has both superior speed and area characteristics as compared to static CMOS logic circuits [1, 10, 13, 16]. We restrict the analysis to the integer FUs because it is these units that are most heavily utilized.

Some domino logic designs have a *sleep* mode in which the circuit expends very little static energy. However, due to the energy cost of entering this mode, it has thus far been proven useful only to reduce leakage during long-term standby mode. Idle times in the functional units can often be relatively short, from one to a few hundred cycles. We develop an energy model appropriate for the architecture-level analysis of logic circuits and explore strategies to employ the *sleep* mode in the dual-V_t circuits so as to minimize the overall energy when idle times are short. We use this energy model to develop insight into the dependencies among the application behavior, activation of the idle mode, and the underlying technology characteristics of the circuit.

We study both analytically and empirically (by determining the effects on the performance and energy of a set of integer benchmarks) the benefits and costs of aggressively enabling the sleep mode at every opportunity (*MaxSleep*) relative to never enabling the sleep mode (*AlwaysActive*). These two extreme sleep mode management policies, *MaxSleep* and *AlwaysActive*, are the two simplest policies possible and provide bounds on the energy savings to which other sleep management methods should be compared. Our results show that with idle intervals as short as ten cycles, the *MaxSleep* policy performs well across a broad range of parameters. We also propose a circuit-based scheme we call *GradualSleep* that blends the best behaviors of *MaxSleep* and *AlwaysActive* and reduces the energy impact of using the *sleep* mode for even smaller idle periods. We show that *GradualSleep* performs well across a wide range of conditions. The simple *GradualSleep* design achieves most of the potential energy savings, indicating that more complex control strategies may not be warranted.

The rest of the paper is organized as follows. The low-leakage domino circuit and its behavior is described in Section 2. A static energy model appropriate for architectural energy studies of functional units is developed in Section 3. Our experimental methodology is described in Section 4. The use of the *sleep* mode to reduce overall energy in integer functional units is evaluated in Section 5. Related work is discussed in Section 6. Finally, concluding remarks are made in Section 7.

2 Low-leakage logic-based circuit design

Dynamic domino logic gates are frequently used in critical paths within the functional units of high speed processors. The structures of a static CMOS *AND*-gate with its counterpart implemented as dynamic domino logic are contrasted in Figures 1a and 1b. In static CMOS, the inputs are loaded by both the PMOS and NMOS transistors. In domino logic, the inputs have only the NMOS device as a load and thus are inherently faster. The operation of the domino *AND*-gate is shown in Figure 1c. The internal node *Dynamic* is precharged during the low phase of the clock. Note that the path to ground is cut-off by an NMOS transistor during this time. When the clock transitions high, the path to ground is enabled and the inputs are evaluated. When both inputs are high, the dynamic node is discharged and the output goes high. When either input is low the dynamic node remains charged and the output is low. The state of the dynamic node is preserved against coupling noise, charge sharing, and charge leakage by the keeper transistor. In contrast to static CMOS, every clock cycle the dynamic nodes are precharged and the inputs evaluated regardless of whether the inputs change state. When the circuit is not required, useless re-evaluation (and energy cost) can be avoided by gating the clock such that the clock input is forced high.

As described in [1, 13, 16], domino circuits permit the use of dual threshold voltage techniques to reduce subthreshold leakage current without sacrificing active mode circuit performance. The key to achieving this balance is to place low-V_t transistors only along the critical evaluation path as shown in Figure 2a, in which the shaded transistors are the slower high-V_t devices. The leakage current of a dual-V_t domino circuit is asymmetric and depends on the voltage level at the internal dynamic node. If either *In 1* or *In 2* are low, the dynamic node will remain high. In this state, the voltage across the high leakage transistors *N1*, *N2*, *N3*, as well as *N4* results in a large subthreshold leakage current. Alternatively, when both inputs and the clock are high, the dynamic node is discharged and the low leakage transistors *P1*, *P2*, and *N5* are strongly cutoff. When the dynamic node is discharged, the voltage drop is across the high-V_t devices, which act as high resistance switches, and not across the low-V_t transistors. In this state, the static energy of the circuit is dramatically reduced.

Dual-V_t domino circuits that incorporate a low leakage sleep mode do so by adding the ability to force the internal nodes into the low leakage state. Many circuits incorporating a sleep mode have been proposed [1, 12, 13, 16]. For the purpose of this paper, the essential behavior of all these circuits is similar. The differences are in the complexity and energy overhead of the sleep mode function. For the ensuing discussion, we select a circuit from [16] that is simple and incurs minimal energy overhead.

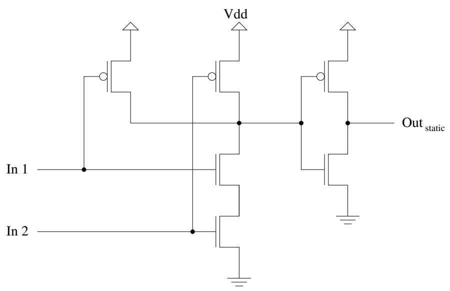
The proposed method for incorporating the low leakage sleep (idle) mode into a dual-V_t domino circuit is shown in Figure 2b. A high-V_t transistor is added to discharge the dynamic node when the *Sleep* signal is asserted, regardless of the input vector. Only the first stage in a sequence of domino circuits requires this additional transistor. Asserting the *Sleep* signal drives the *Out* signal high which turns off the keeper and forces any subsequent domino gates to evaluate to the low leakage state in a domino fashion. Not shown is the standard gating of the clock when *Sleep* is asserted to disable the precharge phase. An important aspect of this design is that the activation energy overhead of the sleep transistor is negligible relative to the switching energy of the gate, $0.14\,fJ$ versus $22.2\,fJ$.

The delay and energy parameters of an 8-input *OR* (*OR8*)

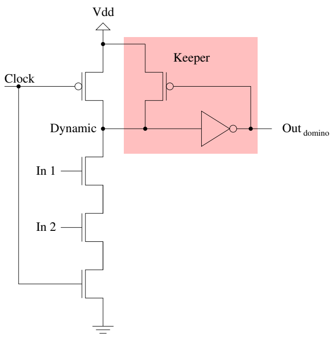
Table 1. ***OR8*** **gate characteristics (70** ***nm*****),** $V_{dd} = 1.0V$**,** $V_{t_n} = 100mV$**,** $V_{t_p} = -120mV$**, Period=250 ps**

	Delay (ps)		Energy (fJ)			
Circuit	Evaluation	Sleep	Dynamic (1 gate)	Vector LO Lkg	Vector HI Lkg	Sleep
low-V_t	19.3	na	26.7	1.2	1.4	na
dual-V_t no sleep mode	15.0	na	22.2	7.1E-4	1.4	na
dual-V_t w/sleep mode	15.0	16.0*	22.2	7.1E-4	7.1E-4*	0.14*

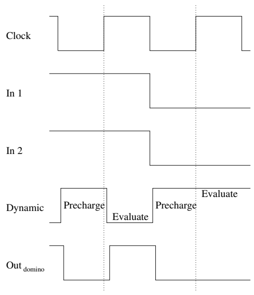
* indicates sleep mode is enabled



(a) Static CMOS *AND*-gate

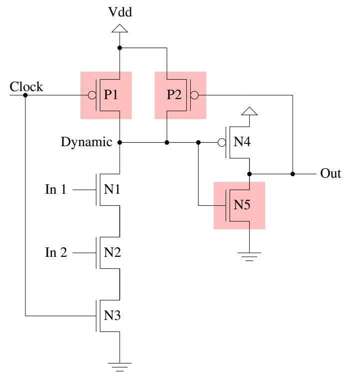


(b) Domino *AND*-gate

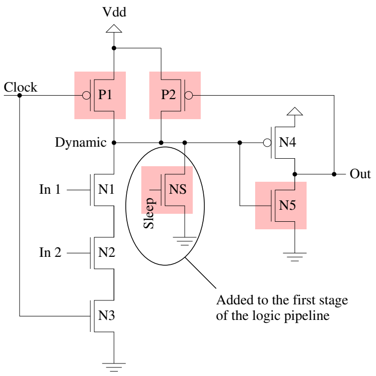


(c) Domino Operation

Figure 1. Static vs. dynamic domino AND**-gates**



(a) Dual-V_t



(b) Dual-V_t with sleep mode

Figure 2. Low leakage domino ***AND*****-gates**

domino gate in 70 *nm* technology is compared in Table 1 for low-V_t, dual-V_t without the sleep mode, and dual-V_t with the sleep transistor. The parameters are $V_{dd} = 1.0V$ and $V_{t_n} = 0.10V$. Since leakage energy in dual-V_t domino circuits depends on the state of the circuit, *Vector LO Lkg* is the input *10000000* which discharges the dynamic node to the low leakage state, and *Vector HI Lkg* is the input *00000000* which does not discharge the dynamic node. The keeper maintains the dynamic node at the high voltage level, which is also the high leakage state.

The lower gate overdrive of the high-V_t keeper transistor in dual-V_t domino circuits reduces the contention current when switching the output and improves the propagation delay and dynamic power characteristics as compared to the low-V_t domino circuit. In the dual-V_t domino circuit, the difference in leakage energy between the *LO Lkg* and *HI Lkg* vectors is a factor of 2,000. Our method of incorporating a sleep mode is not in the evaluation path of the gate so there is no impact on the propagation speed of the circuit. The sleep transistor is minimally sized and introduces negligible additional loading on the dynamic node of a domino gate. With the sleep mode capability, we can force the internal state of all of the gates to the low leakage state, drastically reducing the leakage energy regardless of the input vector. Enabling the sleep transistor, however, requires some additional energy (0.14 *fJ*) which must be accounted for. The delay in discharging the gate via the sleep transistor, 16 *ps*, is comparable to the delay of the evaluation phase, 15 *ps*, so the circuit can transition to the sleep state in one cycle. The measurements assume a 4 GHz clock.

The overhead of enabling the sleep mode depends upon the state of the circuit from the previous evaluation phase. The contributors to the dynamic energy dissipated during an evaluation are the circuits whose input vectors cause the dynamic nodes to discharge. In a complex circuit such as an ALU, on average not all dynamic nodes will discharge during an evaluation cycle. An *activity factor* is the probability that a domino logic gate will evaluate and place the dynamic node into a low voltage state at any given clock cycle. The average activity factor (α), therefore, determines the fraction of the dynamic nodes that are discharged during each evaluation period, $0 \leq \alpha \leq 1$. Activating the sleep switch leaves the circuit in the same state as if the activity factor were 1.0 in the last evaluation; thus, activating the sleep mode discharges the dynamic nodes of the rest of the gates in the circuit. This portion is the $(1 - \alpha)$ fraction of the gates that were not discharged during the previous evaluation period before the sleep mode. To return to the active mode, the clock is again enabled and one precharge phase readies the circuit for evaluation; thus, activation also occurs within a single clock cycle.

2.1 Tradeoffs between active versus sleep modes

Enabling the sleep mode reduces the static energy dissipated, however, this mode is entered by discharging all of the dynamic nodes within the circuit that did not discharge in the evaluation phase. Thus, there is a tradeoff between the energy saved due to lower leakage current and the additional energy expended in the next active cycle from precharging these dynamic nodes that would have remained charged had the sleep mode not been entered. The activity factor α affects both the leakage energy and the energy overhead in transitioning to the sleep mode. As previously mentioned, activating the sleep mode is equivalent to an evaluation with a maximum activity factor of $\alpha = 1.0$. We approximate a generic functional unit (FU) by a circuit consisting of 500 *OR8*-gates arranged as 100 rows of five cascaded domino circuits. The circuit contains the drivers that distribute the *Sleep* signal throughout the FU and this energy is accounted for. The energy expenditure for this circuit relative to the idle interval is shown in Figure 3. The plot compares the effects of enabling the sleep mode versus idling the circuit with clock gating only (the clock is gated

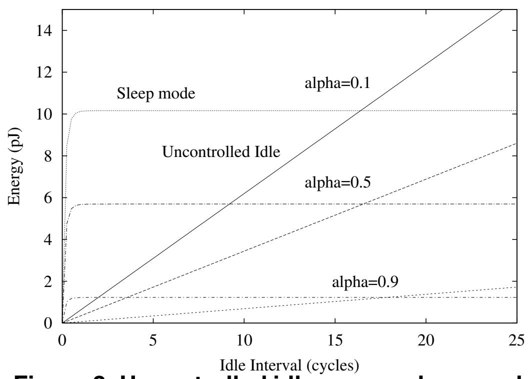


Figure 3. Uncontrolled idle versus sleep mode

high and *Sleep* is not enabled). We refer to this latter case as *uncontrolled idle*. We compare the tradeoffs at three activity levels, $\alpha = \{ 0.1,\ 0.5,\ 0.9 \}$. Results using only an uncontrolled idle (*Sleep* signal not asserted) are the straight lines emanating from the origin. Plots of using the sleep mode rise quickly then plateau.

The graph shows that for a low activity factor there is a considerable expenditure of energy to transition to the sleep mode after which the additional energy is minimal. If the circuit is not idle for at least 17 cycles then more energy is used than is saved by shifting to the low leakage sleep state. This extra energy decreases as the activity factor increases since more nodes enter the low leakage state during the previous evaluation phase before the idle mode. Interestingly, the time to break even is relatively insensitive across this range of activity factor. The reason is that as the activity factor increases, both the sleep transition overhead and the uncontrolled idle circuit leakage energy decrease at a similar rate, roughly proportional to (1 - α).

3 Static energy model

A precise energy model depends heavily on the details of the logic design and the circuit design. General circuit methods to reduce static power include combining high-V_t devices (slow, low leakage) with low-V_t devices (fast, high leakage) and placing the high-V_t transistors along the non-critical paths throughout a functional unit. We develop a simple energy model that is parameterized and can represent the energy characteristics across a wide range of logic and circuit designs at a level useful for architectural studies. The model parameterizes the contribution of the low leakage and high leakage transistors in the overall energy dissipation of the circuit. This parameterization of the fraction of high leakage transistors abstracts the circuit specifics into a single primary parameter that can be varied.

The total energy of a circuit is shown in equation (1). The total energy consists of the dynamic and leakage energy during active cycles plus the leakage energy when the circuit is idle. We divide the total run-time into three categories of operation. The cycles of actual computation are called *active* cycles and their number is denoted as n_A. The cycles when the circuit is clock-gated (no computation) but the sleep mode is not enabled are called *uncontrolled idle* cycles and denoted by n_{UI}. Cycles when the circuit is forced into the low-leakage state of the sleep mode are called *sleep* cycles and denoted by n_Z.

$$\begin{aligned} E'_{total} &= n_A(\alpha E_A + (1-D)E_{S_1}) \\ &\quad + (n_A D + n_{UI})\,(\alpha E_{S_0} + (1-\alpha)E_{S_1}) \\ &\quad + M_Z\,((1-\alpha)\,E_A + E_{Sleep}) \\ &\quad + n_Z\,E_{S_0} \end{aligned} \tag{1}$$

The dynamic energy is the number of active cycles n_A times the maximum dynamic energy per cycle, E_A, prorated by an activity factor α, which is the fraction of the internal dynamic nodes that are actually discharged during the evaluation phase. Recall that the dynamic nodes are precharged prior to evaluation. The precharged state is also the high leakage state of the circuit. If the clock has a duty cycle (i.e., fraction of time the clock signal is high) of D, $0 < D < 1$, then the precharge portion is $(1-D)$ of the clock period. The leakage energy of every active cycle is accounted for by prorating the per cycle high leakage energy, $(1-D)E_{S_1}$. Also added is the leakage energy after evaluation. This active mode leakage energy consists of two components. The first energy component is for the fraction of nodes α that are placed into the low-leakage state, E_{S_0} (per cycle leakage energy with the internal dynamic node discharged), in the normal operation of the circuit. The second component is the fraction of nodes $(1-\alpha)$ that are not discharged (internal dynamic node is high) and have a greater leakage energy, E_{S_1}. In the active cycles we account for this energy only for the portion of the clock period when the clock is high, $n_A D$. For uncontrolled idle cycles where gating the clock prevents precharge, we do not prorate n_{UI} by the duty cycle. If the circuit is sometimes placed into the sleep mode, we add the energy expended in transitioning M_Z times into the sleep state. This energy cost is the additional energy to precharge the $(1-\alpha)$ nodes that would not have been discharged if the circuit had not been forced into the sleep mode. The per transition energy is thus $(1-\alpha)E_A$. Also included is the overhead of activating the sleep mode transistors and distributing the sleep signal across the FU, E_{Sleep}. The final term is the static energy while in the sleep state, i.e., *all* internal dynamic nodes have been discharged and the gates dissipate an energy of E_{S_0} for n_Z cycles.

Since we are using circuits based on dual-V_t domino logic, we can simplify (1). In dual-V_t circuits, the static energy E_{S_0} is much less than E_{S_1} [16]. We define a relationship between the two as $E_{S_0} = s \times E_{S_1}$ where s is typically in the range of $0.0001 \leq s \leq 0.01$. Furthermore, for a given technology, we can define the leakage energy as a fraction of the dynamic energy for a device, $E_{S_1} = pE_A$ where $0 \leq p$. To elaborate, for a single gate the factor p is the ratio of the maximum leakage energy expended to the maximum energy for evaluation per time unit (1 cycle). For circuits in a 130 *nm* technology, the value will be small, $p < 0.01$. This leakage factor p is a versatile parameter. Functional units may be designed using all domino logic or a mix of dynamic domino logic and static logic. In the latter case where there is a mix of low-V_t devices along the critical paths and high-V_t devices along the non-critical paths, we can consider the circuit as a whole and use the ratio of its leakage energy to its evaluation energy as the factor p. This value of p is lower for a single low-V_t gate but greater for a high-V_t gate. Thus, the factor p abstracts the details of the circuit into a single value that models the worst-case leakage behavior relative to the dynamic energy E_A. The factor p becomes a key parameter that we vary to explore the technology design space. Applying the above relationships results in equation (2).

$$\begin{aligned} E'_{total} &= n_A(\alpha E_A + (1-D)pE_A) \\ &\quad + (n_A D + n_{UI})\,(\alpha spE_A + (1-\alpha)pE_A) \\ &\quad + M_Z\,((1-\alpha)\,E_A + E_{Sleep}) \\ &\quad + n_Z\,spE_A \end{aligned} \tag{2}$$

In this architectural study we focus on the relative energy between control policies. We can further simplify (2) by normalizing to the active energy E_A as in (3).

$$\begin{aligned} E_{total} &= n_A(\alpha + (1-D)p) \\ &\quad + (n_A D + n_{UI})\,(\alpha sp + (1-\alpha)p) \\ &\quad + M_Z\,((1-\alpha) + \frac{E_{Sleep}}{E_A}) \\ &\quad + n_Z\,sp \end{aligned} \tag{3}$$

Equation (3) represents the total energy of a circuit in terms of three factors: the technology, the control policy, and the application. The technology defined parameters are p, s, E_{Sleep}, and E_A. Together, the control policy and application determine the active, uncontrolled idle, and idle times n_A, n_{UI}, and n_Z, respectively. The application determines the activity factor α.

To give perspective to the magnitude of the technology variables, we calculate the values for the circuit characterization described in Section 2 from the data listed in Table 1. The maximum dynamic energy, E_A, is 22.2 *fJ*. The ratio $\frac{E_{Sleep}}{E_A}$ is 0.006. The ratio s of the static energy per cycle in the low leakage state to that in the high leakage state is $s = 0.0005$. The ratio of per cycle leakage energy to the dynamic switching energy is the leakage factor $p = \frac{1.4}{22.2} = 0.062$. Since (3) models the energy relative to E_A, the relatively small values of the other factors means the leakage factor p has the greatest impact. We note here that from a similar circuit characterization by Heo and Asanovic [10] we estimate from the data in the paper that their implementation of a Hans-Carlson adder circuit in 70 *nm* technology has a leakage factor that is comparable to our result, between $0.036 \leq p \leq 0.056$.

3.1 Analysis

The analytical model permits quick exploration of the parameter space to find interesting regions that might not be evident from simulating individual data points. We choose values for s and E_{Sleep} that are in agreement with the circuit measurements but somewhat pessimistic (higher). Specifically, we set $s = 0.001$ and $E_{Sleep} = 0.01\ E_A$. We vary the leakage factor p from $0 \leq p \leq 1$ to cover a broad range of technology points that include relatively extreme points in terms of the energy contribution caused by subthreshold leakage current. In some of the results we select specific values for p. In these cases, we restrict p to be either 0.05 (motivated by the values calculated from the circuit characterization) or 0.50 (a convenient number to demonstrate contrasting behavior). These two technology points act as representatives for two distinct behavior regions that, as we shall see, require very different methods for reducing leakage energy. In the rest of this paper, we assume a fixed clock duty cycle of 50% ($D = 0.50$).

Breakeven idle interval. The break even idle interval is the length of time that a circuit must be idle in order that the energy saved in the sleep mode offsets the additional energy required for the transition. Let us parameterize E_{total} from (3) as $E_{total}(n_A, n_{UI}, n_Z, M_Z, \alpha, p)$ and calculate the break even point for a single idle interval. Thus, the break even interval n_{idle} is the interval that satisfies the following relationship:

$$E_{total}(1, n_{idle}, 0, 0, \alpha, p) = E_{total}(1, 0, n_{idle}, 1, \alpha, p) \qquad (4)$$

$$n_{idle} = \frac{(1 - \alpha + \frac{E_{Sleep}}{E_A})}{p(1 - \alpha - s + \alpha s)} \qquad (5)$$

In (4), the left-hand side represents the energy if the circuit is not placed into the sleep mode, $n_Z = M_Z = 0$, and is left as uncontrolled idle, $n_{UI} = n_{idle}$ cycles. The right-hand side of (4) is the energy required for a single transition to the sleep mode, $M_Z = 1$ and $n_Z = n_{idle}$, assuming no uncontrolled idle cycles, $n_{UI} = 0$. We omit the simple algebraic manipulations and give the solution for n_{idle} in (5) and a graph of (5) is shown in Figure 4a (the curves for $\alpha = 0.1$ and $\alpha = 0.5$ are almost identical at this scale). The vertical line at $p = 0.05$ indicates where the near-term technology point lies. The plot delineates the break even intervals across a range of leakage factors, p, for three activity levels, $\alpha = \{0.1, 0.5, 0.9\}$. From the graph it is apparent that as leakage becomes a larger component of the energy, the break even interval decreases, approximately as $\frac{1}{p}$.

Modeling control strategies of the sleep mode. An advantage of a mathematical model is that a model permits exploration of the parameter space before any simulations are run. For this section, we explore three basic methods for controlling the *Sleep* signal. These methods are distinguished by being easily modeled and defining the boundary cases of managing the sleep mode. The first method, *AlwaysActive*, represents the case of doing nothing other than clock gating. We never enable the sleep mode so all idle cycles are *uncontrolled idle* cycles and the circuit expends greater leakage energy. The second method, *MaxSleep*, aggressively enables the sleep mode whenever the circuit has no useful calculation to perform in the following cycle. The *MaxSleep* method incurs the maximum energy transition overhead. The third method, *NoOverhead*, provides an upper bound on energy savings. This method is the same as *MaxSleep* but we omit the energy overhead for transitioning into the sleep mode. Thus, the *No Overhead* strategy represents an unachievable lower bound on total energy and, therefore, is an upper bound on energy savings for any control method. Formally, the energies for each of the strategies are defined in (6)-(8). E_{max} of (9) is the maximum dynamic energy that the circuit can expend by performing a calculation on every cycle assuming an activity factor α, and N is the total number of cycles for the simulation. We normalize the graphs to E_{max} as a useful baseline for the magnitude of the energy differences. Here, we are exploring how the relative energy costs vary across the parameter space.

$$E_{AlwaysActive} = E_{total}(n_A, n_{UI}, 0, 0, \alpha, p) \qquad (6)$$

$$E_{MaxSleep} = E_{total}(n_A, 0, n_Z, M_Z, \alpha, p) \qquad (7)$$

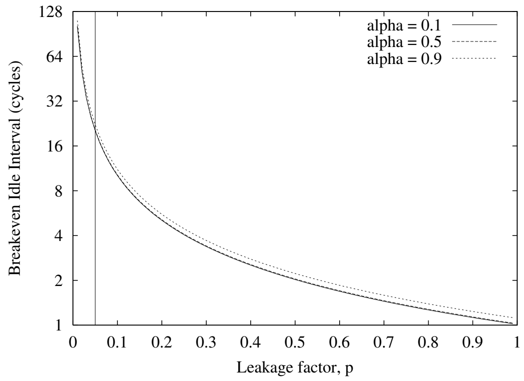
$$E_{NoOverhead} = E_{total}(n_A, 0, n_Z, 0, \alpha, p) \qquad (8)$$

$$E_{max} = E_{total}(N, 0, 0, 0, \alpha, p) \qquad (9)$$

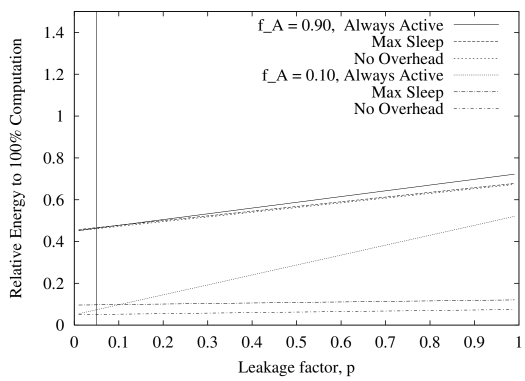
To limit the degrees of freedom, we link the four parameters n_A, n_{UI}, n_Z, and M_Z by a single parameter called a *usage factor (f_A)*. We define this relationship as follows. Assuming a simulation with a total of N cycles, we define $n_A = f_A N$, $0 \leq f_A \leq 1$. Since $N = n_A + n_{UI} + n_Z$, for the *AlwaysActive* policy in which we do nothing and all idle cycles are *uncontrolled idle* cycles (there are no sleep cycles), $n_{UI} = (1 - f_A)N$ and $n_Z = 0$. Conversely, in the *MaxSleep* policy, all idle cycles are sleep cycles (there are no *uncontrolled idle* cycles) so $n_{UI} = 0$ and $n_Z = (1 - f_A)N$. We also define M_Z as a function of n_A, n_Z, and $\bar{n}_{idle}$, the average idle interval duration. Recall that M_Z is the number of times the circuit is placed into the sleep mode and determines how often the transition energy overhead is incurred. For a given average interval length $\bar{n}_{idle}$, the number of transitions to the sleep mode (in the *MaxSleep* policy) is $M_Z = min(\frac{n_Z}{\bar{n}_{idle}}, n_A)$ or, equivalently, $M_Z = min(\frac{(1-f_A)N}{\bar{n}_{idle}}, f_A N)$. The min function is necessary because we must limit the number of transitions to be no greater than the number of active cycles. This restriction ensures that every transition into the sleep mode implies at least one prior active cycle. The energy for the *NoOverhead* method is the same as for *MaxSleep* if $M_Z = 0$. The base energy E_{max} is the energy of the functional unit if during every cycle the unit performs a calculation, thus, $n_A = N$ and $n_{UI} = n_Z = M_Z = 0$.

The total energy for the three control strategies versus the leakage factor p, for a fixed activity factor $\alpha = 0.5$, and normalized to E_{max} is plotted in Figure 4b with the idle interval $\bar{n}_{idle} = 10$ cycles. The bottom three lines are for $f_A = 0.10$. The top three lines are for $f_A = 0.90$. The plots for *MaxSleep* and *NoOverhead* lie almost on top of each other. A similar plot is shown in Figure 4c but with $\bar{n}_{idle} = 100$ cycles. Together, these plots show behavior at extremes of the usage factor and idle intervals (100 cycles happens to be a long idle interval).

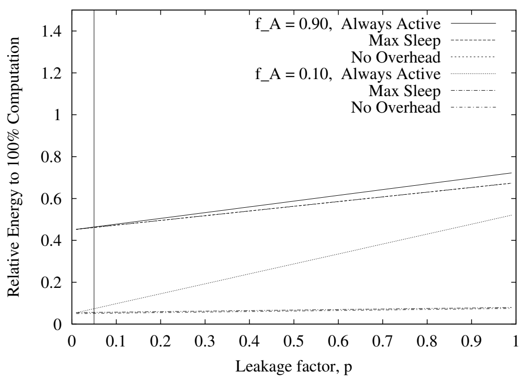
In Figure 4b, the lower grouping of three lines is for a 10% usage factor. The lowest energy line is the *NoOverhead* policy. The slope is relatively flat since 90% of the time the circuit is in the low leakage sleep mode. The *AlwaysActive* line shows a sharp rise as the leakage factor increases. The line for the *MaxSleep* policy runs parallel to that of the *NoOverhead* policy. The difference between the



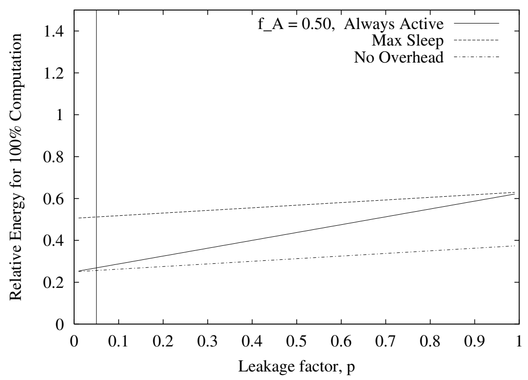
(a) Break even Idle Interval



(b) Idle Interval=10 cycles



(c) Idle Interval=100 cycles



(d) Worst-case: Idle Interval=1 cycle

Figure 4. Exploring the parameter space ot the model

two lines is the energy overhead to place the circuit into the low leakage state when the *Sleep* signal is enabled. At small values of p (low-leakage), the *MaxSleep* policy uses more energy than the *AlwaysActive* policy when the break even interval is greater than 10 cycles (see Figure 4a).

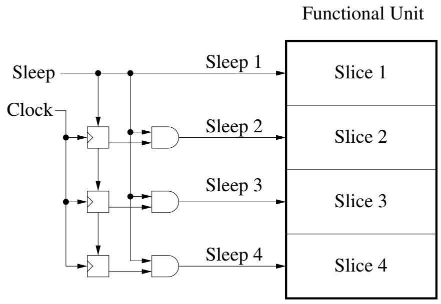
The relative behavior of the policies at the 90% usage factor is similar but the differences are compressed. Since all three policies have identical energy in the active phase, which accounts for 90% of the time, differentiation between the policies can occur only in the remaining 10% of the cycles. Figure 4c is a similar plot with $\bar{n}_{idle} = 100$ cycles. With the larger idle interval the *MaxSleep* policy is nearly identical to the *No Overhead* policy at the 10% usage level. The difference between Figures 4b and 4c is that in the latter figure the transition energy is amortized over 100 cycles as compared to only 10 cycles. The worst case at the 50% usage level is shown in Figure 4d where $\bar{n}_{idle} = 1$ cycle, meaning the circuit alternates between one active and one sleep cycle to incur the maximum transition overhead.

3.2 The *GradualSleep* design

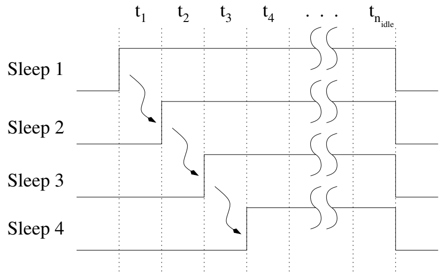
The brief exploration of the energy model space in Section 3.1 showed that the preferred policy for managing the sleep mode depends on parameters for the technology (p) and the control policy/application behavior (embodied by f_A and $\bar{n}_{idle}$ in the discussion). The *MaxSleep* policy works well if the average idle interval is longer than the break even interval, $n_{BE} < \bar{n}_{idle}$, but the *AlwaysActive* policy performs better when the idle interval is shorter, $\bar{n}_{idle} < n_{BE}$. A policy that selects the minimum energy between the two options, $min(E_{MaxSleep}, E_{AlwaysActive})$, is the best combination of the two policies.

Here we propose a method that is a hybrid of the *MaxSleep* and *AlwaysActive* control schemes. By dividing the circuit into slices and staggering the *Sleep* enable signal, we can incrementally place the circuit into the sleep mode and avoid the initial energy dissipation in the first idle cycle as in the *MaxSleep* policy. This method also protects against excessive static energy consumption that the *AlwaysActive* policy would incur in the event of a long idle interval. A block diagram of a circuit divided into four slices is shown in Figure 5a. The timing of the *Sleep* signal is shown in Figure 5b. The *Sleep* signal feeds one end of a shift register whose bits supply the *Sleep* signal to the different slices of the circuit. The AND gates ensure simultaneous re-activation of the circuit. All of the register bits are simultaneously cleared upon de-assertion of the *Sleep* signal. While any level of granularity can be used, we assume the number of slices matches the number of cycles in the break even interval for the technology, n_{BE}, so that every cycle $\frac{1}{n_{BE}}$ of the circuit enters the sleep mode. Using fewer slices changes the curve for *GradualSleep* to be more similar to the *MaxSleep* behavior. Adding more slices results in a shift towards the *Always Active* behavior. We hide assertion/deassertion of the *Sleep* signal behind the register read stage of the pipeline. The basic pipeline of the Alpha 21264 [15] is shown in Figure 6, as is a single, generic *Sleep* signal to one of the FUs. At the end of the *issue* stage the number of integer instructions to be executed is known and the appropriate FUs are activated before the instructions reach the *execute* stage. Since the *Sleep* signal is staged and is not along a critical path, the shift register and AND gates can be constructed from slower, high V_t transistors with very low subthreshold leakage current. We do not include the small additional dynamic energy in the analysis.

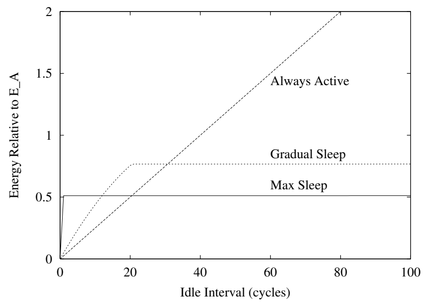
The energy costs of transitioning to the sleep mode for the three policies is compared in Figure 5c. We set $p = 0.05$ for reasons discussed in Section 3.1 and arbitrarily set $\alpha = 0.50$ and the usage factor $f_A = 0.5$. The relative shape of the curves is consistent regardless of the parameter values. The *GradualSleep* policy saves energy over the *MaxSleep* policy when the idle interval is short and is better than the *AlwaysActive* policy when the interval is long. Near the break even point the *GradualSleep* policy expends more energy than the other two policies. The *GradualSleep* design acts as a hedge against the pathological case of short alternating active and idle intervals as highlighted in Figure 4b of Section 3.1. The results described in Section 5 show that the *GradualSleep* policy successfully avoids the extremes of the other two policies.



(a) Block diagram



(b) *GradualSleep* signal timing



(c) Energy to transition to the sleep mode

Figure 5. The *GradualSleep* design

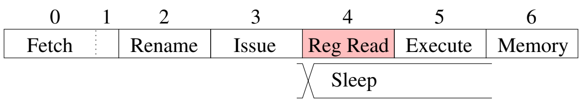


Figure 6. The *Sleep* signal timing

4 Experimental methodology

We use the Simplescalar simulator [5] to verify the preceding analytical analysis in Section 3. The processor is modeled after the Alpha 21264 and the configuration parameters are given in Table 2. We have modified the simulator to have individual structures for the reorder buffer, integer queue, floating point queue, and load store queue as in the Alpha 21264. We restrict the study to the integer units since integer operations are generally the dominant type of instructions executed, thus, these functional units are heavily utilized. The integer benchmarks are listed in Table 3.

The goal of this study is to explore the potential for improving energy efficiency with fine-grained control of static energy in large logic circuits. To ensure the results are not inflated by excess resources that can be trivially put to sleep, we limit the number of functional units. Our processor configuration supports a maximum of up to four integer functional units. For each application, we determine the minimum number of functional units required to achieve at least 95% of the peak performance from using four functional units. Restricting the number of functional units makes it more difficult for a control method to successfully exploit the sleep mode and, thus, makes differences between control methods more meaningful. Implicit in this methodology is the assumption that some technique of profiling [19] or compiler analysis [18] can be used to identify when functional units are not needed *a priori*. Such an analysis could be used to signal the run-time system that some functional units are unnecessary and can be disabled at the start of an application. The second to last column in Table 3 shows the number of integer units used for each benchmark in all of the simulations. The fourth column lists the maximum IPC with four functional units, while the fifth column lists the achieved IPC for a given number of functional units.

Table 2. Architectural Parameters

Fetch queue	8 entries
Branch predictor	comb. of bimodal and 2-level gshare; bimodal/Gshare Level 1/2 entries- 2048, 1024 (hist. 10), 4096 (global), resp.; Combining pred. entries - 1024; RAS entries - 32; BTB - 4096 sets, 2-way
Branch mispred. latency	10 cycles
Fetch, decode, issue width	4 instructions
Reorder buffer	128 entries
Integer issue	32 entries
Floating point issue	32 entries
Physical integer regs	96 entries
Physical floating point regs	96 entries
Load entries	32 entries
Store entries	32 entries
Instruction TLB	256 entry 4-way, 8K pages, 30 cycle miss
Data TLB	512 entry 4-way, 8K pages, 30 cycle miss
Memory latency	80 cycles
L1 I-cache	64 KB, 4-way, 64B line, 2 cycle
L1 D-cache	64 KB, 4-way, 64B line, 2 cycle
L2 unified	2 MB 8-way, 128B line, 12 cycle

Table 3. Benchmarks

App	Suite	Instr. Window	Max IPC	IPC	FUs
health	Olden	80M-140M	0.560	0.554	2
mst	Olden	entire pgm 14M	1.748	1.748	4
gcc	SPEC95 INT	1650M-1750M	1.622	1.619	2
gzip	SPEC2K INT	2000M-2050M	2.120	2.120	4
mcf	SPEC2K INT	1000M-1050M	0.523	0.503	2
parser	SPEC2K INT	2000M-2100M	1.692	1.692	4
twolf	SPEC2K INT	1000M-1100M	1.542	1.475	3
vortex	SPEC2K INT	2000M-2100M	2.387	2.387	4
vpr	SPEC2K INT	2000M-2100M	1.481	1.431	3

In the simulations, we allocate operations to the set of functional units in round robin fashion and record precise statistics on the idle times for each functional unit. From this data, we calculate the total energy used by each functional unit by summing the energies for active cycles, uncontrolled idle cycles, and sleep mode cycles as given in equation (3). The total energy of the integer unit is the sum of the energies of the individual functional units. Values of the equation parameters are listed in Table 4.

We present results for three values of the activity factor, $\alpha = \{0.25, 0.50, 0.75\}$. Since values in the integer units are dominated by either zeros or ones [4], we expect the final state after evaluation of the domino gates in the functional units to also be biased to either the high leakage state or the low leakage state depending on the bias. A low activity factor ($\alpha < 0.50$) corresponds to a bias of the input values that leaves the majority of the domino gates in the high leakage state. Conversely, a high activity factor ($\alpha > 0.50$) sets the majority of gates to the low leakage state.

5 Results

The distribution of idle intervals across the benchmarks is plotted in Figure 7. The x-axis is a $\log_2$ scale in cycles of the length of the idle interval and the y-axis is the fraction of the total time that the ALUs are idle. The data for each of the functional units from the different applications are combined as fractions to give the data equal weight regardless of the instruction window size of the application. To improve readability, idle intervals longer than 8192 cycles have the

Table 4. Parameter values for energy calculations

Parameter	Value
n_A	Distribution from simulation data
n_{UI}	Distribution from simulation data
n_Z	Distribution from simulation data
M_Z	Distribution from simulation data
α	$\{0.25, 0.50, 0.75\}$
p	$\{0.05, 0.50\}$
s	0.001
$\frac{E_{Sleep}}{E_A}$	0.01

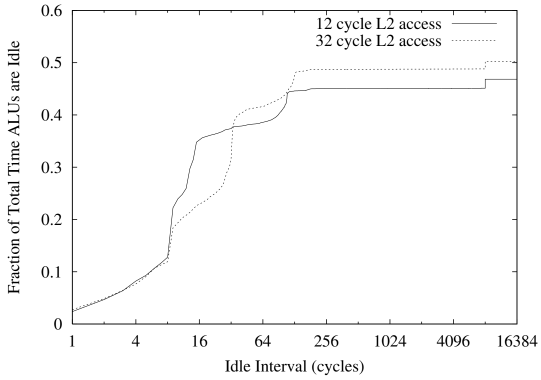


Figure 7. Distribution of idle intervals

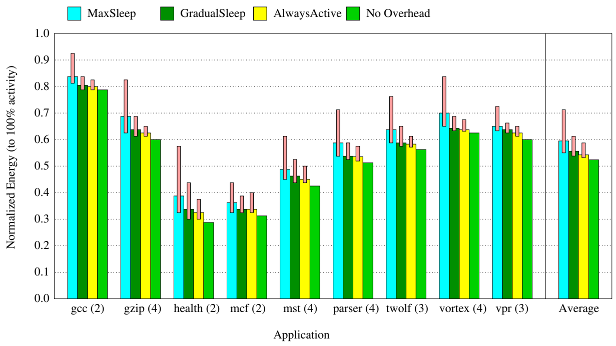
total idle time accumulated at the 8192 cycle marker, hence, the short but sharp step at the right of the graph. The graph shows that across the suite of benchmarks, any given integer ALU is idle 46.8% of the time when the L2 access latency is 12 cycles. Furthermore, nearly all of the idle intervals are shorter than 128 cycles and a large fraction, 75%, occur within the L2 access latency time. To highlight the influence that the L2 access latency has on the distribution, also plotted is the idle interval distribution using a 32 cycle L2 access latency. The increased overall idle time reflects the additional time to access the L2 cache. As demonstrated in Figure 7, extremely large idle intervals are rare and relatively short intervals are common.

The relative energies of the three policies presented in Section 3 are compared in Figure 8. The energy is normalized to the energy that would be expended if the circuit performs a calculation every cycle, i.e., there are no idle cycles (E_{max} of Section 3). The results for a circuit with a subthreshold leakage factor of $p = 0.05$ are shown in Figure 8a. The applications are listed below with the number of functional units. In each grouping of bars, the first bar is the *MaxSleep* policy that enables the *Sleep* signal at a functional unit as soon as there is an idle cycle for that unit. Multiple functional units are managed independently. The second bar is the *GradualSleep* design that incrementally places a circuit into the sleep mode. The third bar in the grouping is the *AlwaysActive* policy which never enables the *Sleep* signal. The fourth bar plots the *NoOverhead* policy which represents in this study an unachievable lower bound for reducing static energy. For each policy, the primary bar represents $\alpha = 0.50$. The small bar at the top delineates the range for $\alpha = 0.25$ (the top) and $\alpha = 0.75$ (the bottom).

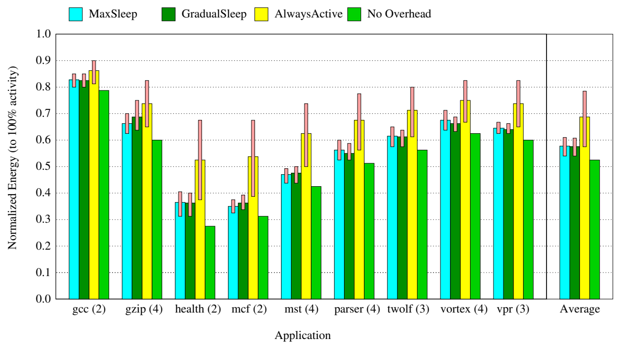
Let us discuss only the primary bars when $\alpha = 0.50$. From the bar chart, when $p = 0.05$, the *MaxSleep* policy always uses more energy than the simpler (i.e., do nothing) *AlwaysActive* policy, 8.3% more on average. The reason is that at the lower p value the breakeven interval to recoup the transition energy is significantly greater than the average idle interval in this set of applications. The *AlwaysActive* policy is within only 5.3% of the energy of the *NoOverhead* method. Thus, at this technology point, there is no need to enable the sleep mode. The *GradualSleep* design uses slightly more energy than the *AlwaysActive* policy, but is within 2.0%. These conclusions hold when $\alpha = 0.25$ ($\alpha = 0.75$) except the differences increase (decrease). Recall that at $\alpha = 0.25$, less of the domino logic gates end up in the low leakage state from the evaluation so transitioning to the sleep mode discharges more energy than when $\alpha = 0.50$. The converse is true for $\alpha = 0.75$.

The results are considerably different when the technology involves high leakage transistors. The same plot is shown but for a relatively high leakage factor $p = 0.50$ in Figure 8b. The greater subthreshold leakage current shortens the breakeven interval (recall Figure 4a) such that the *MaxSleep* policy is always more energy efficient than the *AlwaysActive* policy, saving an average of 19.2% at $\alpha = 0.50$. This savings represents 70.4% of the maximum potential bounded by the *NoOverhead* policy. The remaining 29.6% difference represents the overhead to transition to the sleep mode and can be reduced by decreasing the number of these transitions, possibly with a policy that schedules operations on the functional units. Notice that the *GradualSleep* design performs about as well as the *MaxSleep* policy and even slightly better on three applications, *parser*, *vortex*, and *vpr*. Averaged across the benchmark suite, *GradualSleep* is essentially identical to the *MaxSleep* policy (the difference is negligible). Here, again, the differences increase for $\alpha = 0.25$ and decrease for $\alpha = 0.75$.

The energy of each of the three policies relative to the energy of the *NoOverhead* policy across the range of values $0 \leq p \leq 1$ is plotted in Figure 9a. We do not show the results for the individual benchmarks, only the average. For each data point, we calculate the average of the relative energies for the benchmark suite. This plot shows the relative behavior of each policy across the technology space. The technology points of $p = 0.05$ and $p = 0.50$ used to generate the results illustrated in Figures 8a and 8b are marked on the graph. As described before, when the leakage energy of the circuit is small the *AlwaysActive* policy outperforms the *MaxSleep* policy, but the reverse is true when the leakage energy becomes large. The *GradualSleep* design, however, exhibits well behavior across the complete technology range, and performs better near the breakeven point for the distribution of idle intervals of the benchmarks. Thus, the ability to blend both policies has little negative impact and can actually improve the overall energy efficiency when the distribution of idle times centers around the breakeven point. The fact that the *GradualSleep* design avoids the extreme behaviors of the other two policies means that the



(a) $p = 0.05$



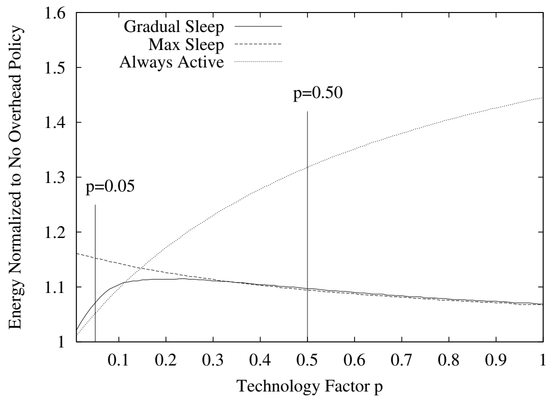
(b) $p = 0.50$

Figure 8. Comparing *MaxSleep*, *GradualSleep*, *AlwaysActive*, and *NoOverhead* policies

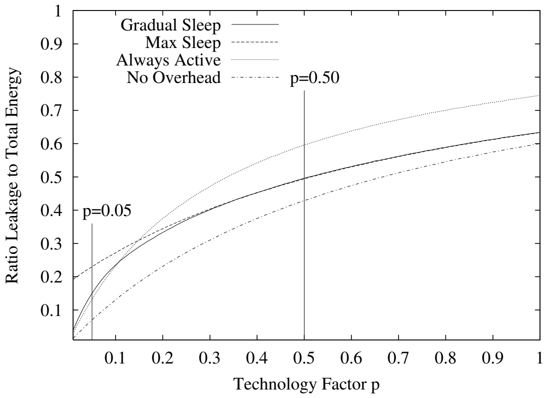
GradualSleep policy will still perform well as a design is scaled in the same circuit technology or implemented in a different technology having a different value for p.

The problem of leakage energy is often reported as the fraction of the total energy due to leakage. This view of the data is plotted in Figure 9b. At $p = 0.05$, the leakage energy is 13% of the total energy for the *AlwaysActive* policy, but increases to 60% at $p = 0.50$.

The results shown in Figure 9b are best appreciated in the context of the processor as a whole. Borkar [3] indicates that at 70 *nm* dimensions and beyond ($p \geq 0.05$, approximately), leakage will comprise 30% or more of the total power. Our results showing only 13% at $p = 0.05$ do not conflict with this conclusion for the following reasons. The primary factor producing the lower than projected fraction of leakage energy is our methodology of eliminating unnecessary functional units that would contribute significantly to leakage but not to dynamic energy. For example, in our simulations *mcf* utilizes only 31% of the two functional units and the fraction of leakage energy is 15%. The fraction increases to 25% for a microarchitecture with four functional units. Second, we do not include the non-interger functional units in our analysis because they are mostly idle in this benchmark suite (and, thus, trivially controlled). In



(a) Average energy relative to *NoOverhead*



(b) Ratio of leakage energy to total energy

Figure 9. Averaged simulation results

the integer benchmarks, these non-integer functional units add disproportionately to the leakage portion of the total energy. This effect would further increase the overall fraction of leakage energy relative to the total energy.

Depicted in Figure 9b is the plot for the *No Overhead* policy. This policy represents a lower bound on the fraction of static energy since all the idle cycles are at the lowest leakage state and there is no additional energy cost to transition to that state. Thus, for this policy, the static energy is almost entirely due to leakage during computation cycles. The active mode leakage energy is a significant fraction of the overall leakage energy, and becomes the dominant fraction as p becomes larger. Circuit techniques are required to reduce this portion of the leakage energy.

6 Related work

Dual-V_t domino logic circuits with a sleep mode have been proposed in [1, 10, 13, 16]. While all of these circuits limit leakage energy by forcing the dynamic nodes into the low leakage state, the overhead of this sleep mechanism varies. We selected the circuit from [16] because the technique has no delay penalty and a low energy overhead. However, the energy model parameters can be adjusted to reflect many other circuit techniques.

Heo and Asanovic [10] introduce the technique of controlling the sleep mode of dual-V_t circuits for fine-grained reduction of leakage energy. The focus is on the circuit itself and ends with an analysis of the breakeven interval for an adder. We extend this work by introducing an analytical energy model for a logic functional unit and perform a detailed study on how to implement fine-grained control of the sleep mode in heavily used functional units of a microprocessor. Our results reveal the interdependencies among the circuit technology, the application, and the control strategy.

Butts and Sohi [6] introduce a static energy model for estimating static power consumption early in the design process at the architectural level. This static energy model can be parameterized to provide *steady-state* estimates of various types of circuits, e.g., RAM cells, CAM cells, and logic gates. To relate this work to our own, the Butts and Sohi model is appropriate for estimating the parameter E_A and the leakage factor p. In contrast, our model is specialized for logic but estimates total energy of the functional units, both dynamic and static, based on the behavior of the application. The ability to consider the dynamic behavior of a circuit is essential in analyzing the tradeoffs between schemes that manage the sleep mode of the circuit.

Rele *et al.* [18] use the compiler to identify when functional units will be idle for long periods of time and can be power gated, thus reducing the static power. The basis of our study presumes a technique such as [18] has already been applied. By limiting the number of functional units, our study explores how to manage resources that are critical to performance and, consequently, have short idle times.

Both Brooks and Martonosi [4] and Ghose *et al.* [8] demonstrate that many operands do not require the full width of the datapath. To save dynamic energy, datapath hardware detects these bytes and gates the logic from performing unnecessary work. In the context of this paper, this phenomenon might be able to be exploited in the *Gradual-Sleep* policy by placing the high order bytes to sleep initially and upon re-activation only activate these bytes that are also enabled by the datapath hardware.

Pyreddy and Tyson [17] use dual speed pipelines to save dynamic energy by scheduling non-critical instructions on the slow pipeline. A slow pipeline could have a higher threshold voltage and lower leakage current. Off-loading the non-critical instructions from the fast pipeline will increase the average idle duration in the fast pipeline. This strategy may offer additional opportunities to enable the sleep mode of the fast pipeline.

At the architectural level, the study of leakage reduction has centered on the storage structures in the microprocessor. Yang *et al.* [20] gate the power supply voltage to the L1 instruction cache RAM cells to turn off power to the storage cells and essentially eliminate the leakage energy. The state of the cell is lost. Kaxiras *et al.* [14] present a control scheme that dynamically adjusts when to place the cache lines into the sleep mode to minimize leakage energy. Flautner *et al.* [7] propose a *drowsy* cache design for the L1 data cache that maintains the cell state in the sleep mode but at the cost of higher leakage energy than if power to the cell were completely turned off. Their study concluded that a simple control scheme sufficed to achieve most of the energy savings. Hanson *et al.* [9] compare these two techniques and a third method in an extensive study that includes the L1 instruction cache, the L1 data cache, and the L2 unified cache. Heo *et al.* [11] take a novel approach to reduce the static energy associated with the bitlines in a RAM by simply tristating the drivers to the lines. The floating bitlines settle naturally at the voltage level that minimizes the leakage energy.

7 Conclusion

In this study, we evaluate the circuit technology of dual-V_t domino logic along with the *sleep* mode as a promising technique for reducing subthreshold leakage energy at a fine-grained time scale, from one to a few hundred cycles. Taking the energy cost of entering the low-leakage *sleep* state into account, we introduce an analytical energy model to characterize the energy behavior of functional logic units at the architectural level. We use this model to characterize the interaction of the application with the technology as well as evaluate the effects on performance and energy of a set of integer benchmarks as technology parameters are varied. We show that the simple *GradualSleep* design works well across a range of technology and application parameters by amortizing the energy cost of entering the *sleep* mode across several cycles. Our results indicate that a more complex control strategy to determine when to enter the *sleep* state may not be warranted and that the leakage energy lost during the *active* cycles of the functional units may eventually become the dominant component of the overall leakage energy.

References

[1] M. W. Allam, M. H. Anis, and M. I. Elmasry. High-Speed Dynamic Logic Styles for Scaled-Down CMOS and MTCMOS Technologies. In *IEEE International Symposium on Low Power Electronics and Design*, July 2000.

[2] S. I. Association. The International Technology Roadmap for Semiconductors. In *http://www.semichips.org*, 2001.

[3] S. Borkar. Design Challenges of Technology Scaling. In *IEEE Micro*, July 1999.

[4] D. Brooks and M. Martonosi. Value-Based Clock Gating and Operation Packing: Dynamic Strategies for Improving Processor Power and Performance. In *ACM Transactions on Computer Systems*, May 2000.

[5] D. Burger and T. Austin. The Simplescalar toolset, version 2.0. Technical Report TR-97-1342, University of Wisconsin-Madison, June 1997.

[6] J. A. Butts and G. S. Sohi. A Static Power Model for Architects. In *33rd Annual International Symposium on Microarchitecture*, December 2000.

[7] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *29th Annual International Symposium on Computer Architecture*, May 2002.

[8] K. Ghose, D. Ponomarev, G. Kucuk, A. Flinders, P. M. Kogge, and N. Toomarian. Exploiting Bit-Slice Inactivities for Reducing Energy Requirements of Superscalar Processors. In *Kool Chips Workshop, MICRO-33*, 2000.

[9] H. Hanson, M. S. Hrishikesh, V. Agarwal, S. W. Keckler, and D. Burger. Static Energy Reduction Techniques for Microprocessor Caches. In *2001 International Conference on Computer Design*, September 2001.

[10] S. Heo and K. Asanovic. Leakage-Biased Domino Circuits for Dynamic Fine-Grain Leakage Reduction. In *Symposium on VLSI Circuits*, June 2002.

[11] S. Heo, K. Barr, M. Hampton, and K. Asanovic. Dynamic Fine-Grain Leakage Reduction Using Leakage-Biased Bitlines. In *29th Annual International Symposium for Computer Architecture*, May 2002.

[12] S. Jung, S. Yoo, K. Kim, and S. Kang. Skew-Tolerant High-Speed (STHS) Domino Logic. In *IEEE International Symposium on Circuits and Systems*, May 2001.

[13] J. Kao and A. Chandrakasan. Dual-Threshold Voltage Techniques for Low-Power Digital Circuits. In *IEEE Journal of Solid-State Circuits*, volume 35, July 2000.

[14] S. Kaxiras, Z. Hu, and M. Martonosi. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In *28th Annual International Symposium on Computer Architecture*, June 2001.

[15] R. E. Kessler. The Alpha 21264 Microprocessor. In *IEEE Micro*, April 1999.

[16] V. Kursun and E. G. Friedman. Low Swing Dual Threshold Voltage Domino Logic. In *12th Great Lakes Symposium on VLSI*, April 2002.

[17] R. Pyreddy and G. Tyson. Evaluating Design Tradeoffs in Dual Speed Pipelines. In *Workshop on Complexity-Effective Design in conjunction with ISCA 2001*, June 2001.

[18] S. Rele, S. Pande, S. Onder, and R. Gupta. Optimizing Static Power Dissipation by Functional Units in Superscalar Processors. In *International Conference on Compiler Construction*, April 2002.

[19] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott. Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling. In *Eighth International Symposium on High Performance Computer Architecture*, February 2002.

[20] S.-H. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. N. Vijaykumar. An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep-Submicron High Performance I-Caches. In *Seventh International Symposium on High-Performance Computer Architecture*, January 2001.