# Beyond Memory Cells for Leakage and Temperature Control in SRAM-based Units, The Peripheral Circuits Story

Houman Homayoun, *member IEEE*, Avesta Sasan, *member IEEE*, Aseem Gupta, *member IEEE*, Alex Veidenbaum, *member IEEE*, Jean-Luc Gaudiot, *Fellow IEEE*, Fadi Kurdahi, *Fellow IEEE*, Nikil Dutt, *Fellow IEEE*

## ABSTRACT

Leakage currents in on-chip SRAMs: caches, branch predictor, register files and TLBs, are major contributors to the energy dissipated by processors in deep sub-micron technologies. High leakage also increases chip temperature and some SRAM-based structures become thermal hotspots. Previous work has addressed major sources of SRAM leakage in memory cells and bit-lines, making remaining SRAM components, in particular *large drivers*, the primary source of leakage. This paper proposes an approach to reduce this source of leakage in all major SRAM-based units of the processor, controlling them in a uniform way, yet treating each unit individually based on its behavior and memory organization. The new approach uses multiple bias voltages in sleep transistors allowing a trade-off between leakage reduction and wakeup delay in multi-stage peripheral drivers. Four low-power modes are defined, from basic to ultra low power, and SRAMs dynamically transition between these modes to minimize leakage without sacrificing performance. A novel control mechanism monitors and predicts future processor behavior for mode control. The leakage reduction in individual units is evaluated and shown to vary from 38% for IL1 to 75% for L2 caches. Resulting temperature reduction, including the effect of positive feedback between temperature and leakage power, is evaluated. A significant temperature reduction is achieved in each unit. It is also shown to reduce hot spots in the instruction TLB and branch predictor.

## Categories and Subject Descriptors

B.3.2 [**MEMORY STRUCTURES**], Design Styles: Cache memories; C.1.1 [**PROCESSOR ARCHITECTURES**], Single Data Stream Architectures: Pipeline processors Systems.

**General Terms**: Design.

## Keywords

SRAM Memory, Leakage Power, Peripheral Circuits, Multiple Sleep Mode, Temperature Reduction.

## 1. INTRODUCTION

Leakage energy dissipation has become the dominant component of the total energy dissipation in deep sub-micron technologies. On-chip SRAM memories such as caches, branch predictor, and TLBs account for a large fraction of total processor power consumption and much of it is leakage power because of their large size. High leakage power dissipation not only increases the overall processor power dissipation but also increases its temperature. The *positive feedback loop between temperature and leakage power* causes a further increase in both of them [27, 42]. Furthermore, some of the SRAM-based structures are temperature hot-spots on a chip, e.g. register files, BTB, and ITLB [28]. Finally, higher temperature reduces chip reliability and usable lifetime and increases the complexity of packaging and cooling design.

A number of process and circuit techniques have been proposed to significantly reduce the leakage of the memory cell array in *SRAMs*. Recent results have shown that leakage in *SRAM peripheral circuits*, such as word-line, input and output drivers, etc. *are now the main sources* of leakage [2, 3, 8, 12, 17, 37]. For instance, a wordline driver drives its signal to a large number of memory cells. Given such a high capacitive load a chain of inverter buffers of increasing size is used, typically with three to five levels. We compared the leakage power consumption of a 45nm SRAM6T memory cell [1]with an inverter of different sizes. The results are shown in Figure 1(a). It shows that the leakage power of a standard memory cell is significantly lower than the leakage power of inverter buffers and that the inverter leakage grows exponentially with its size.

For instance let us assume that a driver has to drive 256 one-bit memory cells. This will require three stages of inverter buffers (of increasing size, by a factor of *e*). The combined leakage power of these three drivers is 12 times larger than the leakage of the 256 memory cells. In addition to the wordline driver one has to consider leakage in data input and output drivers which are also high. Such a large leakage power in peripheral circuits of SRAM memories has been analyzed and discussed in detail in [2]. In brief two main reasons explain this difference in leakage:

- Memory cells are designed with minimal sized transistor mainly for area considerations. Unlike memory cells, peripheral circuits use larger, faster and accordingly more leaky transistors in order to satisfy timing requirements.
- Memory cells use high threshold voltage transistors which have a significantly lower leakage reduction compared with typical threshold voltage transistors used in peripheral circuit

In summary, SRAM memory cells are optimized for low leakage current and area without a significant impact on the cell performance [2, 3, 8, 12, 41]. In addition, circuit techniques such as gated-vdd and drowsy cache can be applied to further reduce the memory cell leakage and widen the gap between cell array and peripheral leakage power dissipation.

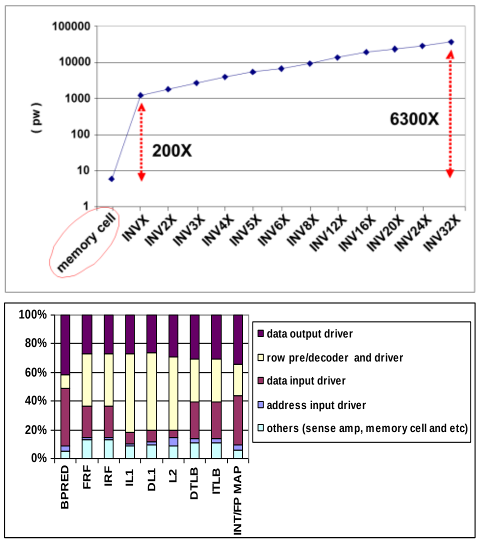


**Figure 1. (a) Leakage power dissipation of one SRAM6T memory cell compared with different size inverter buffers (INVX is the smallest inverter buffer with drive strength of 1) (b) Leakage power in on-chip SRAMs.**

A similar result is obtained using CACTI 5.1 [22]. CACTI uses characteristics of transistors modeled by the ITRS [43]. It includes

[1] Similar results were obtained for TSMC, TOSHIBA, IBM and UMC foundries using their libraries and evaluating leakage with Spice.

data for two device types that the ITRS defines - High Performance (HP) and Low Standby Power (LSTP). The HP transistors are fast transistors with short gate lengths, thin gate oxides, low $V_{th}$, and low VDD. The LSTP transistors, on the other hand, have longer gate lengths, thicker gate oxides, higher $V_{th}$, and higher VDD. As explained in [2] HP transistors are used in the peripheral circuitry while the LSTP transistors are used in the memory cells array. While it is possible to use LSTP transistors in peripheral circuits for reducing leakage the impact on memory access time would be significant (for instance an increase from 3.8ns to 12.5ns access delay for a 2MB L2 cache). CACTI results show that the peripheral circuitry are leaking considerably more than memory cell array.

Figure 1(b) shows the leakage for different components of various on-chip SRAMs, such as branch predictor, TLBs, L1 and L2 caches, register files and register map table for 45nm technology (based on CACTI 5.1 [22]). It demonstrates that the peripheral circuits – data drivers, address driver, decoder, and wordline drivers – account for over 80% of the overall SRAM leakage.

Figure 2 presents results of detailed thermal modeling and temperature estimation of on-chip cache memories. The figure shows the thermal image of a large 2MB L2 cache and a small 128KB L1 cache. We integrated CACTI 6.5 with Hotspot 5.0 to generate this thermal image. The area data for memory block height/width, bank height/width, mat height/width and etc. were extracted from CACTI and used to generate a detailed floorplan of the SRAM cache. Also the leakage and dynamic power of the cache components, including the data in/out horizontal and vertical h-tree, address horizontal/vertical h-tree, and predecoder driver were extracted from CACTI. As seen in Figure 2, peripheral circuits occupy a relatively small part of the cache area. Most of the area is occupied by memory banks. The steady state temperature of various blocks of the cache was calculated using Hotspot . The steady state temperature is estimated assuming that accesses are randomly distributed among memory banks. The thermal images show a large thermal gradient across an entire cache. The hottest units (in red) are the peripheral circuits both in the L1 and the L2 cache. To better understand such a large thermal difference across the L2, consider the power breakdown in Figure 1(b) and the area breakdown of the L1 and L2 caches in Figure 2. In fact, a large thermal difference between peripherals and memory banks is due to their large power density differences. Peripherals have a noticeably higher power density than the memory banks and as a result they have a higher temperature. In addition, any access to L1 and L2 caches has to go through peripherals while accesses to memory cells are distributed across all banks. In other words, the activity factor of peripherals is higher than the activity factor of a single memory bank. The thermal variation as shown in the figure is as large as 12 degrees (comparing P0 temperature vs. B0) for both the L1 and L2 caches.This paper proposes a combination of circuit and architectural techniques to minimize leakage power dissipation and consequently also reduce the temperature of on-chip SRAMs. Unlike some of the previous work, it targets *all* large on-chip SRAMs simultaneously: branch predictor, register files, L1 data and instruction cache (DL1 and IL1), L2 cache (L2), instruction and data translation lookaside buffers (ITLB/DTLB), and register map tables. It focuses on leakage in *SRAM peripheral* circuits. Some of the on-chip SRAMs are small, such as register files, while others are very large, such as the L2 cache. Reducing leakage in small SRAMs has a minor impact on overall chip leakage compared to large SRAMs, but our leakage reduction approach leads to significant temperature reduction in these small SRAMs. For large SRAMs, it provides both leakage and temperature reduction.

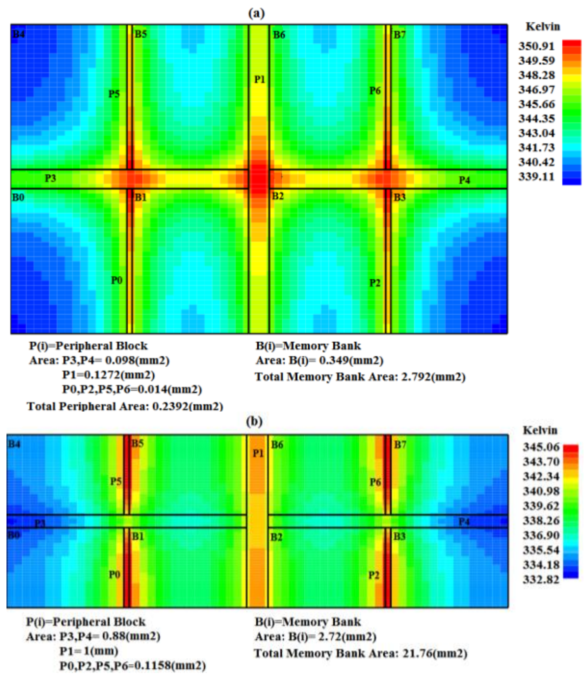


**Figure 2. Thermal variation within a (a) 128 KB 8-way set associative, 64b readout bits L1 cache and (b) 2MB 8-way set associative 256b readout bits L2 cache.**

Reducing leakage in units such as branch predictor, register files and TLBs is also important because these small SRAMs are among the hottest units due to their small size and high power density [28]. Any leakage reduction in these units is further increased due to the positive feedback loop between temperature and leakage. At the circuit level, one approach to reduce the sub-threshold leakage in SRAM *peripheral circuits* is to use stacked sleep transistors [3, 10, 40]. The drawback of using sleep transistors is the time delay that they add to SRAM access time, which may lead to increased execution time and therefore potentially higher energy consumption. To reduce SRAM "wakeup" delay [3] proposed *sharing sleep transistors* and using them in a *zig-zag*, or alternating, fashion across stages of multi-stage drivers, such as the SRAM word-line driver.

This paper shows that by increasing the bias voltage of the NMOS footer sleep transistor in *zig-zag share* circuit (and decreasing it for the PMOS header transistor) one can trade leakage reduction and wakeup delay in the zig-zag share scheme. It proposes to use several low-leakage power modes with different wakeup times to better control the SRAM peripheral circuit leakage. For instance, one can have a ***low-leakage mode*** for DL1 cache peripheral circuit with a one-cycle wakeup delay but it would reduce leakage by only 47%. Alternatively, one can define an ***ultra low power mode*** with a 3-cycle wakeup that saves 84% of leakage. Only the sleep transistor bias voltage used in these modes is different, otherwise the circuit is the same. Thus one can *dynamically switch modes during execution with almost no delay*. The question is when and how to use these different low-leakage modes for each of the SRAM units to maximize the leakage reduction while minimizing the wakeup delay and its impact on performance.

This paper first defines a mechanism that exploits both L1 and L2 cache miss information to decide when to transition on-chip SRAM peripherals to different low-power modes. It uses a state machine which is quite small and does not impact area or power consumption. The state machine tracks L2 or multiple L1 cache misses to guide these transitions because such events usually stall the processor. Applying the same general mechanism *may not deliver optimal* results in each unit. Furthermore, *it can degrade performance significantly* in some units, such as the IL1 cache which is accessed frequently. Therefore, to maximize the leakage reduction at *no* performance cost the control algorithm is optimized for individual on-chip SRAM-based units depending on their behavior and their ability to hide the wakeup delay associated with various low power modes. We thus propose to ***customize*** the general algorithm for each on-chip SRAM unit and to make the control local. For instance, for branch predictor (BP) we propose a novel run-time history-based mechanism to predict the period when BP is accessed very infrequently. Once such a period is identified the branch predictor peripheral is put into a low power mode.

The leakage and the corresponding temperature reduction of various SRAM-based units in the processor are evaluated in this paper using the proposed circuit and architectural techniques. Temperature is evaluated using a modified version of HotSpo t [28]. The modification replaced the library of temperature vs. leakage power data with post-fabrication measurements for 45nm technology for different types of transistors in the range from 27$^oC$ to 150$^oC$ in increments of 0.1$^oC$; a similar methodology was introduced in [42].

This work makes the following major contributions:

1) Highlights the importance of leakage power reduction in on-chip SRAM peripherals, demonstrating that the peripheral circuits – data drivers, address driver, decoder, and wordline drivers – account for over 80% of the overall SRAM leakage.
2) Highlights large thermal variations in on-chip SRAMs and shows the peripherals to be the thermal hotspot in on-chip memories.
3) Extends the zig-zag share circuit to enable multiple sleep modes for cache peripherals. Each mode represents a trade-off between leakage reduction and the wakeup delay.
4) Proposes a low-power design for the bias generator circuit required for the multiple sleep modes operation, shown to be fairly robust against six-sigma process, voltage and temperature variations.
5) Explores the design space of sleep transistor insertion in SRAM peripheral circuitry
6) Defines several low-leakage modes using multiple bias voltages in sleep transistors to allow a trade-off between wakeup delay and leakage reduction. These are combined with a recently proposed zig-zag share circuit technique (multiple sleep mode zig-zag share).
7) Proposes a low-overhead state machine control which exploits DL1 and L2 cache miss information to control multiple sleep mode zig-zag share circuitry.
8) Optimizes state machines for each individual unit to exploit its behavior and hide the wakeup delay associated with each low power mode.
9) Evaluates leakage power and corresponding steady state and peak temperature reduction for individual SRAM-based units. Also, evaluates the power/area overhead of the controlling circuitry.

Overall, the average leakage reduction is very significant across all SRAM units; 75% for L2, 57% for Branch Predictor, 57% for DL1, 38% for IL1, 38% for Floating Point and Integer Register File (FRF and IRF), 49% for INT/FP Rename, and 45% for DTLB/ITLB. The peak temperature across different blocks can be reduced by as much as 8.4$^oC$. A noticeable steady state temperature reduction is observed across all the units, varying form 3.5 $^oC$ for register map unit to 7.3 $^oC$ for L2 cache.

## 2. SLEEP TRANSISTOR STACKING

Stacking sleep transistors have been proposed to reduce sub-threshold ($I_{Dsub}$) or weak inversion current [9]. As shown in Figure 3 by stacking transistor N with slpN, the source to body voltage ($V_M$ ) of transistor N increases. When both transistors are off increase in $V_M$ reduces the $V_T$ of the transistor N and therefore reduces sub-threshold leakage current. [9]. Size (W/L) and bias ($V_{gslpn}$) voltage of the stacked sleep transistor determines the $V_M$ [9,15]. Reducing sleep transistor bias reduces the leakage but increases the circuit wakeup period, the time to pull the $V_M$ down to ground. Thus there is a trade-off between the amount of leakage saved and the wakeup overhead [15]. Now let us study the source of subthreshold leakage in a wordline driver. A wordline driver drives the gate of access transistors of all connected memory cells. The number and size of inverters in the chain are chosen to meet the timing requirements for charging or discharging the wordline. The inverter chain has to drive a logic value 0 to the pass transistors when a memory row is not selected. Thus the driver cannot be simply shut down when idle. Transistors N1, N3 and P2, P4 are in the off state and thus they are leaking.

Stacking header and footer sleep transistors with all NMOS and PMOS transistors in the chain reduces their leakage; however, aside from the area overhead, it increases the propagation delay of the inverters in the driver chain followed by an increase in the rise/fall time of the wordline [3,7]. While increasing the rise time and propagation delay (due to its impact on access time) is not desirable, increasing the fall time is not tolerable since it can affect memory functionality [18, 20]. Increase in the fall times of the wordline increases the access transistor's active period of a memory cell during a read operation. This results in the bitline over-discharge and the memory content over-charge during the read operation. Such over-discharge not only increases the dynamic power dissipation of bitlines but, more importantly, can cause a memory cell content to flip if the over-discharge period is large [7,20]. In brief, to avoid impacting memory functionality the sense amplifier timing circuit and the wordline pulse generator circuit need to be redesigned. To avoid the redesign of these critical units and, moreover, not to increase bitline dynamic power dissipation we use the ***zig-zag horizontal and vertical share*** circuit technique proposed in [3].

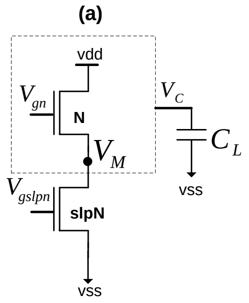


**Figure 3. Stacking sleep transistor to reduce leakage.**

### 2.1 Zig-zag Share Circuit

In [3] several approaches for reducing leakage power through sleep transistor insertion have been studied. They proposed zig-zag share scheme which uses typical $V_{th}$ sleep transistors. They have shown this technique to be the most effective in reducing leakage in SRAM peripheral while requiring minimal design overhead with minimal or no impact on circuit speed. Note that while it is possible to use high $V_{th}$ transistors in the peripheral circuits to reduce leakage it is not justified due to the extra mask layer cost and the impact on circuits speed [3, 44]. Unlike zig-zag share scheme which has minimal/no impact on peripheral circuits timing components, due to timing impact of using high $V_{th}$ transistors, the peripheral circuits must be redesigned. To avoid such an extra design spin and cost overhead, in this work we deploy and further explore ***zig-zag horizontal and vertical share*** circuit technique (in brief *zz-hvs*).

In this approach, sleep transistors are inserted in a zig-zag fashion [38, 39] keeping the Rpeq of the first and third inverters and Rneq of the second and fourth inverters constant. This technique keeps the fall time of the circuit the same as in the baseline circuit with no leakage control. However, the rise time of the circuit is affected by the zig-zag scheme. In addition, using one sleep transistor per inverter logic increases the area for the zig-zag scheme. To improve both leakage reduction and area-efficiency of the zig-zag scheme, one set of sleep transistors is being shared between multiple stages of inverters which have similar logic behavior, such as stage 1 and 3 in a studied chain of inverters. To further reduce leakage power one set of sleep transistors (slpN and slpP) is shared vertically with adjacent rows of a (wordline) driver. [3] further explored the design space of sleep transistor insertion in SRAM peripheral circuitry and shown the effect of sleep transistor size, number of horizontal and vertical level sharing on the trade off between the leakage power savings and the impact on instability, area, dynamic power, propagation delay, rise time and fall time delay increases on the peripheral circuit of SRAM.

Intuitively, in vertical sharing, the virtual ground voltage (VM in Figure 3) increases in comparison to when there is no vertical sharing. Results show that using zz-hvs reduces the leakage power significantly, by 10 to 100X, when 1 to 10 wordlines share the same sleep transistors. Such noticeable savings comes at negligible impact on memory access time, dynamic power and area increase. The maximum switching power of the sleep transistors in zz-hvs scheme is shown to be ~8% of total inverter chain power dissipation. Sharing the sleep transistor across multiple stages of inverter chain, both horizontally and vertically, combined with infrequent switching of sleep transistors makes the additional power dissipation much smaller. Also the memory access delay shown to be increased by up to 5% in a non- pipelined memory [3]. Pipelined memories such as L1 and L2 caches, can hide this small increase in peripheral circuit latency. In this paper we assumed that other on-chip SRAMs, register file, branch predictor, DTLB/ITLB and rename table, can tolerate such an increase without impacting the processor operating clock frequency.

## 3. ZIGZAG-SHARE WITH MULTIPLE SLEEP MODES

As explained in Section 2, to benefit the most from the leakage savings of stacking sleep transistors we need to keep the bias voltage of NMOS footer sleep transistor as low as possible (and for PMOS header transistor as high as possible). The drawback of such biasing is its impact on the wakeup latency of the circuit transitioning from sleep mode to active mode, which requires the voltage of virtual ground to reach the true ground. Such wakeup delay would significantly impact performance if incurred frequently. Appropriately sizing the sleep transistor (both footer and header) [3] and controlling its bias voltage [15] are two effective ways to minimize the impact of wakeup delay. For instance, increasing the gate voltage of footer sleep transistor (in Figure 3) reduces the virtual ground voltage ($V_M$), which consequently reduces the circuit wakeup delay. The negative impact of such biasing is a reduction in leakage power savings.

To better explain this, let us examine the wakeup delay and wakeup power overhead as a function of sleep transistor size and its gate bias voltage. The wakeup delay and power overhead are measured as the time and power required for the virtual ground voltage (VM in Figure 2) to be discharged through a sleep transistor to reach the ground level [20]. This wakeup delay is expressed as follows:

$$E_{wakeup} = \frac{1}{2} C_{block} \times V_M^{\ 2} \qquad \text{Equation 1}$$

$$T_{wakeup} = \frac{V_M \times C_{block}}{I_{slp}} \qquad \text{Equation 2}$$

where $C_{block}$ is the total capacitance of the circuit block and $I_{slp}$ is the current of sleep transistor after it turned on to wake up the block. Such wakeup overhead is decided by the equivalent load, as shown in EQ.1. As shown in [15], virtual ground voltage is linearly dependent on the sleep transistor gate voltage; increasing the gate voltage of the NMOS sleep transistor reduces VM. According to EQ.1 and EQ.2, such reduction lowers the wakeup delay and wakeup power overhead. Also as discussed in Section 2, increasing the gate voltage of the sleep transistor results in higher leakage power dissipation. In fact, by controlling the gate voltage of the footer and header transistors we can define different sleep modes where each mode has a different wakeup delay overhead and a different amount of leakage power reduction.

### 3.1 The Bias Generator

Multiple gate bias voltage levels for multiple sleep modes can be generated by a robust bias generation circuit such as [15, 46, 47, 48]. Note that generating multiple bias voltages requires using on-chip voltage converters. Several recent memory products from Intel [46, 47], Hitachi, Renesas [48] and others use such on-chip voltage converters for body bias generation. The area overhead of body bias generation and distribution was shown to be small, 1~3% of total chip area [47, 49].

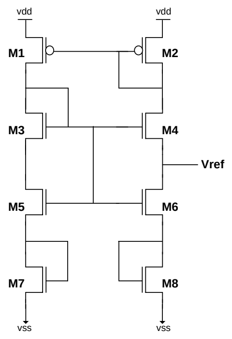


**Figure 4. A robust gate bias generator circuit.**

In this work we use the bias generator circuit shown in Figure 5. Conventional bandgap reference circuit consists of bipolar transistors and resistors that might occupy a noticeable amount of area. A CMOS voltage reference in Figure 4 consisting of transistors in saturation, triode, and subthreshold regions can provide sub-1-V reference required in our design. Transistors M1 – M4 in Figure 4 are always in saturation and generate a supply independent current. Transistors M5 and M6 operate in the triode region as voltage controlled resistors and produce PTAT (Proportional to Absolute Temperature) voltage. Transistors M7 and M8 operate in the subthreshold region, behaving as bipolar transistors to produce CTAT (complementary to absolute temperature) voltage.

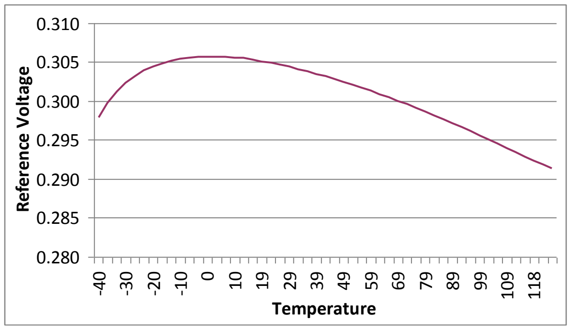


**Figure 5. Reference voltage versus temperature in °C.**

The reference voltage is taken from the drain node of M6. Figure 5 shows the simulation results of reference voltage versus temperature from -40°C to 125°C, where only ±1.6% voltage variation around the nominal value is observed across this wide temperature range.

## 3.2 Impact of Sleep Transistor sizing on Leakage Power Reduction/Wakeup Delay and Propagation Delay

As discussed in [3], inserting sleep transistors in a zig-zag share fashion increases the circuit propagation delay as well as its rise time. Appropriately sizing of the sleep transistor can minimize the impact on circuit propagation delay as well as on its rise time. Assume that a β% increase in the wordline driver propagation delay is tolerable. Now let's find the appropriate size of sleep transistor so that the propagation delay does not increase beyond β%. Delay of a gate without sleep transistor is expressed as:

$$T_d = \frac{C_L \times V_{DD}}{(V_{DD} - V_{tl})^{\alpha}} \qquad \text{Equation 3}$$

Delay of a gate with sleep transistor is expressed as,

$$T_{dsleep} = \frac{C_L \times V_{DD}}{(V_{DD} - V_x - V_{tl})^{\alpha}} \qquad \text{Equation 4}$$

CL is the load capacitance at the gate's output, Vtl is the threshold voltage of the inverter chain.

With a β% delay overhead allowed during active operation of the word line driver,

$$\frac{T_d}{T_{dsleep}} = (1 - \frac{\beta}{100}) \qquad \text{Equation 5}$$

$$V_x = (\sqrt[\alpha]{(1 - \frac{\beta}{100})} - 1)(V_{DD} - V_{tl}) \qquad \text{Equation 6}$$

$$I_{sleep} = \mu_n \times C_{ox}\left(\frac{W}{L}\right)_{sleep}\left((V_{DD} - V_{th}) \times V_x - \frac{V_x^{\,2}}{2}\right) \qquad \text{Equation 7}$$

For the word line driver, $V_{tl}$ is the same as $V_{th}$ of sleep transistor.

$$I_{sleep} = \mu_n \times C_{ox}\left(\frac{W}{L}\right)_{sleep}\left(\left(\left(\sqrt[\alpha]{(1-\frac{\beta}{100})}-1\right) - \frac{\left(\sqrt[\alpha]{(1-\frac{\beta}{100})}-1\right)^2}{2}\right)(V_{DD} - V_{th})^2\right)$$

Equation 8

Isleep is the maximum current flowing through the ground, which is the total discharge current of the word line driver. Assuming each inverter stage is sized 'a' times larger than the previous stage the total discharge current is expressed as,

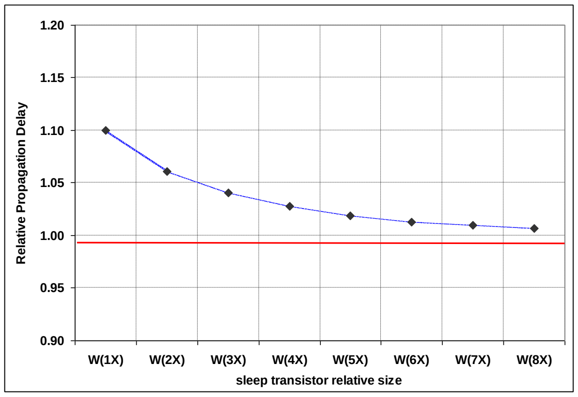


**Figure 6. Impact of sleep transistor sizing on propagation delay**

$$I_{discharge} = \frac{(V_{DD} - V_x))}{R_{eq,n}} + \frac{(V_{DD} - V_x))}{\frac{R_{eq,n}}{a^2}} + \frac{(V_{DD} - V_x))}{\frac{R_{eq,n}}{a^4}} + \ldots = \qquad \text{Equation 9}$$

$$\frac{(V_{DD} - V_x))}{R_{eq,n}} \times (1 + a^2 + a^4 + \ldots)$$

The size of sleep transistor can be found by substituting the discharge current into EQ.8.

$$\left(\frac{W}{L}\right)_{sleep} = \frac{I_{discharge}}{\mu_n \times C_{ox}\left(\left(\left(\sqrt[\alpha]{(1-\frac{\beta}{100})}-1\right) - \frac{\left(\sqrt[\alpha]{(1-\frac{\beta}{100})}-1\right)^2}{2}\right)(V_{DD} - V_{th})^2\right)}$$

Equation 10

The impact of sleep transistor sizing on the propagation delay for ZZ-HVS circuit when 10 rows of wordline drivers share sleep transistor is shown in Figure 6 . Increasing the size of sleep transistor reduces the propagation delay overhead, for instance increasing the size of sleep transistor by 8X reduces the impact on propagation delay from 10% down to near 1%.

**Table 1. Impact of sleep transistor sharing and sizing on the wakeup delay**

| **#shared inverter chains** | W(1X) (ns) | W(2X) (ns) | W(3X) (ns) | W(4X) (ns) | W(5X) (ns) | W(6X) (ns) | W(7X) (ns) | W(8X) (ns) |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.256 | 0.137 | 0.093 | 0.064 | 0.045 | 0.037 | 0.032 | 0.029 |
| 2 | 0.620 | 0.367 | 0.273 | 0.205 | 0.155 | 0.136 | 0.124 | 0.115 |
| 3 | 1.190 | 0.732 | 0.583 | 0.464 | 0.381 | 0.345 | 0.321 | 0.309 |
| 4 | 1.655 | 1.072 | 0.877 | 0.736 | 0.637 | 0.596 | 0.564 | 0.556 |
| 5 | 2.130 | 1.438 | 1.214 | 1.065 | 0.952 | 0.905 | 0.884 | 0.882 |
| 6 | 2.595 | 1.817 | 1.609 | 1.453 | 1.336 | 1.298 | 1.277 | 1.275 |
| 7 | 3.050 | 2.196 | 1.983 | 1.830 | 1.739 | 1.708 | 1.699 | 1.696 |
| 8 | 3.525 | 2.609 | 2.432 | 2.291 | 2.203 | 2.178 | 2.171 | 2.170 |
| 9 | 4.010 | 3.036 | 2.887 | 2.767 | 2.695 | 2.675 | 2.667 | 2.663 |
| 10 | 4.450 | 3.471 | 3.338 | 3.235 | 3.182 | 3.168 | 3.163 | 3.160 |

Table 1 reports the impact of sleep transistor sharing and sizing on wakeup delay when each wordline drives 256 one-bit memory cells. Higher degree of sharing of sleep transistor results in larger wakeup delay. Increasing the size of sleep transistor reduces the wakeup delay. Increasing the number of shared wordlines reduces the benefit of using a larger sleep transistor on wakeup delay reduction.

Figure 7 reports the relative leakage power reduction as a function of sleep transistor size and the number of sharing inverter chains. These results show that the sleep transistor size has a small impact on leakage power savings.

Finally, note that the power overhead of waking up peripheral circuits from any low power mode is negligible, almost equivalent to the switching power of sleep transistors. Sharing a set of sleep transistors horizontally and vertically for multiple stages of (wordline) drivers makes the power overhead even smaller.

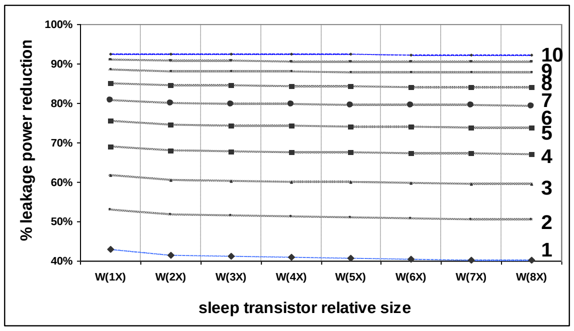


**Figure 7. Relative leakage power reduction.**

## 3.3 MULTIPLE SLEEP MODES SRAMS

The proposed multiple sleep mode zig-zag share approach was applied to various SRAM wordline drivers. Several test experiments were set up in which the wordline inverter chain drives 256, 128, 64 and 32 one-bit memory cells. The drivers were laid out using Mentor Graphic IC-Station in a 45nm technology and simulated using Synopsis Hspice at typical corner (25 º) with extracted netlist and the supply voltage of 1.0V. A standard SRAM6T memory cell was used.

Figure 8 shows normalized wakeup delay and normalized leakage power for different pairs of footer and header gate bias voltage when zz-hvs is shared by 10 rows of wordline drivers with each wordline driving 256 one-bit memory cells. A clear trade-off can be seen between the normalized wakeup delay and leakage power. For other cases where the driver load changes (driving 32, 64 and 128 one-bit memory cell) we observed that the *relative* leakage reduction does not change significantly. Also it should be noted that when the peripheral circuits are in low power mode the overall time delay for transition to/from standby mode, STL, is the sum of sleep transistors wakeup delay and propagation delay of sleep signal. Both of these delays increase as the memory area increases, especially the latter delay, because the sleep signal needs to be transmitted over a greater distance. Accordingly, depending on memory size and configuration, there is a different wakeup delay overhead for a specific zz-hvs bias voltage. To find the STL delay for different SRAM memories, Spice and CACTI were used to measure the wakeup delay of sleep transistor and propagation delay of the sleep signal, respectively, for various SRAM units. To estimate the propagation delay we assume that the sleep signal has to be transmitted across the SRAM peripherals. Based on these experimental results, four sleep modes with different wakeup delays were defined for each on-chip SRAM memory. The first sleep mode is the basic-lp mode, which requires a near-zero wakeup overhead. In fact, for on-chip SRAMs with access delay smaller than clock period, the zz-hvs circuit is biased such that the overall access delay and wakeup delay overhead is less than the clock period, so that the SRAM still can be accessed in one (processor clock) cycle while waking its peripherals from basic-lp mode. The other low-power modes are low power (lp), aggressive low power (aggr-lp) and ultra low power (ultra-lp). Their peripheral wakeup delays are 1, 2 and above 3 cycles, respectively.

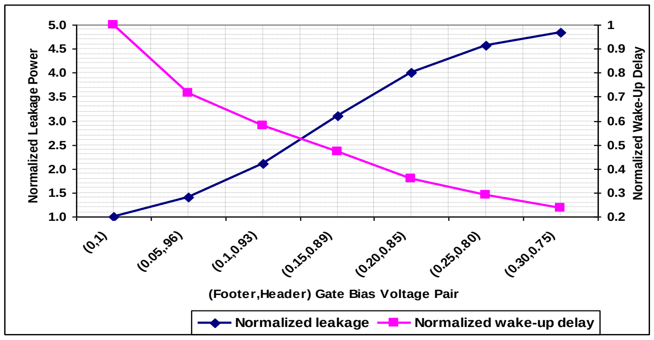


**Figure 8. Normalized wakeup delay and leakage power for different pair of footer and header gate bias voltage.**

Table 2 shows relative peripheral circuit leakage reduction for different modes for each on-chip SRAMs. For IL1, DL1 and L2, due to their large size, the overall wakeup delay (sleep transistor wakeup delay+sleep signal propagation delay) is more than 1 cycle for any bias voltage. As such, these units can not be put into basic-lp mode.

As the results suggest, the rest of the SRAMs can be put into basic-lp mode to reduce their leakage but their access can still be completed in one processor cycle. Recall that the wakeup delay for basic-lp mode is virtually zero. Aggressive and ultra sleep power modes have higher leakage savings but also a longer wakeup delay. Access to an SRAM in one of these modes requires two or more extra cycles in addition to its access time. For instance, FRF/IRF access while in aggr-lp mode requires 3 cycles (a 2-cycle wakeup and 1-cycle access) while for L2 and DL1 this is 4 cycles. (a 2-cycle wakeup and 2-cycle access).

**Table 2. On-chip SRAM peripherals multiple sleep mode normalized leakage power savings**

| | BPRED | FRF | IRF | IL1 | DL1 | L2 | DTLB | ITLB | RENAME |
|---|---|---|---|---|---|---|---|---|---|
| **basic-lp** | 0.29 | 0.21 | 0.2 | -- | -- | -- | 0.25 | 0.25 | 0.31 |
| **lp** | 0.63 | 0.51 | 0.5 | 0.4 | 0.4 | -- | 0.54 | 0.54 | 0.53 |
| **aggr-lp** | 0.75 | 0.68 | 0.6 | 0.5 | 0.5 | 0.5 | 0.69 | 0.69 | 0.67 |
| **ultra-lp** | 0.91 | 0.85 | 0.8 | 0.7 | 0.8 | 0.9 | 0.93 | 0.93 | 0.87 |

Finally, note that the power overhead of waking up peripheral circuits from any low power mode is negligible and almost equivalent to the switching power of sleep transistors (and they do not switch very frequently). Sharing a set of sleep transistors horizontally and vertically (as explained in section 2.1) for multiple stages of (wordline) drivers makes the power overhead even smaller. As a result, the power benefit of the proposed circuit scheme is less sensitive to transition frequency between different power modes.

## 4. CONTROLLING MULTIPLE SLEEP MODE ZZ-HVS FOR ON-CHIP SRAMS

This section describes the architectural approach used to control multiple low-power modes based on zz-hvs sleep transistors in BPRED, FRF, IRF, IL1, DL1, L2, DTLB, ITLB and RENAME SRAMs. The approach was evaluated for a 64-bit processor similar to Alpha 21264, described in Table 3, The processor clock frequency was assumed to be 2.2 GHz. It was simulated using an extensively modified version of SimpleScalar4 [11] and SPEC2K benchmarks with reference data sets. Benchmarks were compiled using the Compaq compiler with the -O4 flag targeted for the Alpha 21264 processor. The benchmarks were fast–forwarded for 2 billion instructions, then fully simulated for 2 billion instructions.

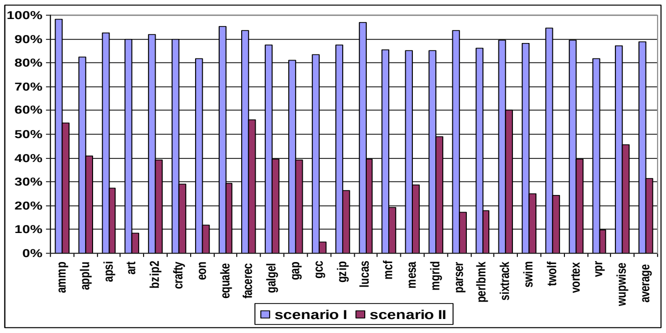
**Table 3. Processor Configuration**

| | | | |
|---|---|---|---|
| L1 I-cache | 128KB, 64 byte/line, 2 cycles | Instruction queue | 64 entry (32 INT and 32 FP) |
| L1 D-cache | 128KB, 64 byte/line, 2 cycles, 2 R/W ports | Register file | 128 integer and 128 floating point |
| L2 cache | 2MB, 8 way, 64 byte/line, 20 cycles | Load/store queue | 32 entry load and 32 entry store |
| issue | 4 way out of order | Arithmetic unit | 4 integer, 4 floating point units |
| Branch predictor | "tournament" predictor | Complex unit | 2 INT, 2 FP multiply/divide units |
| Reorder buffer | 128 entry | Pipeline | 15 cycles (some stages are multi-cycles) |

## 4.1 Reducing Leakage in On-Chip SRAM Peripherals

To maximize the leakage reduction in each of the on-chip SRAM memories peripherals one solution would be to always put them into ultra low power mode. However, this requires wakeup of their peripheral circuits before accessing them adding 3 cycles to their access latency and significantly reducing performance. In addition to performance degradation, increased access time for some units, such as a register file, would require significant modification of the pipeline and further complicate the instruction scheduler [24]. Alternatively, one can put SRAM peripherals into the basic low power mode (except for IL1, DL1 and L2), which requires virtually zero cycles to wakeup thus not degrading performance. However, this doesn't significantly reduce leakage power (see Table 2**.** To achieve the large leakage reduction of ultra and aggressive low power modes with the performance impact of basic-lp mode one has to dynamically change the peripheral circuit sleep modes. During periods of frequent access they need to be kept in basic-lp mode and when their access frequency is low they can be kept in aggr-lp or ultra-lp modes.

One period of infrequent access to any of discussed units is when the processor performance (in terms of IPC) drops significantly. Our study shows that during such period access to many on-chip SRAM units such as L1 cache, L2 cache, DTLB/ITLB, register file, branch predictor and rename table drops significantly. Cache misses are the main reason for processor performance drops significantly. For instance, after an L2 cache miss the processor executes some independent instructions but finally ends up stalled [52] (scenario I). Similarly, a considerable performance reduction occurs during any period in which multiple L1 misses are pending (scenario II). Performance degradation in both cases is due to the fact that a load instruction missing in a cache (DL1 or L2) prevents any dependent instruction from being issued until the miss is serviced. For a long-latency L2 miss, after the processor executes a number of independent instructions, either the *ROB, LQ/SQ* or instruction queue fills up with subsequent instructions and the processor ends up stalled until the miss is serviced. We refer to the interval between L2 miss occurrence and processor stall as *L2 miss pre-stall period*. We refer to the stall period following L2 miss as *L2 miss stall period*.



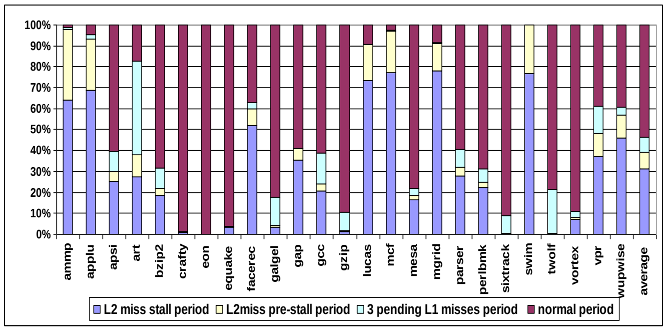


**Figure 9 (a) Performance decrease for scenarios I and II. (b) Fraction of execution time in different periods.**

Note that when a DL1 cache miss occurs, its dependent instructions cannot be issued and that all the subsequent instructions cannot be committed as discussed above. This reduces the performance and increases the occupancy of the aforementioned queues. In the presence of many pending DL1 cache misses (referred as *L1 miss period*), the impact on performance could be large.

We refer to a period during which one or more L2 miss/misses and/or multiple DL1 misses are pending as a "cache miss period," and to the rest of program execution time as a "normal period." It should be noted that the two scenarios discussed above would occur when the missed load is part of a correct prediction path, otherwise after the correct path has been identified, the missed load instruction will be flushed and will release *ROB*/IQ/ LQ/SQ entries so that program execution can continue (return to the normal period). Figure 9(a) shows the IPC reduction for scenario I compared to when there is no pending L2 miss and the IPC reduction for scenario II compared to the period where there are less than 3 pending DL1 misses. The IPC decreases significantly in both cases. Across all benchmarks, the IPC drops by more than 88% for scenario I and more than 31% for scenario II. Figure 9(b) presents the fraction of execution time the processor spends in different low-power periods. On average, the processor pipeline is stalled for more than 30% of execution time due to L2 cache misses, it spends 8% of execution time during L2 miss pre-stall period. For benchmarks such as ammp, applu, lucas, mcf, mesa, and swim there is a large fraction of stall time - more than 60%. For more than 7% of program execution time there are at least 3 pending L1 misses in the pipeline. The processor spends 53% of execution time in normal period.

## 5. LEAKAGE CONTROL MECHANISM

Based on the results presented in Figure 9, one can use the L2 and L1 cache miss information to decide when to put SRAM peripherals into different low power modes. Figure 10 shows the general state machine is proposed to control the power transitions:

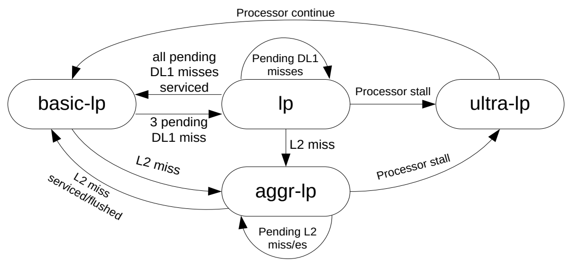


**Figure 10. General state machine to control power mode transitions.**

On an L2 cache miss the SRAM peripheral circuits are transitioned from basic to a deep low power mode, aggr-lp. The pipeline continues issuing and executing instructions (pre-stall period) until one of the ROB, instruction queue, or load/store queue fills up. Once the pipeline stalls, the SRAM peripherals transition to the ultra-lp mode until the miss is serviced. A transition from basic-lp to lp mode occurs when at least three DL1 misses are pending (a L1 cache miss period). Occurrences of multiple DL1 misses increase the probability of pipeline stalls due to data dependencies. It thus makes sense to put the on-chip SRAM peripherals into a sleep mode with higher leakage savings (lp). A processor stall is detected by monitoring the issue width of the processor after L2/multiple DL1 cache miss/es occur. The processor is transitioned into ultra-lp once it doesn't issue any instructions for at least five consecutive cycles. The processor returns from any of these low power states back to the basic-lp mode once one of the two following conditions is met:

- Stall condition removed, i.e. instruction issue resumes
- All pending DL1 misses are serviced

The proposed general algorithm may not deliver optimal results for all units. Therefore, the algorithm is modified for individual on-chip SRAM-based units to maximize the leakage reduction at NO performance cost, as described next.

## 5.1 Branch Predictor

On average, one out of every 9 fetched instructions in integer benchmarks and out of 63 fetched instructions in floating point benchmarks accesses the branch predictor (see in Table 4). Such infrequent access would seem to make the branch predictor a good candidate for always staying in deep low power modes (lp, ultra-lp or aggr-lp) and waking up on access. However, this approach results in noticeable performance degradation for some benchmarks [54].

One reason is that in some benchmarks the branch predictor is accessed very frequently, such as ammp, equake and mcf. Another is that within a benchmark there is significant variation in Instructions Per Branch (IPB). Figure 11 shows IPB measured every 512 cycles for two benchmarks with different average IPB: *swim* with a very high average IPB and *equake* with a very low average IPB. The highlighted circles for swim identify regions in which the number of branch in a 512-instruction interval is significantly lower than its average IPB of 77. A similar situation is true for equake. Also note the duration of a period where the IPB stays low (or high): once the IPB drops (increases) significantly it may remain low (high) for a long period of time. Based on these observations one can identify the high IPB period, once the first low IPB period is detected. The following algorithm is proposed for this prediction:

*The number of fetched branches is counted every 512 cycles, once the number of branches is found to be less than a certain threshold (24 in this work) a high IPB period identified. The IPB is then predicted to remain high for the next twenty 512 cycles intervals (10K cycles).*

Using this algorithm branch predictor peripherals transition from basic-lp mode to lp mode when a high IPB period is identified. During pre-stall and stall periods the branch predictor peripherals transition to aggr-lp and ultra-lp mode, respectively.

**Table 4. Instruction per branch count**

| | IPB | | IPB | | IPB | | IPB |
|---|---|---|---|---|---|---|---|
| **ammp** | 4.5 | **equake** | 4.21 | **mcf** | 3.9 | **twolf** | 7.6 |
| **applu** | 324.1 | **facerec** | 20.0 | **mesa** | 11.0 | **vortex** | 5.7 |
| **apsi** | 28.9 | **galgel** | 14.3 | **mgrid** | 310.4 | **vpr** | 9.0 |
| **art** | 8.1 | **gap** | 14.2 | **parser** | 6.0 | **wupwise** | 8.7 |
| **bzip2** | 6.7 | **gcc** | 6.3 | **perlbmk** | 7.2 | **average** | 37.8 |
| **crafty** | 8.5 | **gzip** | 9.5 | **sixtrack** | 11.9 | | |

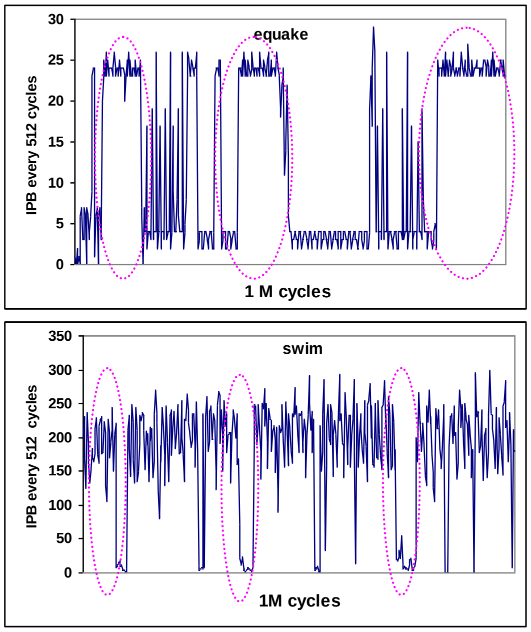


**Figure 11. Distribution of the number of branches per 512-instruction interval (over 1M cycles)**

## 5.2 L1 Data Cache

DL1 cache read ports are accessed more frequently than its write ports. Results in Table 5 shows that on average a DL1 write port is accessed once every 30 cycles, while a read port is accessed every 8 cycles. Such different access pattern, require different control mechanism for reducing leakage. Our evaluation showed that making a write port one cycle slower has almost no impact on performance. But a one extra cycle of delay on every DL1 read (port) leads to noticeable performance degradation in some of the benchmarks (e.g. gzip, twolf and vpr). Therefore, the DL1 write port is always kept in lp mode and is awaken only when it is accessed. The read port is put into lp mode only during L1 cache miss period where processor performance drops noticeably and slowing load instructions execution by one cycle does not appreciably degrade performance. During pre-stall period the L1 peripherals are put into aggr-lp mode. Both read and write ports are put into ultra-lp mode once processor is stalled.

**Table 5. Average latency (number of cycles) between two successive accesses to DL1 read and write ports**

| | DL1 read | DL1 write | | DL1 read | DL1 write |
|---|---|---|---|---|---|
| **ammp** | 15.8 | 73.8 | **lucas** | 22.7 | 58.1 |
| **applu** | 10.5 | 23.6 | **mcf** | 28.9 | 203.8 |
| **apsi** | 5.4 | 11.1 | **mesa** | 3.7 | 9.8 |
| **art** | 6.3 | 29.5 | **mgrid** | 7.0 | 52.9 |
| **bzip2** | 4.2 | 14.3 | **parser** | 6.4 | 18.2 |

| | | | | | |
|---|---|---|---|---|---|
| **crafty** | 3.2 | 17.8 | **perlbmk** | 5.9 | 11.5 |
| **eon** | 3.9 | 6.02 | **sixtrack** | 3.7 | 10.3 |
| **equake** | 5.7 | 14.1 | **swim** | 18.1 | 44.8 |
| **facerec** | 6.1 | 10.8 | **twolf** | 6.6 | 22.2 |
| **galgel** | 3.5 | 33.6 | **vortex** | 4.0 | 7.6 |
| **gap** | 7.8 | 15.5 | **vpr** | 6.6 | 20.8 |
| **gcc** | 5.8 | 8.9 | **wupwise** | 8.9 | 18.2 |
| **gzip** | 5.3 | 19.7 | **average** | 8.2 | 30.4 |

## 5.3 L1 Instruction Cache

IL1 is accessed very frequently. As explained in Sec. 3, due to its large size, the overall wakeup delay of its peripherals from basic-lp mode (sleep transistor wakeup delay plus sleep signal wakeup delay) takes more than 1 cycle for any bias voltage. Therefore, because of frequent accesses, putting IL1 cache peripherals into basic-lp mode would result in significant performance degradation.

Due to very different activity patterns of IL1 read and write ports we treat them differently. The IL1 cache read port (data output drivers, read address input drivers, decoder/predecoder drivers) is being accessed more frequently than its write port (data input drivers, write address input driver and row decoder/predecoder drivers). IL1 read port is accessed on average almost every cycle, while the write port is accessed very infrequently, almost once every 93 cycles in SPEC2K benchmarks. Such different access patterns require a different control mechanism for reducing leakage. Our evaluation showed that making a write port one cycle slower has almost no impact on performance, but one extra cycle of delay on every IL1 read (port) leads to noticeable performance degradation in some of the benchmarks. Therefore, the IL1 write port is always kept in lp mode and is awakend only when it is accessed. Both IL1 read and write ports (in both architectures) transition to the ultra-lp mode when a processor stalls. Therefore, the IL1 cache peripherals transition to the ultra-lp mode only when an L2 cache miss is detected. Our results indicate that this approach doesn't degrade performance. By putting IL1 peripherals into ultra-lp mode every access to it will take 5 cycles (2-cycle access time + 3-cycle wakeup time). Since during an L2 miss period the processor executes few independent instructions and finally ends up being stalled, slowing down the execution of independent instructions by slowing down the IL1 cache access, does not degrade the performance as long as independent instructions execution can be completed before the L2 cache miss is serviced.

Figure 12 shows the fraction of program execution time the IL1 cache spends in each of the power modes. On average, 92% of the total execution time the IL1 write port can be put into one of the low power modes. Most of the time is spent in the lp mode (67%). 25% of time is spent in ultra-lp mode. In some benchmarks the ultra-lp mode has a noticeable contribution. Examples are ammp, applu, lucas, swim and mgrid. In fact these are the benchmarks with the highest L1 miss rate (results not shown here) and as a result potentially large processor stall. The IL1 write port transitions to high power mode (hp) for only 8% of the execution time, on average. This is different than the read port which it spends most of the time in high power mode. In many benchmarks IL1 read port is being accessed almost every cycle and as a result there is no opportunity to put its read peripherals into low power mode. Examples are crafty, eon, equake, gzip, galgel, sixtrack, twolf. On average, IL1 read port can be put into leakage savings ultra -low power mode for only 25% of total program execution time.

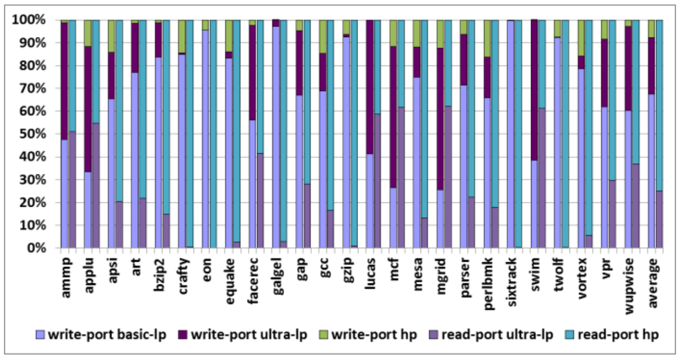


**Figure 12. (a) Fraction of total execution time IL1 cache spends in each of the power modes.**

## 5.4 L2 Cache

L2 cache is accessed infrequently (on average 1 out of 500 instruction access L2). It thus makes a lot of sense to always keep it in deep low power mode (aggr-lp) and only wake it up before accessing it. The wakeup delay of aggr-lp mode is 2 cycles which is much less than the L2 cache latency (20 cycles). As a result the performance impact of always keeping the L2 in aggr-lp mode is minimal. Once an L2 miss occurs, we put L2 into ultra-lp mode.

## 5.5 Register File

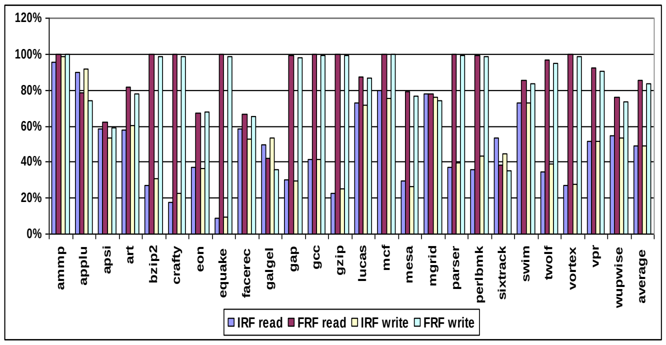
Register file is a multi-ported SRAM structure. For a 4-wide issue processor it has 4 write and 8 read ports. Evaluation results in Figure 13 (a) show that, on average, in more than 50% of all cycles there are no read and write port accesses to integer register file. The fraction is even higher for FRF: more than 80%. Given such low port utilizations, all register files read and write ports are kept in basic-lp mode. Note that as shown in Sec. 3 the register files can still be accessed in one cycle while waking up its peripherals from basic-lp mode. While such low port utilization makes it feasible to put register file peripherals into deeper low power mode and potentially save more leakage, it is not done to avoid significant modifications to the processor pipeline. Register file peripherals transition to the ultra-lp mode once the processor stalls completely.

## 5.6 DTLB/ITLB and Rename Unit

ITLB, DTLB and register rename unit are accessed very frequently (almost once every 2 cycles for DTLB and renamer and at least once every cycle for ITLB) as can be seen in Figure 13(b) (ammp, mcf, lucas and swim spend a large fraction of their execution time stalled, and thus behave differently). Therefore, these units are always in basic-lp mode and only woken up when accessed. They transition to the ultra-lp mode once the processor stalls. Other low power modes are not utilized to minimize the complexity overhead.

## 6. EXPERIMENTAL EVALUATION

This section describes the power and timing assumptions used and presents results for power and temperature reduction. We used the relative leakage power reduction of various power modes for different SRAM units reported in Table 2. These numbers were obtained using Hspice with extracted netlist at a supply voltage of 1.0V in a 45nm technology. The energy and power consumption for each unit were computed by multiplying its access counts by its per-access SRAM energy consumption. The energy per access and leakage power dissipation for individual SRAM units were obtained using CACTI-5.1 assuming 45nm technology, an operating frequency of 2.2GHz, and 1.0V supply voltage. For temperature calculations the floor plan shown in Figure 14 was used. The units marked as "**X**" are the SRAM units to which we applied the new leakage reduction techniques. Thermal model is described in Table 6. HotSpot [28] was used to estimate both power density and thermal profiles for different processor blocks.



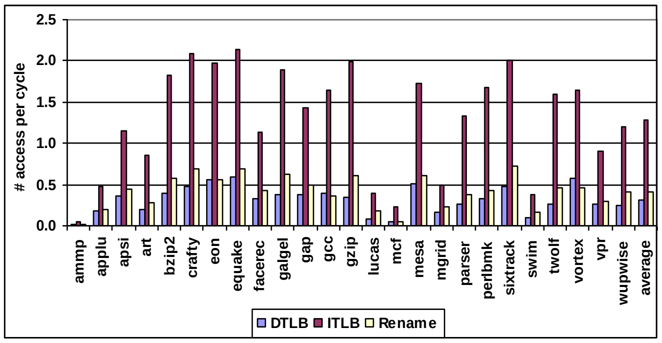


**Figure 13. (a) % execution time when register file ports are not accessed. (b) DTLB, ITLB and Renamer relative accesses per cycle.**

As explained earlier, both steady state temperatures and temperature traces were obtained every 10K cycles. From these temperature traces the peak temperatures reached by individual processor blocks during execution was determined. We also evaluated the power/area overhead of the controlling circuitry; a simple 2-bit state machine, a 2-bit saturating counter for keeping the number of pending DL1 misses and a 1-bit registers for keeping the L2 miss. Using Synopsys dc_compiler [50] we synthesized the state machine with a 45nm standard-cells which estimated the area overhead to be less than 150 gates (NAND2-gate). For branch predictor a 5-bit saturating counter is required which is counting for only 512-cycles in an every 10K-cycles intervals. The synthesized results estimate the gate count to be less than 80.

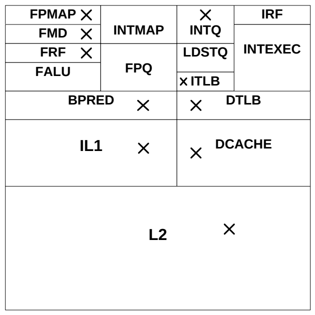


**Figure 14. Processor floorplan.**

**Table 6. Thermal Model**

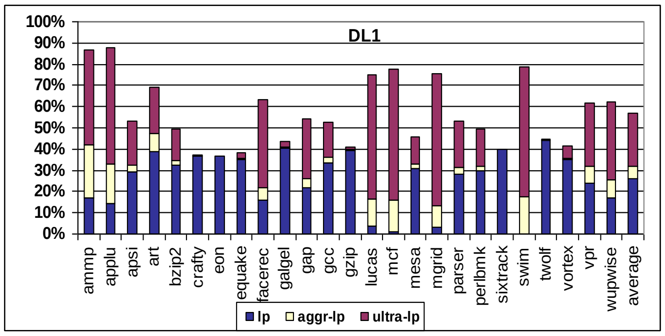
| | | | |
|---|---|---|---|
| Die thickness (μm) | 150 | Convection resistance | 0.1 K/W |
| Ambient temperature | 30°C | Heat sink side | 0.076 m |
| Convection capacitance | 140 J/K | Heat spreader side | 0.035 m |

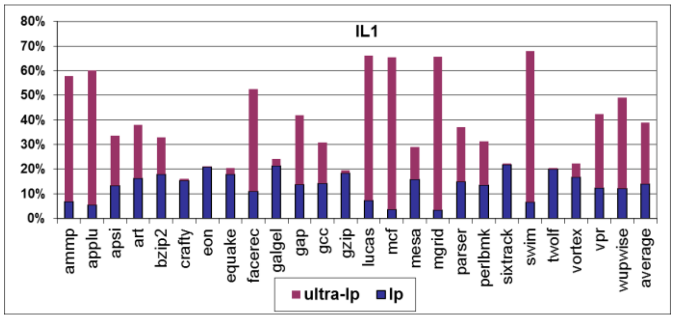
The overall power overhead estimated to be less than 0.9 mw (the power is almost 3nW/gate/MHz). Note that the exact area/energy measurements require the detailed floorplaning and post placement and routing information.

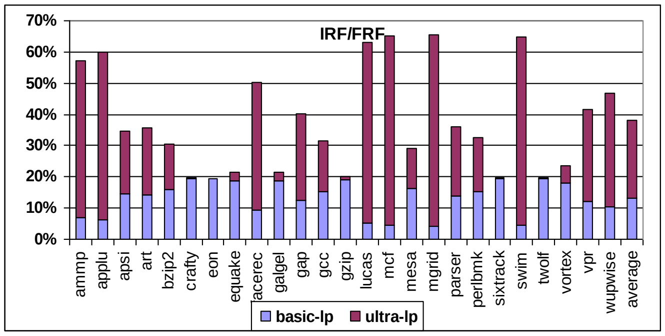
## 6.1 Leakage reduction

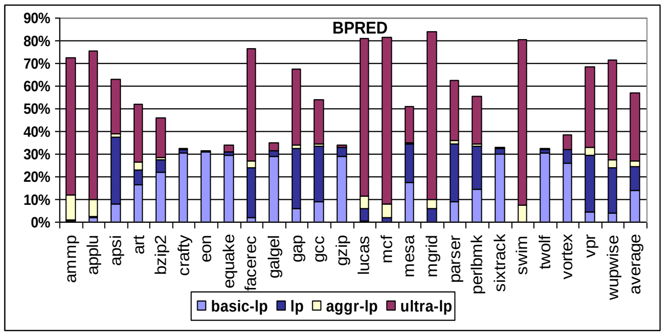
Figure 15 shows leakage reduction from applying our techniques to different on-chip SRAM units. These results consider the interdependence of leakage and temperature. The maximum leakage power reduction of 75% was achieved for L2 cache. This result is not unexpected since during execution the L2 cache is always kept in one of the deep low power modes except when it is accessed. On the other hand, the instruction cache (IL1) has the lowest leakage reduction, 38% on average.

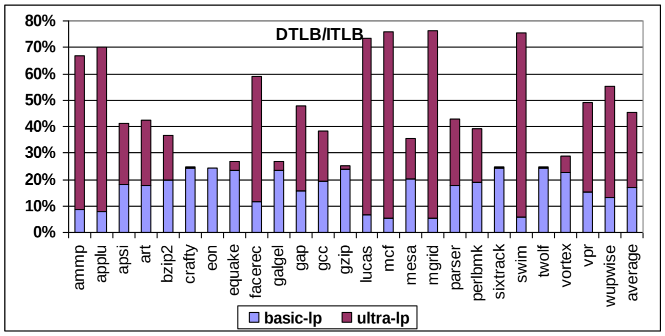
However, this reduction is very significant given the fact that the IL1 is almost continuously accessed providing fewer opportunities to change its power mode. Another interesting observation is that the reduction due to ultra-lp mode is very significant and is the highest of all low-leakage modes in many cases. This is because all on-chip SRAM units transition to this mode when the processor stalls. Large number of stalls, reported in Figure 9, combined with large leakage savings associated with the ultra-lp mode make this mode the major source of leakage reduction for all SRAM units. This is most noticeable in benchmarks with a large L2 miss period, such as ammp, applu, lucas, mcf and swim. The basic-lp mode makes a relatively small contribution to total leakage power reduction varying from 10 to 20%. While the basic-lp mode occurs more frequently than the ultra-lp mode, its contribution to total leakage reduction is less than the ultra-lp mode. The reason is that the peripheral leakage reduction of ultra-lp mode is 2 to 3 times that of basic-lp mode. The only SRAM units benefiting from the lp mode are the branch predictor and DL1 cache. In fact, these are the only units which are occasionally put into the lp mode. Overall, the average leakage reduction is very significant across all SRAM units; 75% (2.66 watt reduction) for L2, 57% (1.17 watt) for DL1, 25% (0.51 watt) for IL1, 57% (0.07 watt) for Branch Predictor, 38% (0.122 watt) for IRF,

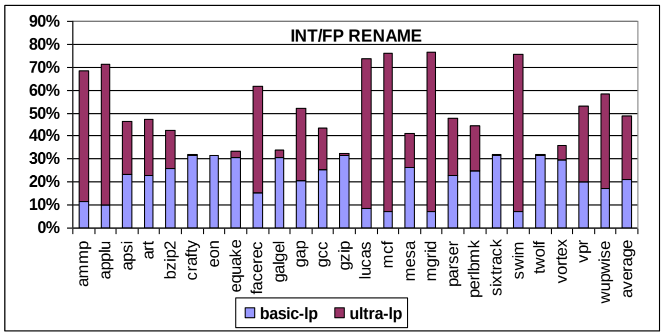












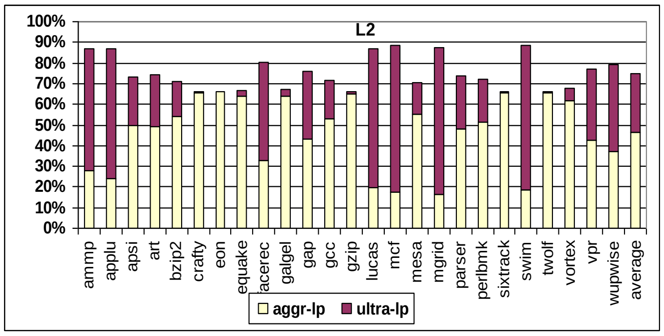


**Figure 15. Leakage reduction with thermal consideration for various on-chip SRAM units.**

38% (0.151 watt) for FRF, 49% (0.2 watt) for INT/FP Rename, and 45% (0.2 watt) for DTLB, 45% (0.31 watt) for ITLB. The total power reduction is 5.15 watt which translates to 10.86% of processor power dissipation (The studied processor dissipates 47 watt).

These results show the effectiveness of the proposed leakage reduction techniques. Our simulation results also show that applying our techniques does not degrade performance noticably. The maximum performance loss observed while applying the proposed techniques in all of the benchmarks was 0.063% in mcf. The reason is that the architectural algorithm hides the wake-up latency of de-activated SRAM peripherals. For instance all SRAM units transition to ultra-lp mode, which has a large wakeup delay, only once the processor stalls. During the stall period there is no access to the de-activated units and as a result there is no performance loss.

## 6.2 Temperature Impact of Leakage Reduction

Table 7 shows the average peak temperature for different blocks across all benchmarks before and after applying the leakage reduction techniques. For all units a large average temperature reduction is observed. We observe that the peak temperature across different blocks can be reduced by as much as 8.4$^{o}$C (for swim and in L2 cache). Consistent with the power results reported in

Reduction in the peak temperature of a processor is desirable because it eases the constraints on package design parameters and dynamic thermal management techniques, and may allow ***higher operating frequency***. We also observed a large peak temperature reduction in LdStQ and ITLB. Temperature reduction of LdStQ is of particular interest because we did not apply our leakage reduction technique to this block. The temperature reduction can be explained by *thermal diffusion*. The floor plan in Figure 14 shows that the LdStQ is placed next to the ITLB which has high leakage power and temperature reductions. The average reduction in peak temperature is the highest for the L2 cache (6.3$^{o}$C) corresponding to 75% reduction in L2 leakage power.

Table 8 shows the reduction in the steady state temperature of different blocks of the processor. An average reduction is higher for the steady state temperature compared to the peak temperature. Due to a positive feedback loop between temperature and leakage power, the reduction in temperature, shown in Table 7 and Table 8, is accompanied by a further reduction in leakage power (leakage results reported earlier account for this). Thermal consideration has further enhanced the significance of leakage reduction as a result of our proposed techniques.

## 7. RELATED WORK

A number of techniques were proposed for reducing leakage power at technology, circuit, architecture and compiler/OS levels.

### 7.1 Circuit-level leakage control

Several circuit techniques were proposed to reduce the leakage power in SRAM memories. These techniques were mainly targeting the SRAM memory cell leakage.

The primary technique is voltage scaling which due to short-channel effects in deep submicron processes reduces the leakage current significantly [21].

Another technique is Gated-Vdd which turns off the supply voltage of memory cells by using a sleep transistor and eliminating the leakage virtually completely [53]. However, it doesn't retain the state of the memory cells. The third technique, ABB-MTCMOS, increases threshold voltage of a SRAM cell dynamically through controlling its body voltage [16]. The overhead of applying this technique in terms of performance and area makes it inefficient. Device scaling leads to threshold voltage fluctuation, which makes the cell bias control difficult to achieve. In response, [8] proposed a Replica Cell Biasing scheme in which the cell bias is not affected by Vdd and Vth of peripheral transistors.

[4] and [14] proposed a forward body biasing scheme (FBB) in which the leakage power is suppressed in the unselected memory cells of cache by utilizing super Vt devices.

In addition to these four major techniques applied to SRAM memories, there are also leakage reduction techniques in literature which concentrated on generic logic circuits. Examples are sleepy stack [10], sleepy keeper [45] and zig-zag super cut-off CMOS (ZSCCMOS) techniques [55], [56]. ZSCCMOS reduces the wakeup overhead associated with the Gated-Vdd technique by inserting the sleep transistors in a zig-zag fashion. Sleepy stack proposed to divide the existing transistors into two half sizes and then insert sleep transistors to further reduce leakage. This approach was shown to be area-inefficient as it comes with 50 to 120% area overhead. Sleepy keeper is a variation of Gated-Vdd approach which can save logic state during sleep mode. The draw back of this approach is a significant additional dynamic power overhead compared to the base circuit.

Optimal sizing of sleep transistors to minimize the impact of sleep transistor insertion on the circuit delay has been researched extensively in [57], [58], [59] and [31]. [58] used the average current consumption of logic gates to find the size of sleep transistor for satisfying circuit speed. Their proposed technique is based on the assumption that the circuit speed is weakly dependent on the circuit operating pattern for large enough sleep transistor size.

### 7.2 Architectural techniques

A number of architecturally driven techniques have been proposed in literature to reduce leakage in different on-chip SRAM memories. Powell et al. proposed applying *gated-$V_{dd}$* approach to gate the power supply for cache lines that are not likely to be accessed [13].

Kaxiras et al. proposed a cache decay technique which reduces cache leakage by turning off cache lines not likely to be reused [19]. Flautner et al. proposed a drowsy cache which reduces the supply voltage of the L1 cache line instead of gating it off completely [21]. The advantage of this technique is that it preserves the cache line information but introduces a delay in accessing drowsy lines.

Zhang et al. proposed a compiler approach to turn off the cache lines for the region of the code that would not be accessed for a long period of time [6]. Meng et al. presented a perfecting scheme which combines the drowsy caches and the Gated-$V_{dd}$ techniques to optimize cache leakage reduction [30].

In addition to caches, there has been several works on reducing leakage in other on-chip SRAM memories. Hu et al. applied decay techniques to branch predictor, exploring strategies for spatial and temporal locality to make decay effective [25].

Many leakage optimized design techniques use low leakage, slow transistors, (with high-$V_t$) on non-critical paths [32]. However, these methods cannot be applied to critical path, such as register read path as shown to impact processor performance significantly [31, 36, 51]. Jin et al. proposed a low-leakage register file cell design exploiting the observation that physical registers have short life cycles [33]. Heo et. al. proposed segmenting register file read bitline [26]. Once an entire register file subbank is dead, the subbank read bitline is turned off, saving the leakage on the bitline. Kondo and Nakamura proposed bit-partitioned register file to reduce leakage based on the observation that many operands do not need the full-bit width of a register entry. [34]

All research mentioned above primarily targeted the leakage in the SRAM *cells*. Given the results in Figure 1, *peripheral circuits* are equally if not more important to address in SRAM memories. In addition there is a significant body of work on reducing temperature in processor [1, 2, 5, 23]. Our work considers the effect of temperature on leakage power and we report temperature reduction as well.

## 8. CONCLUSION

This paper addressed the issue of leakage power dissipation in peripheral circuits of on-chip SRAMs. It showed how to simultaneously reduce leakage and temperature in the L2, DL1 and IL1 caches, Branch Predictor, Floating Point and Integer Register Files, Floating Point and Integer Rename units, and Instruction and Data TLBs using a novel zig-zag share circuit, with minimal area and delay overheads. It proposed multiple low-power modes which differ in the bias voltage of sleep transistors used, allowing a tradeoff between leakage reduction and wakeup delay in peripheral circuits. At the architectural level a new control mechanism was proposed for switching between different low-power modes in each SRAM unit, achieving leakage and temperature reduction with minimal impact on performance. We observed a very significant average leakage reduction varying from 75% for L2 cache to 38% for IL1cache. This resulted in a reduction of up to 8.6 degree Celsius in the steady state temperature based on the interdependency of leakage and temperature.

While this work focused on single core architecture, applying zz-hvs technique to a multicore architecture for all non-shared on- core SRAM units, such as IL1 caches, Branch Predictor, Floating Point and Integer Register Files, Floating Point and Integer Rename units, and Instruction and Data TLBs is similar. For cache hierarchies that are shared between multiple cores in CMP architecture such as shared L2 or L3 caches, controlling sleep signals can be very challenging. For example it is common that in a CMP architecture the L32 cache to beis shared across multiple cores and as a result it can not be simply transitioned to various low power modes by just tracking individual core's cache miss rate information. New algorithms need to be developed to predict L2/L3 cache behavior based on all cores' cache miss information to control the sleep transistor.

**Table 7. Peak temperature for different blocks before (B) and after (A) applying the leakage reduction techniques**

| | L2 | | IL1 | | DL1 | | Branch PRED | | DTLB | | FALU | | FRF | | FMD | | FP Map | | Int Map | | IntQ | | IRF | | Int ExecU | | FPQ | | LdStQ | | ITLB | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **B** | **A** | **B** | **A** | **B** | **A** | **B** | **A** | **B** | **A** | **B** | **A** | **B** | **A** | **B** | **A** | **B** | **A** | **B** | **A** | **B** | **A** | **B** | **A** | **B** | **A** | **B** | **A** | **B** | **A** | **B** | **A** |
| ammp | 67 | 58 | 72 | 67 | 67 | 60 | 64 | 58 | 65 | 58 | 64 | 59 | 67 | 62 | 65 | 61 | 62 | 58 | 63 | 59 | 70 | 62 | 69 | 63 | 70 | 63 | 66 | 62 | 79 | 72 | 81 | 75 |
| applu | 70 | 62 | 74 | 67 | 72 | 65 | 68 | 62 | 69 | 63 | 71 | 66 | 75 | 69 | 73 | 69 | 67 | 63 | 69 | 65 | 74 | 67 | 71 | 64 | 73 | 67 | 72 | 68 | 79 | 72 | 80 | 73 |
| apsi | 68 | 62 | 80 | 76 | 74 | 69 | 69 | 66 | 70 | 66 | 71 | 67 | 74 | 71 | 71 | 68 | 68 | 65 | 69 | 66 | 76 | 71 | 78 | 73 | 78 | 74 | 73 | 70 | 87 | 82 | 89 | 85 |
| art | 78 | 71 | 79 | 74 | 75 | 69 | 72 | 67 | 73 | 67 | 71 | 67 | 72 | 68 | 70 | 67 | 67 | 63 | 68 | 64 | 83 | 79 | 78 | 74 | 80 | 76 | 74 | 71 | 92 | 87 | 89 | 84 |
| bzip2 | 68 | 62 | 86 | 83 | 75 | 70 | 74 | 70 | 71 | 67 | 66 | 63 | 67 | 63 | 66 | 63 | 67 | 64 | 67 | 64 | 77 | 72 | 89 | 85 | 83 | 79 | 69 | 66 | 93 | 88 | 98 | 94 |
| crafty | 67 | 61 | 88 | 85 | 76 | 72 | 74 | 71 | 71 | 67 | 66 | 63 | 67 | 64 | 66 | 63 | 68 | 65 | 68 | 65 | 77 | 73 | 92 | 89 | 85 | 81 | 70 | 67 | 94 | 90 | 100 | 97 |
| eon | 67 | 62 | 88 | 85 | 77 | 73 | 74 | 71 | 72 | 68 | 69 | 66 | 72 | 68 | 69 | 66 | 68 | 65 | 68 | 65 | 77 | 73 | 83 | 80 | 82 | 79 | 72 | 69 | 92 | 88 | 98 | 94 |
| equake | 66 | 61 | 86 | 83 | 75 | 71 | 72 | 69 | 70 | 67 | 65 | 62 | 66 | 63 | 65 | 62 | 67 | 64 | 67 | 64 | 75 | 71 | 90 | 87 | 82 | 79 | 69 | 66 | 92 | 88 | 97 | 94 |
| facerec | 68 | 62 | 79 | 74 | 73 | 68 | 70 | 66 | 70 | 65 | 71 | 67 | 74 | 70 | 71 | 67 | 67 | 64 | 69 | 65 | 76 | 71 | 79 | 74 | 78 | 73 | 72 | 69 | 86 | 81 | 88 | 83 |
| galgel | 67 | 62 | 87 | 85 | 75 | 70 | 72 | 69 | 70 | 66 | 73 | 70 | 79 | 76 | 76 | 73 | 69 | 67 | 71 | 68 | 77 | 73 | 76 | 73 | 80 | 77 | 76 | 73 | 91 | 87 | 98 | 95 |
| gap | 67 | 61 | 82 | 77 | 74 | 69 | 71 | 67 | 70 | 65 | 66 | 62 | 67 | 63 | 66 | 63 | 67 | 64 | 67 | 63 | 75 | 70 | 88 | 84 | 82 | 78 | 68 | 65 | 89 | 84 | 93 | 88 |
| gcc | 70 | 63 | 86 | 82 | 77 | 72 | 74 | 70 | 73 | 68 | 67 | 63 | 68 | 64 | 66 | 63 | 65 | 62 | 66 | 62 | 79 | 74 | 84 | 79 | 83 | 78 | 71 | 68 | 95 | 90 | 98 | 93 |
| gzip | 67 | 62 | 88 | 85 | 75 | 71 | 74 | 71 | 71 | 67 | 66 | 63 | 67 | 64 | 66 | 63 | 68 | 65 | 68 | 65 | 77 | 73 | 91 | 88 | 84 | 81 | 70 | 67 | 95 | 90 | 100 | 96 |
| lucas | 69 | 62 | 71 | 64 | 69 | 61 | 66 | 61 | 67 | 61 | 69 | 64 | 70 | 65 | 69 | 65 | 65 | 61 | 66 | 62 | 72 | 65 | 71 | 63 | 71 | 64 | 69 | 65 | 76 | 68 | 75 | 68 |
| mcf | 68 | 60 | 68 | 61 | 67 | 60 | 65 | 59 | 66 | 59 | 65 | 60 | 65 | 60 | 64 | 60 | 63 | 59 | 63 | 59 | 71 | 62 | 69 | 61 | 69 | 62 | 65 | 61 | 72 | 64 | 72 | 64 |
| mesa | 66 | 61 | 84 | 80 | 75 | 71 | 71 | 68 | 71 | 67 | 68 | 65 | 70 | 67 | 68 | 65 | 68 | 65 | 68 | 65 | 75 | 71 | 86 | 82 | 81 | 77 | 71 | 68 | 90 | 86 | 95 | 91 |
| mgrid | 68 | 61 | 73 | 66 | 71 | 63 | 68 | 62 | 69 | 62 | 71 | 66 | 73 | 68 | 69 | 65 | 66 | 62 | 67 | 63 | 73 | 66 | 70 | 63 | 72 | 66 | 71 | 67 | 77 | 70 | 77 | 70 |
| parser | 68 | 62 | 85 | 81 | 74 | 69 | 73 | 69 | 71 | 66 | 67 | 63 | 67 | 63 | 66 | 63 | 67 | 64 | 67 | 64 | 77 | 71 | 88 | 83 | 83 | 78 | 69 | 66 | 92 | 86 | 96 | 91 |
| perlbmk | 69 | 62 | 85 | 82 | 75 | 70 | 74 | 71 | 72 | 67 | 67 | 63 | 67 | 63 | 67 | 63 | 66 | 63 | 67 | 63 | 77 | 71 | 85 | 81 | 83 | 78 | 68 | 65 | 91 | 85 | 95 | 90 |
| sixtrack | 66 | 61 | 85 | 82 | 75 | 71 | 72 | 69 | 70 | 66 | 73 | 70 | 79 | 76 | 76 | 73 | 69 | 67 | 72 | 69 | 76 | 73 | 76 | 73 | 79 | 76 | 76 | 73 | 91 | 87 | 95 | 92 |
| swim | 69 | 61 | 72 | 65 | 70 | 62 | 67 | 61 | 68 | 61 | 69 | 64 | 71 | 66 | 69 | 65 | 65 | 61 | 66 | 62 | 73 | 66 | 71 | 64 | 72 | 65 | 69 | 65 | 78 | 70 | 76 | 69 |
| twolf | 65 | 60 | 78 | 75 | 69 | 65 | 67 | 65 | 66 | 63 | 64 | 61 | 65 | 62 | 64 | 61 | 64 | 62 | 65 | 62 | 71 | 67 | 77 | 74 | 73 | 70 | 67 | 64 | 82 | 78 | 86 | 83 |
| vortex | 67 | 62 | 84 | 80 | 76 | 72 | 72 | 69 | 72 | 68 | 66 | 63 | 66 | 63 | 65 | 62 | 66 | 63 | 66 | 63 | 76 | 71 | 86 | 83 | 82 | 78 | 69 | 66 | 90 | 86 | 94 | 90 |
| vpr | 69 | 62 | 78 | 73 | 73 | 67 | 71 | 66 | 70 | 65 | 67 | 63 | 68 | 64 | 67 | 63 | 66 | 62 | 66 | 63 | 75 | 68 | 80 | 75 | 78 | 73 | 69 | 65 | 85 | 78 | 86 | 81 |
| wupwise | 67 | 61 | 82 | 77 | 73 | 68 | 71 | 67 | 69 | 64 | 70 | 66 | 72 | 68 | 70 | 66 | 68 | 64 | 69 | 65 | 75 | 70 | 80 | 75 | 79 | 74 | 72 | 69 | 87 | 82 | 92 | 87 |
| **Ave.** | 68 | 62 | 81 | 76 | 73 | 68 | 71 | 67 | 70 | 65 | 68 | 64 | 70 | 66 | 68 | 65 | 66 | 63 | 67 | 64 | 75 | 70 | 80 | 76 | 79 | 74 | 70 | 67 | 87 | 82 | 90 | 85 |

**Table 8. Reduction in Peak(P) and Steady-State (S) Temperature in Different Units**

| | L2 | | IL1 | | DL1 | | Branch PRED | | DTLB | | FALU | | FRF | | FMD | | FP Map | | Int Map | | IntQ | | IRF | | Int ExecU | | FPQ | | LdStQ | | ITLB | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **P** | **S** | **P** | **S** | **P** | **S** | **P** | **S** | **P** | **S** | **P** | **S** | **P** | **S** | **P** | **S** | **P** | **S** | **P** | **S** | **P** | **S** | **P** | **S** | **P** | **S** | **P** | **S** | **P** | **S** | **P** | **S** |
| ammp | 8.2 | 8.6 | 5.8 | 7.5 | 6.7 | 7.8 | 5.8 | 6.3 | 7.0 | 7.5 | 5.0 | 5.3 | 4.5 | 5.7 | 4.1 | 4.6 | 3.8 | 4.2 | 4.0 | 4.3 | 7.2 | 9.1 | 6.2 | 7.8 | 3.8 | 4.2 | 4.0 | 4.3 | 7.2 | 9.1 | 6.2 | 7.8 |
| applu | 8.1 | 8.5 | 7.4 | 7.3 | 7.2 | 7.7 | 5.7 | 6.1 | 6.9 | 7.3 | 4.6 | 4.8 | 5.1 | 5.3 | 4.1 | 4.4 | 3.7 | 4 | 3.9 | 4.2 | 7.0 | 7.5 | 6.9 | 7.3 | 3.7 | 4 | 3.9 | 4.2 | 7.0 | 7.5 | 6.9 | 7.3 |
| apsi | 6.0 | 7.2 | 4.1 | 4.8 | 4.9 | 5.9 | 3.7 | 4.6 | 4.6 | 5.7 | 3.5 | 4.0 | 3.7 | 4.1 | 3.2 | 3.6 | 3.0 | 3.4 | 3.1 | 3.5 | 5.0 | 5.9 | 4.3 | 5.2 | 3.0 | 3.4 | 3.1 | 3.5 | 5.0 | 5.9 | 4.3 | 5.2 |
| art | 6.9 | 7.4 | 5.3 | 5.2 | 6.0 | 6.5 | 4.4 | 4.8 | 5.7 | 6.1 | 3.5 | 3.8 | 3.8 | 4.0 | 3.3 | 3.6 | 3.2 | 3.4 | 3.3 | 3.5 | 4.6 | 4.9 | 4.9 | 5.3 | 3.2 | 3.4 | 3.3 | 3.5 | 4.6 | 4.9 | 4.9 | 5.3 |
| bzip2 | 5.9 | 7.0 | 3.9 | 4.5 | 4.7 | 5.6 | 3.5 | 4.4 | 4.4 | 5.4 | 3.4 | 3.8 | 3.6 | 3.9 | 3.1 | 3.6 | 3.0 | 3.3 | 3.0 | 3.4 | 4.7 | 5.6 | 4.0 | 4.9 | 3.0 | 3.3 | 3.0 | 3.4 | 4.7 | 5.6 | 4.0 | 4.9 |
| crafty | 5.3 | 6.5 | 3.1 | 3.7 | 4.1 | 5.0 | 3.1 | 3.7 | 3.8 | 4.8 | 3.1 | 3.4 | 3.2 | 3.5 | 2.9 | 3.3 | 2.8 | 3.1 | 2.8 | 3.2 | 4.1 | 5.0 | 3.4 | 4.1 | 2.8 | 3.1 | 2.8 | 3.2 | 4.1 | 5.0 | 3.4 | 4.1 |
| eon | 5.4 | 6.4 | 3.3 | 3.7 | 4.1 | 5.0 | 3.1 | 3.7 | 3.9 | 4.8 | 3.1 | 3.5 | 3.2 | 3.6 | 2.9 | 3.2 | 2.8 | 3.1 | 2.9 | 3.1 | 4.2 | 5.1 | 3.5 | 4.1 | 2.8 | 3.1 | 2.9 | 3.1 | 4.2 | 5.1 | 3.5 | 4.1 |
| equake | 4.9 | 6.5 | 3.3 | 3.8 | 3.8 | 5.1 | 3.0 | 3.8 | 3.6 | 4.9 | 2.9 | 3.4 | 3.1 | 3.5 | 2.8 | 3.2 | 2.8 | 3.1 | 2.8 | 3.1 | 3.6 | 4.6 | 3.3 | 4.1 | 2.8 | 3.1 | 2.8 | 3.1 | 3.6 | 4.6 | 3.3 | 4.1 |
| facerec | 6.6 | 7.8 | 5.4 | 6.0 | 5.6 | 6.7 | 4.3 | 5.3 | 5.3 | 6.5 | 3.7 | 4.2 | 3.9 | 4.6 | 3.4 | 3.9 | 3.2 | 3.7 | 3.3 | 3.8 | 5.1 | 6.0 | 5.0 | 6.1 | 3.2 | 3.7 | 3.3 | 3.8 | 5.1 | 6.0 | 5.0 | 6.1 |
| galgel | 5.3 | 6.5 | 2.9 | 3.8 | 4.1 | 5.2 | 3.1 | 3.9 | 3.9 | 5.0 | 3.0 | 3.5 | 3.1 | 3.5 | 2.8 | 3.3 | 2.8 | 3.1 | 2.8 | 3.2 | 3.9 | 4.7 | 3.4 | 4.2 | 2.8 | 3.1 | 2.8 | 3.2 | 3.9 | 4.7 | 3.4 | 4.2 |
| gap | 6.1 | 7.4 | 4.3 | 5.2 | 5.1 | 6.2 | 3.8 | 4.8 | 4.8 | 5.9 | 3.6 | 4.0 | 3.9 | 4.3 | 3.3 | 3.7 | 3.1 | 3.5 | 3.2 | 3.6 | 5.1 | 6.2 | 4.6 | 5.5 | 3.1 | 3.5 | 3.2 | 3.6 | 5.1 | 6.2 | 4.6 | 5.5 |
| gcc | 6.5 | 7.2 | 4.3 | 4.9 | 5.5 | 6.0 | 4.1 | 4.6 | 5.1 | 5.7 | 3.6 | 3.9 | 3.7 | 4.1 | 3.4 | 3.7 | 3.1 | 3.4 | 3.2 | 3.5 | 5.0 | 5.9 | 4.6 | 5.2 | 3.1 | 3.4 | 3.2 | 3.5 | 5.0 | 5.9 | 4.6 | 5.2 |
| gzip | 5.5 | 6.5 | 3.3 | 3.7 | 4.3 | 5.0 | 3.2 | 3.9 | 4.0 | 4.8 | 3.2 | 3.5 | 3.3 | 3.7 | 3.0 | 3.3 | 2.9 | 3.1 | 2.9 | 3.2 | 4.5 | 5.3 | 3.6 | 4.2 | 2.9 | 3.1 | 2.9 | 3.2 | 4.5 | 5.3 | 3.6 | 4.2 |
| lucas | 7.6 | 8.4 | 7.7 | 7.2 | 7.2 | 7.6 | 5.7 | 6.1 | 6.8 | 7.2 | 4.7 | 4.9 | 5.1 | 5.4 | 4.2 | 4.5 | 3.8 | 4.0 | 3.9 | 4.2 | 7.7 | 8.1 | 7.2 | 7.4 | 3.8 | 4.0 | 3.9 | 4.2 | 7.7 | 8.1 | 7.2 | 7.4 |
| mcf | 8.3 | 8.6 | 7.4 | 7.5 | 7.6 | 7.9 | 6.0 | 6.2 | 7.1 | 7.5 | 4.9 | 5.1 | 5.3 | 5.7 | 4.3 | 4.6 | 3.8 | 4.1 | 4.0 | 4.3 | 8.2 | 8.5 | 7.5 | 7.7 | 3.8 | 4.1 | 4.0 | 4.3 | 8.2 | 8.5 | 7.5 | 7.7 |
| mesa | 5.4 | 6.9 | 3.8 | 4.3 | 4.3 | 5.5 | 3.3 | 4.2 | 4.1 | 5.3 | 3.2 | 3.6 | 3.3 | 3.8 | 3.0 | 3.4 | 2.9 | 3.2 | 2.9 | 3.3 | 4.2 | 5.3 | 3.7 | 4.7 | 2.9 | 3.2 | 2.9 | 3.3 | 4.2 | 5.3 | 3.7 | 4.7 |
| mgrid | 7.9 | 8.4 | 6.9 | 7.2 | 7.1 | 7.7 | 5.6 | 6.1 | 6.8 | 7.3 | 4.6 | 4.9 | 5.1 | 5.3 | 4.2 | 4.5 | 3.8 | 4.0 | 3.9 | 4.2 | 7.3 | 7.8 | 6.8 | 7.4 | 3.8 | 4.0 | 3.9 | 4.2 | 7.3 | 7.8 | 6.8 | 7.4 |
| parser | 6.4 | 7.3 | 4.6 | 5.1 | 5.1 | 6.1 | 4.0 | 4.7 | 4.9 | 5.8 | 3.7 | 4.2 | 3.8 | 4.4 | 3.4 | 3.8 | 3.1 | 3.5 | 3.2 | 3.6 | 5.6 | 6.6 | 4.5 | 5.5 | 3.1 | 3.5 | 3.2 | 3.6 | 5.6 | 6.6 | 4.5 | 5.5 |
| perlbmk | 6.6 | 7.1 | 4.2 | 4.7 | 5.1 | 5.8 | 3.9 | 4.5 | 5.1 | 5.5 | 3.8 | 4.0 | 4.0 | 4.2 | 3.4 | 3.6 | 3.1 | 3.4 | 3.2 | 3.5 | 5.7 | 6.2 | 4.6 | 5.1 | 3.1 | 3.4 | 3.2 | 3.5 | 5.7 | 6.2 | 4.6 | 5.1 |
| sixtrack | 5.1 | 6.5 | 3.1 | 3.7 | 4.0 | 5.1 | 3.0 | 3.8 | 3.7 | 4.9 | 3.0 | 3.3 | 3.1 | 3.4 | 2.8 | 3.2 | 2.8 | 3.0 | 2.8 | 3.1 | 3.7 | 4.6 | 3.3 | 4.0 | 2.8 | 3.o | 2.8 | 3.1 | 3.7 | 4.6 | 3.3 | 4.0 |
| swim | 8.4 | 8.7 | 7.7 | 7.5 | 7.6 | 7.9 | 6.0 | 6.3 | 7.2 | 7.5 | 4.8 | 5.0 | 5.3 | 5.5 | 4.3 | 4.5 | 3.9 | 4.1 | 4.0 | 4.3 | 7.5 | 7.8 | 7.3 | 7.5 | 3.9 | 4.1 | 4.0 | 4.3 | 7.5 | 7.8 | 7.3 | 7.5 |
| twolf | 4.7 | 6.5 | 3.2 | 3.7 | 3.7 | 5.1 | 2.9 | 3.9 | 3.5 | 4.9 | 3.0 | 3.6 | 3.1 | 3.6 | 2.8 | 3.4 | 2.7 | 3.2 | 2.7 | 3.2 | 4.0 | 5.6 | 3.1 | 4.3 | 2.7 | 3.2 | 2.7 | 3.2 | 4.0 | 5.6 | 3.1 | 4.3 |
| vortex | 5.4 | 6.7 | 3.5 | 4.0 | 4.2 | 5.3 | 3.2 | 4.0 | 4.0 | 5.0 | 3.2 | 3.7 | 3.3 | 3.8 | 3.0 | 3.4 | 2.9 | 3.2 | 2.9 | 3.2 | 4.4 | 5.5 | 3.6 | 4.5 | 2.9 | 3.2 | 2.9 | 3.2 | 4.4 | 5.5 | 3.6 | 4.5 |
| vpr | 7.0 | 7.5 | 5.3 | 5.4 | 5.9 | 6.3 | 4.5 | 4.9 | 5.6 | 6.0 | 4.1 | 4.3 | 4.3 | 4.6 | 3.7 | 3.9 | 3.3 | 3.6 | 3.5 | 3.8 | 6.6 | 7.2 | 5.4 | 5.9 | 3.3 | 3.6 | 3.5 | 3.8 | 6.6 | 7.2 | 5.4 | 5.9 |
| wupwise | 6.3 | 7.6 | 5.2 | 5.6 | 5.3 | 6.4 | 4.1 | 5.0 | 5.0 | 6.2 | 3.7 | 4.2 | 3.9 | 4.5 | 3.4 | 3.8 | 3.2 | 3.6 | 3.2 | 3.7 | 5.2 | 6.4 | 4.8 | 5.9 | 3.2 | 3.6 | 3.2 | 3.7 | 5.2 | 6.4 | 4.8 | 5.9 |
| **Ave.** | 6.3 | 7.3 | 4.7 | 5.2 | 5.5 | 6.2 | 4.0 | 4.8 | 4.9 | 5.9 | 3.6 | 4.1 | 3.7 | 4.3 | 3.3 | 3.8 | 3.1 | 3.5 | 3.2 | 3.6 | 5.0 | 6.2 | 4.5 | 5.5 | 3.1 | 3.5 | 3.2 | 3.6 | 5.0 | 6.2 | 4.5 | 5.5 |

## 9. REFERENCES

[1] I. Koren et al. Temperature Aware Floorplanning, *in TACS 2005*.

[2] Y. Nakagome et al. Review and future prospects of low-voltage RAM circuits, IBM Journal of R&D'03.

[3] H. Homayoun et al., ZZ-HVS: Zig-Zag Horizontal and Vertical Sleep Transistor Sharing to Reduce Leakage Power in On-Chip SRAM Peripheral Circuits. In Proc. IEEE International Conference on Computer Design, ICCD, 2008.

[4] C. H. Kim et al,. A forward body-biased low-leakage SRAM cache: device, circuit and architecture considerations. *TVLSI-2005.*

[5] J. Donald and Margaret Martonosi, Techniques for Multicore Thermal Management, Classification and New Exploration. ISCA 2006.

[6] W. Zhang and J. S. Hu. Compiler-directed instruction cache leakage optimization. In Proc. In *MICRO-35*, 2002.

[7] J. M. Rabaey et al., Digital integrated circuits: a design perspective, Prentice Hall, Second. Edition, 2003.

[8] Y. Takeyama et al,. A Low Leakage SRAM Macro with Replica Cell Biasing Scheme. *IEEE Journal Of Solid- State Circuits*, 2006.

[9] J. Kao, S. Narendra, and A. Chandrakasan,"MTCMOS hierarchical sizing based on mutual exclusive discharge patterns," *DAC*, 1998.

[10] J. C. Park, V. J. Mooney III. Sleepy stack leakage reduction. IEEE Trans. VLSI Syst. 14(11): 1250-1263 (2006).

[11] SimpleScalar4 tutorial, http://www.simplescalar.com/tutorial.html.

[12] K. Nii et al., A 90-nm low-power 32 KByte embedded SRAM with gate leakage suppression circuit for mobile applications, *ISSCC 2004*.

[13] M.D. Powell et al,. Gated $V_{dd}$: A circuit technique to reduce leakage in deep-submicron cache memories. in *ISLPED*, 2000 .

[14] A. Agarawal et al., DRG-Cache: A data retention gated-ground cache for low Power, *DAC* 2002.

[15] K. Agarwal, H. Deogun, D. Sylvester, K. Nowka. Power gating with multiple sleep modes. In *ISQED* 2006.

[16] K. Nii, et al. A low power SRAM using auto-backgate-controlled MT-CMOS. In *ISLPED*, 1998, pp. 293-298.

[17] M. Mamidipaka, et al, Analytical models for leakage power estimation of memory array structures. IEEE *CODES+ISSS*, 2004.

[18] B S. Amrutur et al,. Speed and power scaling of SRAMs, *IEEE Journal of Solid State Circuits*. Feb 2000, vol. 35.

[19] S. Kaxiras et al,. Cache decay: exploiting generational behavior to reduce cache leakage power. *IEEE-ISCA, 2001*.

[20] B.S. Amrutur, et al., A replica technique for wordline and sense control in low-power SRAM's, *IJSSC*, vol. 2000.

[21] K. Flautner et al,. Drowsy caches: simple techniques for reducing leakage power. *IEEE ISCA*, 2002.

[22] . Thoziyoor, N. Muralimanohar, J. H. Ahn, and N P. Jouppi "CACTI 5.1 Technical Report" HP Laboratories, Palo Alto, April 2, 2008.

[23] H. Homayoun, A. Gupta, A. Veidenbaum, F. J. Kurdahi, N. Dutt , RELOCATE: Register File Local Access Pattern Redistribution Mechanism for Power and Thermal Management in Out-of-Order Embedded Processor., HiPeac 2010.

[24] J.L. Cruz, A. González, et al., "Multiple-banked register file architectures", in *ISCA* 2000.

[25] Z. Hu et al., "Applying Decay Strategies to Branch Predictors for Leakage Energy Savings," Proc. ICCD 2002.

[26] S. Heo, K. Barr, M. Hampton, and K. Asanovic, "Dynamic fine-grain leakage reduction using leakage-biased bitlines," the 30th International Symposium on Computer Architecture, 2003.

[27] L. He et al.,"Considering the Interdependence of Temperature andLeakage Interdependence of Temperature and Leakage," in DAC 2004.

[28] K. Skadron et al., "Temperature-aware microarchitecture," in Proc. ACM ISCA, pp. 2–13, Jun. 2003.

[29] S. Rusu et al,. A 65-nm Dual-Core Multithreaded Xeon® Processor With 16-MB L3 Cache, *IJSSC 2007.*

[30] Y. Meng, T. Sherwood, and R. Kastner. On the limits of leakage power reduction in caches. In *HPCA-11*, 2005.

[31] Ramalingam, A., Zhang, B., Davgan, A., AND Pan, D., "Sleep Transistor Sizing Using Timing Criticality and Temporal Currents," ASP-DAC 2005.

[32] S. Tang, et al. A leakage-tolerant dynamic register file using leakage bypass with stack forcing (LBSF) and source follower NMOS (SFN) techniques, Symposium on VLSI Circuits, 2002.

[33] L. Jin et al., Reduce Register Files Leakage Through Discharging Cells, in *ICCD* 2006.

[34] M. Kondo and H. Nakamura, A Small, Fast and Low-Power Register File by Bit-Partitioning, in *HPCA* 2005.

[35] D. Brooks, V. Tiwari and M. Martonosi. "Wattch: A framework for architectural-level power analysis and optimizations." in *ISCA* 2000.

[36] H. Homayoun, et al. "Improving performance and reducing energy-delay with adaptive resource resizing for out-of-order embedded processors", LCTES 2008.

[37] G. Gerosa et al., A Sub-lW to 2W Low-Power IA Processor for Mobile Internet Devices and Ultra-Mobile PCs in 45nm Hi-K Metal Gate CMOS, In ISSCC-2008.

[38] K.-S. Min et al., "Zigzag super cut-off CMOS (ZSCCMOS) block activation with self-adaptive voltage level controller: an alternative to clock-gating scheme in leakage dominant era," ISSCC 2003.

[39] M. Horiguchi et al., "Switched-source-impedence CMOS circuit for low-standby subthreshold current giga-scale LSI's," , VLSI circuits Dig. 1993

[40] Y. Wang et al., "A 1.1GHz 12µA/Mb-Leakage SRAM Design in 65nm Ultra-Low-Power CMOS with Integrated Leakage Reduction For Mobile Applications.", ISSCC Dig. Tech. Papers, Feb. 2007.

[41] E. Grossara et al., Statistically Aware SRAM Memory Array Design, in International Symposium on Quality Electronic Design, ISQED-2006.

[42] A. Gupta, et al., STEFAL: A System Level Temperature- and Floorplan-Aware Leakage Power Estimator for SoCs. In VLSID 2007.

[43] Semiconductor Industries Association, "International Technology Roadmap for Semiconductors," 2005, http://www.itrs.net/.

[44] D. Chinnery "Closing the Power Gap Between ASIC & Custom, Tools and Techniques for Low Power Design", 2007, XII, ISBN: 978-0-387-25763-1.

[45] S. H. Kim and V. J. Mooney, Sleepy keeper: a new approach to low-leakage power VLSI design. *VLSI-SoC* 2006

[46] Bias generator for body bias, US Patent 7164307, Vivek De et al.

[47] J. Tschanz, Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage., JSSC'02

[48] M. Yamaoka1,et al., A 65nm Low-Power High-Density SRAM Operable at 1.0V Under 3σ Systematic Variation Using Separate Vth Monitoring and Body Bias for NMOS and PMOS, ISSCC'08.

[49] T. Kuroda et al., Leakage in Nanometer CMOS Technologies. Springer'06.

[50] Design Compiler, Synopsys Incorporation.

[51] H. Homayoun et al, "Dynamic register file resizing and frequency scaling to improve embedded processor performance and energy-delay efficiency". in *DAC* 2008.

[52] H. Li, et al. "VSV: L2-miss-driven variable supply-voltage scaling for low power." In *MICRO* 2003.

[53] M. Powell et al,. Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories. *ISLPED* 2000.

[54] H. Homayoun et al., Multiple Sleep Modes Leakage Control in Peripheral Circuits of a All Major SRAM-Based Processor Units, Technical Report 08-09, University of California Irvine.

[55] Min, K.-S., et al., Zigzag super cut-off CMOS (ZSCCMOS) block activation with self-adaptive voltage level controller: an alternative to clock-gating scheme in leakage dominant era, ISSCC 2003..

[56] Horiguchi, M., et al., Switched-source-impedence CMOS circuit for low-standby subthreshold current giga-scale LSI's, IJSSC, 1993.

[57] Calhoun, B. H. et al. Design methodology for fine-grained leakage control in MTCMOS," in ISLPED 2003.

[58] Mutoh, S., et al, Design method of MTCMOS power switch for lowvoltage high-speed LSIs," ASP-DAC 1999.

[59] Khandelwal, V., AND Srivastava, A., Leakage control through fine-grained placement and sizing of sleep transistors, in Proc.ICCAD 2004.