Width-Aware Fine-Grained Dynamic Supply Gating: A Design Methodology for Low-Power Datapath and Memory

Lei Wang, Somnath Paul and Swarup Bhunia
Department of Electrical Engg. & Computer Science
Case Western Reserve University
Cleveland, USA
Email: {lxw185, sxp190, skb21}@case.edu

***Abstract*—With increasing contribution of leakage in total active power, run-time leakage control techniques are becoming extremely important. Supply gating provides an effective, low-overhead and technology scalable approach for active leakage reduction through the well-known "stacking effect". However, conventional supply gating approaches are typically coarse-grained in both space and time - i.e. are applied to large datapath or memory blocks when an entire logic/memory block is idle for sufficiently long period. They suffer from limited applicability at run time. On the other hand, fine-grained supply gating is constrained primarily by the large wake-up delay and wake-up power overhead. In this paper, we propose a novel fine-grained width-aware dynamic supply gating (WADSG) approach to reduce both active leakage and redundant switching power in datapath and embedded memory (e.g. L1/L2 cache). The approach exploits the abundance of narrow-width (NW) operands in general-purpose and embedded applications to "supply-gate" unused parts of integer execution units and memory blocks while they are in use. We introduce a novel levelized gating strategy to virtually eliminate the wake-up delay overhead. We employ the proposed WADSG approach to a superscalar processor. To reduce the wake-up power we use a width-aware instruction issue policy. In case of L1 and L2 cache, we store the width information per "ways" of associative cache and supply-gate the most significant bits of the NW ways. We also propose a width-aware block allocation and replacement policy to maximize the number of NW ways. Simulation results for 45nm technology with Spec2k benchmarks show major savings (*34.5%*) in total processor power (considering both switching and active leakage power) with *no* performance impact. As a by-product, the proposed scheme also improves the thermal profile of both datapath and memory.**

***Keywords*-Narrow-width operands, Dynamic Supply Gating, Processor Datapath, Cache, Width-Aware Issue**

I. Introduction

Power consumption has emerged as a primary design constraint for integrated circuits (ICs) in both general-purpose and embedded applications [1-2]. The active power in an IC comprises of switching power and active leakage in logic and memory circuits. In the nanometer technology regime, leakage power has become a major component of total power [1] despite breakthrough advances in device technologies, such high-K metal gate devices. Due to exponential dependence of subthreshold leakage (considered to be the largest leakage component) to temperature, leakage current increases significantly at run time in the high-activity regions of a chip e.g. datapath. Active leakage in datapath has introduced both power and power-density induced reliability concerns. Moreover, active leakage in large on-chip memory (e.g. processor caches) can be very high due to large number of memory cells present in an embedded memory. Fig. 1(a) and 1(b) show the relative magnitude of leakage and dynamic power across different technology nodes, for a processor datapath (64-bit logarithmic adder) and cache (2MB), respectively, with predictive technology models (PTM) [15]. We observe that for both datapath (Fig. 1(a)) and memory (Fig. 1(b)) active leakage plays a major role in total power. Hence, it is extremely important to employ run-time leakage control techniques in datapath and embedded memory.

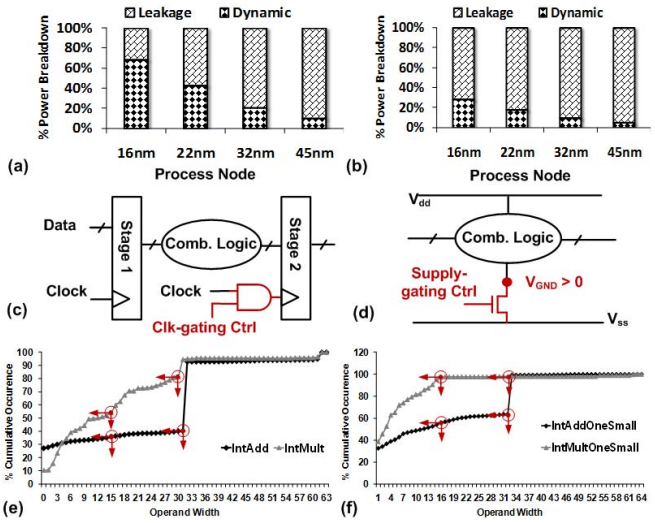


Fig. 1. Percentages of switching and leakage power for: a) functional units and b) on-chip cache in a processor across technology nodes; c) clock gating of sequential elements; d) supply-gating of combinational logic; e) cumulative value of narrow width (NW) operations with both operands NW; and f) only one operand NW for a set of benchmark applications.

Power gating has been widely explored earlier [7, 9, 14] and used in practice as an effective run-time power management technique. Power gating comes in two forms: 1) clock gating and 2) supply gating, as illustrated in Fig. 1(c) and (d). A majority of exiting power gating approach focus on clock gating to reduce active power in datapath modules. They work by "gating" the clock to select registers, thereby preventing the dataflow across pipe stages. While clock gating has been shown to be generally effective to reduce switching power as a coarse-grained power-gating approach, it cannot be effectively used for leakage control in datapath. Besides, they cannot be applied to memory. On the other hand, supply gating (Fig. 1(d)) has been shown to be highly effective to reduce standby leakage in both datapath and memory due to the well-known "stacking effect". Use of supply gating for active leakage reduction faces two major challenges: *1) identifying the idle periods for a functional unit (FU); and 2) minimizing the wake-up delay/power overhead for the gated FU.* While the clock gating circuit typically imposes minimal delay and power overhead for switching between gated and ungated modes,

We acknowledge Intel Corporation for funding part of the research presented in this work.

1063-9667/12 $26.00 © 2012 IEEE
DOI 10.1109/VLSID.2012.94

supply gating can introduce large overhead in delay and power due to gating logic. Two classes of dynamic supply gating solutions have been explored to address the above challenges: 1) a coarse (block-level) gating strategy that chooses to "gate" an entire FU [4] [13]; and 2) leakage-aware synthesis strategy [3] that requires re-design of the datapath. The first class of approaches miss the opportunity of spatial as well as temporal fine-grained gating of a logic block, while the second class suffers from scalability to large design and incurs large area overhead. Supply gating and its variants (e.g. source biasing) has also been explored for leakage reduction in embedded memory [12]. However, they typically suffer from considerable power and performance overhead due to the gating logic. Besides, they cannot take advantage of fine-grained gating (e.g. within a subarray and a words of a subarray).

In this paper, we propose a fine-grained (in both spatial and temporal sense) dynamic supply gating approach that can be employed to both datapath and embedded memory . It can lead to drastic reduction in active leakage and redundant switching power. The proposed approach is based on the observation that most of the operands in general purpose applications are narrow-width (NW) [14] i.e. the higher bits of the operand (64 to 32 or 32 to 16) are 0, while the lower bits are non-zero. Fig. 1(e) shows the occurrence of NW operands in SPEC2KInt benchmarks compiled for 64-b Alpha and simulated using Simplescalar toolset [6] with reference data. From Fig. 1(e), we note that for integer addition, *36%* and *93%* of the operations have both operands less than and equal to 16 and 32 bits, respectively. For integer multiplication, the percentages are *58%* and *81%*. Fig. 1(f) shows the cumulative occurrence of *minimum operand width* which further establishes that the scenario of one NW operand is more frequent than when both operands are NW. We exploit this abundance of NW operands to develop a width-aware dynamic supply gating (*WADSG*) methodology which supply gates unused logic gates(cells) in datapath (memory). In particular, the key technical contributions of this work are as follows:

1. We propose a methodology for fine-grained (in space and time) active power reduction for both datapath and embedded memory by exploiting the abundance of NW operations in general-purpose and embedded applications. The proposed approach reduces both active leakage as well as redundant switching power. We provide extensive analysis on the effectiveness of the proposed width-aware dynamic supply gating approach in a 64-bit superscalar processor by employing WADSG to both processor execution units and L1/L2 cache.

2. The key challenges with DSG are identified as a) hiding wake-up delay; b) minimizing the wake-up power; and c) minimizing the width computation and storage overhead. To address these challenges, we follow a circuit-architecture codesign approach. First, we propose a novel levelized gating strategy to hide the wake-up delay of the gating transistors. Instead of using a single shared gating transistor, it uses multiple ones across logic levels which masks the wakeup latency by the logic propagation delay. Second, we introduce width-aware allocation policies, where operations are allocated to the FUs in a manner that minimizes the wakeup overhead while maximizing the leakage saving. Third, we propose low-overhead circuit/microarchitectural modifications to minimize the width computation overhead.

3. We extend the concept of *WADSG* to achieve both dynamic and leakage power savings for on-chip caches. We exploit the associativity of processor caches to store the width information per "way" of a cache, instead of each block/word. This minimizes the overhead associated with wakeup as well as storing width information. Furthermore, we propose a width-aware way allocation scheme which tries to increase the number of NW ways.

4. We analyze the impact of the proposed scheme on system power saving. We also show that, as a by-product, the proposed approach can improve the die thermal profile by reducing the datapath power density. Finally, we show that WADSG can be applied to floating point datapath and can be extended to embedded domains, which also show abundance of NW operands.

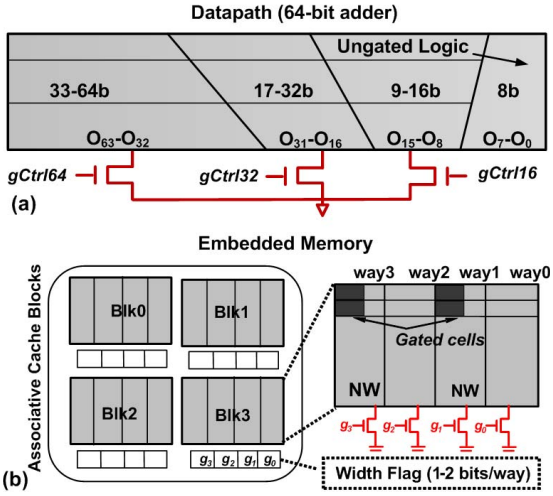


Fig. 2. a) Application of WADSG to processor execution units; b) application of WADSG to cache blocks.

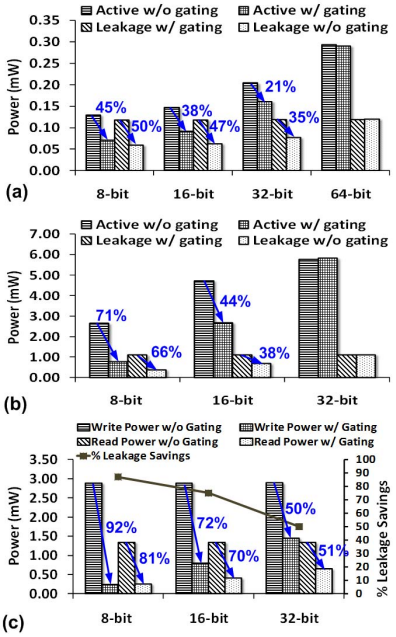


Fig. 3. Power consumption in a) 64-b integer adder and b) 32-b integer multiplier with and without width-aware supply gating; c) Read, write and leakage power saving in a 2Kb SRAM array with width-aware gating.

II. Width-Aware Supply Gating

A. Overall Approach

Fig. 2 illustrates the proposed gating approach. The entire logic for a datapath is partitioned into different blocks corresponding to

different output bits. A levelized gating approach is then applied to each block as shown in Fig. 2(a). Based on 1-2 bits which denote the width of the input operands to the datapath, one or more blocks are "waked-up" in a levelized manner to minimize the wake-up overhead. The approach is applied to cache blocks by assigning 1-2 status bit(s) to each cache way to indicate if they are narrow width (NW) or full width (FW) (Fig. 2(b)). In the following sections, we describe the circuit/architecture co-design approaches to realize the *WADSG* scheme for a conventional processor pipeline.

B. Impact on Power Saving

1) Functional Units: Fig. 3(a) and (b) present the power saving results for a 64-b integer logarithmic adder and a 32-b Wallace tree multiplier through *WADSG* scheme with 45nm PTM [15]. Active power was measured by simulating for 1K input random vectors for a clock frequency of 1GHz with 10% input activity. The delay was measured by sensitizing the circuit critical timing path. From Fig. 3(a) we note that for the integer adder, WADSG can achieve average savings of *35%* and *44%* in dynamic and leakage power, while for the multiplier, savings are higher *57%* and *52%*, respectively.

2) L1 and L2 Data Caches: Power consumed in reading and writing NW words to the data array of on-chip caches can be substantially minimized through *WADSG*. Unused bit locations in a word can be also supply-gated to reduce leakage power. Fig. 3(c) shows the read, write and leakage power savings for a 2Kb SRAM array with peripheral read/write circuitry. With WADSG, we note that average read, write and leakage power savings are *67%*, *71%* and *70%*, respectively.

C. Challenges: Wake-up and Width Computation Overhead

Waking up the entire 32-b or 64-b FU from the gated to active state incurs significant delay and power overhead. For our 64-b adder, this delay overhead was calculated to be *25%*. Compared to the power consumed during normal 64-b operation, the wake-up power overhead is *58%*. For the multiplier, the wake-up delay and power overheads are *22%* and *45%* of the normal 32-b operation. Considering that the width computation hardware is realized using simple 2-input cascaded OR logic, for 64-b operands it will require 6 logic stages. When synthesized at 45nm node, it incurs a delay and power overhead of *272ps* and *22μW*, respectively. It is important to hide the width-computation delay and minimize the power overhead due to it.

III. Architecture-Level Modifications

A. Issue and Wake-up/Select Logic

1) FU Allocation Strategies: Challenges for the *WADSG* scheme as outlined in Section II.C can be addressed through novel width-aware FU allocation policies. These policies were implemented for the baseline processor configuration (Table I).

A. Transition-Aware FU Allocation: Due to large wake-up power overhead, an execution unit which is gated to a NW state should remain in the same state and make minimum number of transitions to FW state. The transition-aware FU allocation aims to minimize $|W_{t,i} - W_{t-1,i}|$, where $W_{t,i}$ and and $W_{t-1,i}$ are the maxm. operand widths to execution unit i in cycle t and $t-1$ respectively. In this policy, $W_{t-1,i} \forall i = 1\ to\ N$, is checked and the operand is assigned to unit i, such that i will undergo minm transition from W_i. For the proposed scheme, width information W_i for each FU is encoded using 1 or 2 bits of state elements.

B. Leakage-Aware FU Allocation: This policy targets at keeping some of the execution units in NW state for major portion of the execution time. For example, in the *Leakage Aware (3:3)* and *Leakage Aware (2:4)* policies, the last 3 and 4 integer ALUs are targeted for NW operation. Wide operand operation is allowed on these units, only if the number of wide units are not enough to support all the ready instructions in a given window.

TABLE I
Processor Configuration

Processor	8-way issue, 128 RUU, 64 LSQ, 6 integer ALUs, 2 integer mul/div units, 4 FP ALUs, 4 FP mul/div units, 2 Wr/Rd ports
Branch Prediction	Combined, 32-entry RAS, 2K 4-way BTB, 8 cycle mis-prediction penalty
Caches	64KB 2-way 2 cycle I/D L1, 2MB, 4-way 12 cycle L2
Main Memory	100 cycle latency, 32-byte wide bus

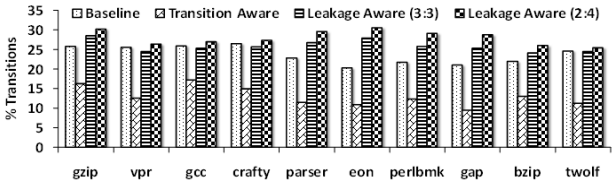
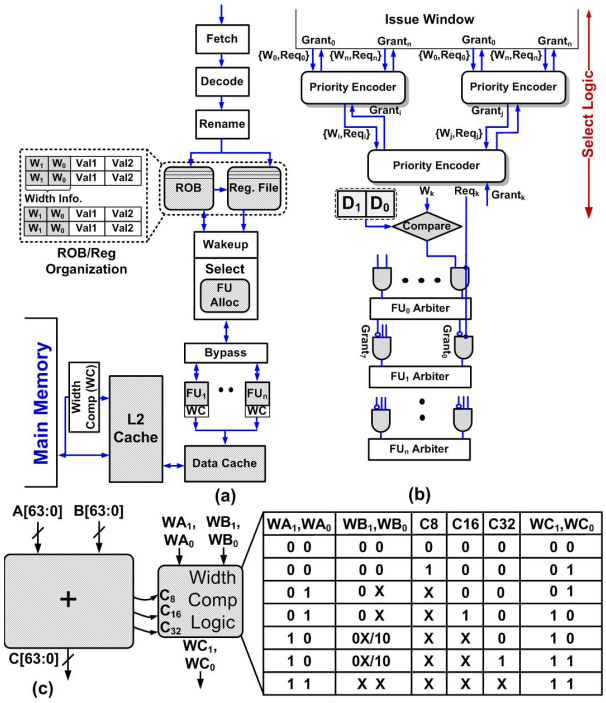


Fig. 4. Total number of transitions from NW to FW operands expressed as a percentage of the total number of operations for baseline and different FU allocation policies.

Fig. 4 shows the transitions, expressed as a percentage of the total number of operations for the baseline configuration as well as for the proposed allocation policies. For the baseline case, integer ALU and the multiplier undergoes transitions for *24%* of the operations on an average. Rate of transition is higher for leakage-aware allocation policies ($\sim$*26%* for 3:3 and $\sim$*28%* for 2:4). However, the transition-aware allocation strategy reduces the average number of transitions to *12%* of the total operations.

2) Issue Logic Modification: Fig. 5(a) shows the scope for *WADSG* scheme in a superscalar based pipeline model [5]. Fig. 5(a) shows that the *WADSG* scheme can be applied to i) the reorder buffer and the register file; ii) functional units and iii) on-chip cache (both L1 and L2). In order to realize this, the width is first calculated when an operand is written into the on-chip memory from the main memory and is encoded into 1-2 bits. The overhead for this width computation logic is reported in Section II.C. Note that this width computation can be performed in parallel to the error correction code (ECC) calculation which is performed in case of an L2 miss, and therefore does not incur any additional delay overhead. In order to propagate this width information along the pipeline, each ROB and register file entry needs to be augmented with additional bits (W_1W_0) for storing the width information for each operand. In the FU allocation policy implemented, the operand width information for the source operands coming from ROB or register file is utilized by a modified issue logic to select either a NW or FW FU. The logic for selecting the ready instructions from the issue window is implemented as a tree of priority encoders (Fig. 5(b)). In the modified FU selection logic illustrated in Fig. 5(b), D_n indicates the previous state of FU_n. For the leakage-aware FU allocation scheme, once a request (Req_k) is selected by the root of the priority encoder tree (Fig. 5(b)), the modified logic checks whether one of the source operands is FW or both are NW. In case of FW operand(s), the request is first forwarded to wide FUs (FU which is not currently gated). If no wide FUs are left for allocation, a FU which is targeted for NW operation is *waked-up* and the wide operation is assigned to this FU. For NW operations, the request is first forwarded to FUs targeted for NW operation. Fig.



WA_1,WA_0	WB_1,WB_0	C8	C16	C32	WC_1,WC_0
0 0	0 0	0	0	0	0 0
0 0	0 0	1	0	0	0 1
0 1	0 X	X	0	0	0 1
0 1	0 X	X	1	0	1 0
1 0	0X/10	X	X	0	1 0
1 0	0X/10	X	X	1	1 1
1 1	X X	X	X	X	1 1

Fig. 5. a) Scope of *WADSG* in the superscalar processor pipeline and modifications to the reorder buffer (ROB) design; b) Detailed implementation of the modified select logic; c) Width calculation logic at the output of the ALU incurs minimal design overhead.

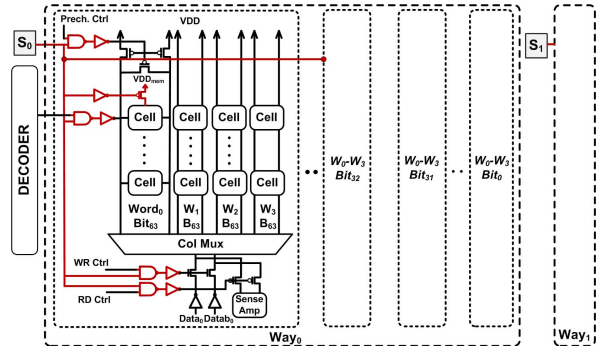


Fig. 6. Width-aware gating of cache line. Two state elements (S_1, S_0) are required to hold the width information for a 2-way set associative cache with each block of size 32B.

5(c) shows the logic for width computation logic for the adder which is based on the width of the input operands and the output sum bits. It requires only 12 gates, incurs minimal power overhead ($2.52\mu W$) and the delay overhead is completely masked by the delay for computing the most significant bits. Note that width for immediate operands are calculated in the *rename* stage and therefore does not affect the performance.

B. Width-Aware Gating in Cache

The idea of width-aware gating can be extended to save leakage power in caches at all levels. To minimize the storage $(S_0, S_1, ...S_n)$ required to store the width information, we restrict the number to only a few per cache subarray dictated by the associativity of the cache and the size of the cache block (Fig. 6). For example, for a 2-way set associative cache with block size of 32B which is organized in a 4-way bit-interleaved fashion, number of state elements per cache block holding the width information is $(\#\ of\ ways) * (Cache\ line\ size)/(Cache\ word\ size) * (Degree\ of\ interleaving) = 2 * 32 * 8/(64 * 4) = 2$. Here we assume, that the width information is encoded into 2 levels, i.e. only bits 32 to 63 are considered for gating. Assuming, the cache is divided into subarrays of size 4KB, number of state elements required for a 32KB L1 data cache (DL1) and a 2MB unified L2 (UL2) are 16 bits and 1Kb, respectively. Fig. 7(a) shows the percentage of state elements in L1 data cache and L2 unified cache which remain in NW state during simulation of 1B instructions for different SPECInt2K benchmarks. This was obtained by noting the number of state elements holding NW information at the end of each simulation cycle and updating them in case of a store or replacement. We noted that for the SPECInt2K suite, $\sim$*6%* of the L1 data cache and *77.5%* of UL2 are on an average occupied by NW operands.

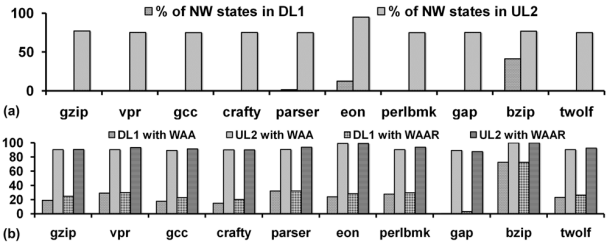


Fig. 7. a) % of NW states in DL1 and UL2 for SPECInt2K benchmarks simulated on the baseline processor; b)% improvement in NW states in DL1 and UL2 with width-aware way allocation and replacement policies.

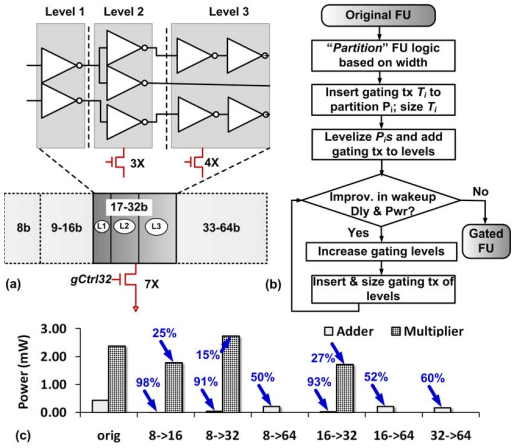


Fig. 8. a) Optimized FU design with levelized gating scheme; b) Methodology to realize a FU design with levelized gating; c) Reduction in wake-up power overhead with proposed FU design; d) Optimized FU design considering one of the two operands is NW.

To further improve the NW states, we explored a novel width-aware allocation (WAA) and replacement (WAAR) policies. For WAA policy, during write operation to a set-associative cache, if invalid cache blocks are present in multiple ways and the operand to be written is a NW operand, it is allocated to the way which is already storing NW operands. Fig. 7(b) shows that with the proposed WAA policy, percentage of NW states improves by *21%* for DL1 and by *18.5%* for UL2 over the baseline configuration with no allocation. Percentage of NW states can be further improved by modifying

the the LRU replacement policy for the baseline configuration such that cache blocks with FW operands is given higher priority during replacement. This increases the percentage of NW state elements and improves leakage saving. A *3%* increase in NW states for both DL1 and UL2 is observed with the *WAAR* compared to *WAA* approach. Penalty is however incurred due to higher DL1 and UL2 miss rates (avg. *1.4%*).

IV. Circuit-Level Optimizations

A. Minimizing Wake-up Overhead

To minimize the wake-up overhead of gated FU designs, we propose a novel FU design methodology based on the concept of *levelized gating*. The proposed design is illustrated in Fig. 8(a). The basic idea is to exploit the logic propagation delay inside a FU (adder or multiplier) to *mask* the FU wake-up delay. We observe that this can be achieved if the FU logic corresponding to the first few logic levels is left ungated while the logic for the following levels is gated. This allows the logic for the higher levels to be already waked up by the time the inputs from the first few logic levels arrive. To achieve a FU design with levelized gating, we first partition the FU logic based on output width (Fig. 8(b)). Each partition is then subdivided into levels, a gating transistor is then added to this level and is properly sized. The process of levelization is continued until no more improvement in wake-up delay and power is achieved. For width aware gating, the adder logic was partitioned into 4 portions corresponding to the output range of ≤ 8, $8-16$, $16-32$ and $32-64$. Logic for the lower 8 output bits was left ungated. The multiplier was also divided into 4 portions and similarly gated. Fig. 8(c) shows that on an average *74%* and *12%* improvement in wake-up power can be achieved for the optimized adder and multiplier designs. Our simulations show that with *levelized gating*, neither adder nor the multiplier incurs any wake-up delay penalty.

V. Simulation Results

To validate the effectiveness of the *WADSG* scheme, detailed circuit level simulations considering the effect of gating transistor were performed to estimate the impact on dynamic and leakage power. These savings were incorporated into the Wattch [8] architecture level simulator. Power savings for SPECInt2K due to *WADSG* was then accurately estimated by fast-forwarding these applications for 500M instructions and simulating them for 1B instructions on the baseline processor configuration. For estimating the total power saving in the FU, we considered ADD, MULT and logic operations. With power traces obtained during these simulations, Hotspot 5.0 [16] was used to estimate the maximum die temperature for the Alpha processor floorplan.

A. Power Saving in FUs

Fig. 9 shows the active power savings for the integer ALU and multiplier with different FU allocation policies. For the integer applications, we see that when the 64-b output range is divided into 4 bins, the best average savings (*25%*) is achieved with a leakage-aware FU allocation scheme. For the integer multiplier, leakage-aware allocation achieves better savings *45%*.

B. Power Saving in L1 and L2 Data Caches

Fig. 10(a) shows the read and write power savings achieved through width-aware way allocation for both L1 and L2 caches. As we note from Fig. 10, for 4 bins of operand width, average savings in read and write power for DL1 are *10%* and *62%* respectively. Savings for L2 are much higher (*45%* and *71%*). Fig. 10(b) shows the percentage

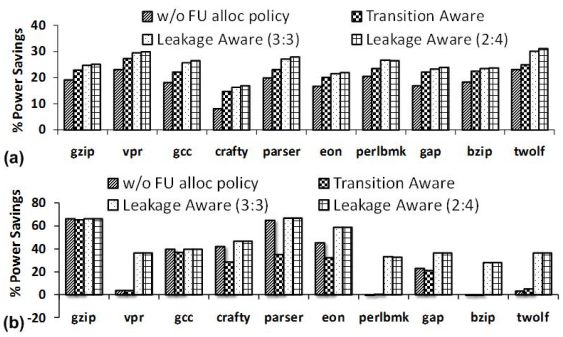


Fig. 9. % power savings in a) Integer ALU and b) integer multiplier for different issue policies, considering 2 bits to encode the operand width.

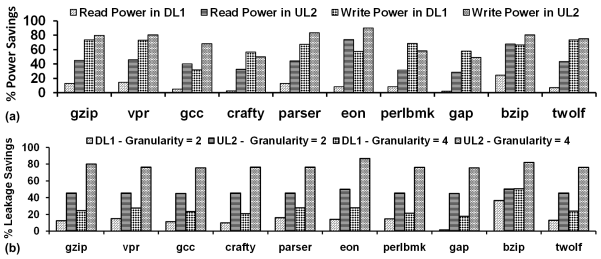


Fig. 10. a) Read and write power savings with *WAAR* in DL1 and UL2 caches; b) Leakage power savings in DL1 and UL2 caches considering 1 and 2 bits to encode operand width.

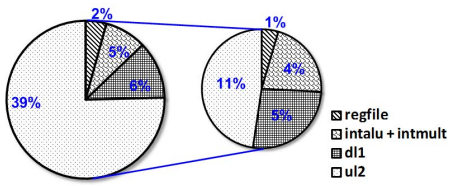


Fig. 11. Improvement in power consumption of individual processor components after application of *WADSG*.

savings for DL1 and L2 compared to the baseline configuration. Average leakage savings for DL1 and L2 considering 1-bit encoding are *15%* and *46%*, respectively. The savings improve to *27%* and *78%* for 2-bit encoding.

C. Overall Power Reduction

We have evaluated the total power savings for the processor considering the leakage contribution to the total power to be 40% for high-performance systems [10]. We assume that the *WADSG* scheme is applied to the integer ALUs, multipliers, register files [11] and the on-chip caches. With these considerations, we achieve *11.5%* saving in active power and *69.3%* saving in leakage power. Total system-level power saving considering the power overhead due to width computation on a L2 miss is calculated to be *34.5%*. Fig. 11 shows the contributions of the processor components to which *WADSG* has been applied to the total power consumption before and application of *WADSG* scheme. Note that FPUs and result buses, which are also amenable to width-aware gating is likely to improve the savings further.

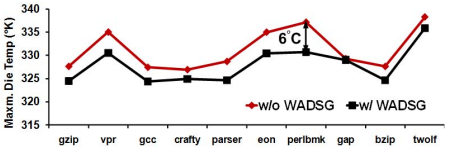


Fig. 12. Reduction in maxm. die temperature with *WADSG* scheme

D. Improvement in Thermal Profile

Improvement in active power (dynamic and leakage) translates directly to reduction in power density and hence temperature for critical hotspots, e.g. integer units. Fig. 12(a) shows that *WADSG* can be reduce the maximum die temperature by $6^{\circ}C$.

VI. Discussion

A. Addressing the Ldi/dt effect

During wake-up from a NW state to a FW state, both IntALU and IntMult draws significant current, almost 2X the normal FW operation. If all the IntALUs and the IntMults in a processor are switching from NW to FW in the same clock cycle, the di/dt would be large enough to cause a voltage drop in the power line. From our simulations, we note that such a scenario is rare and can be easily addressed by modifying the FU allocation policy to prevent all the IntALUs or Mults from switching together without incurring any significant performance overhead.

B. Extension to Register File, Reorder Buffer and FPUs

A width-aware gating to reduce dynamic power during read and write has already been investigated for register files [11]. Our investigation shows that *WADSG* can be applied to register files to achieve drastic reduction in leakage power (*21%* for a 32-entry register file using 45nm PTM HP model). The same applies to the reorder buffer. Finally, WADSG can be applied to FPU of a processor. We observed that mantissa addition and multiplication, especially for double precision FP operands, contribute heavily to the total power dissipated in FP units. As these are integer operations, they are amenable to power reduction through *WADSG*. In this case, *WADSG* benefits from the large percentage of FP operands with NW significands in SPECFP2K benchmarks compiled for 64-b Alpha (Fig. 13(a)). We note that for *31%* and *35%* of FP additions, mantissa for both the operands can be effectively expressed into 8 and 16 significant bits, respectively.

C. Effectiveness for Embedded Applications

The proposed approach can also be highly effective for embedded domains e.g. for multimedia and DSP applications. We have noted the frequency of NW operands for the Mibench benchmark suite using the large reference input data set. Fig. 13(b) shows the percentage cumulative occurrence of operand width for Mibench benchmarks compiled for 64-b Alpha. From these distributions, we note that *38%* and *56%* of the integer addition and multiplication operations are NW. These observations establish that *WADSG* employed to embedded processors would reap similar power savings as general-purpose applications.

VII. Conclusion

We have presented a fine-grained dynamic supply gating approach for datapath and memory. An architecture-circuit co-design approach is followed to maximize the power saving advantage while minimizing the impact on performance and design complexity. Simulation

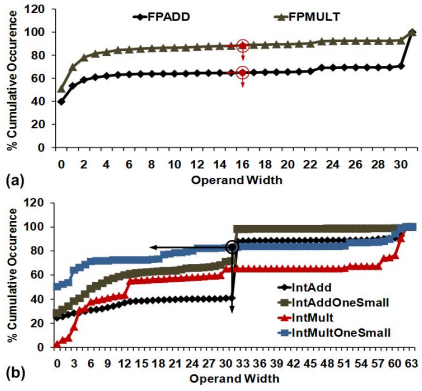


Fig. 13. % Cumulative occurrence of *Maximum Operand Width* for: a) floating point operations in SPECFP2K benchmarks; and b) integer operations for Mibench benchmarks, both compiled for 64-b Alpha.

results for general-purpose and embedded applications with varying data formats show significant power saving in processor execution units and L1/L2 caches. We have shown that the effectiveness of the approach can be enhanced using a width-aware instruction issue and memory block allocation strategy. Besides integer FUs and embedded memory, the approach can be applied to register file, reorder buffer and FPUs which experience width-dependent change in activity. As a by-product, the proposed approach can improve the thermal profile of the FUs. The proposed approach is scalable across technology nodes due to increased effectiveness of transistor stacking. Future work would include gating of sequential elements and other datapaths; application to graphics and digital signal processors; and exploring additional implications of WADSG e.g. reduction of ECC overhead.

References

[1] T. Mudge and U. Hölzle, "Challenges and Opportunities for Extremely Energy-Efficient Processors", *IEEE Micro*, 2010.
[2] J. Balfour et al., "An Energy-Efficient Processor Architecture for Embedded Systems", *IEEE Comp. Arch. Letters*, 2008.
[3] S. Bhunia et al.. "A novel synthesis approach for active leakage power reduction using dynamic supply gating", *DAC*, 2005.
[4] J.W. Tschanz et al. "Dynamic Sleep Transistor and Body Bias for Active Leakage Power Control of Microprocessors", *JSSC*, 2003.
[5] S. Palacharla et al., "Complexity-Efective Superscalar Processors", *ISCA*, 1997.
[6] Simplescalar Toolset V3.0: [Online] http://www.simplescalar.com/
[7] H. Li et al, "DCG: Deterministic Clock-Gating for Low-Power Microprocessor Design", *IEEE TVLSI*, 2004.
[8] Wattch v1.02d: [Online] http://www.eecs.harvard.edu/~dbrooks/wattch-form.html.
[9] R. Canal, A. Gonzalez and J.E. Smith, "Very Low Power Pipelines using Significance Compression", *ISCA*, 2000.
[10] J. Lee and N.S. Kim, "Optimizing Total Power of Many-Core Processors Considering Voltage Scaling Limit and Process Variations", *ISLPED*, 2009.
[11] S. Wang et al., "Exploiting Narrow-Width Values for Thermal-Aware Register File Designs", *DATE*, 2009.
[12] K. Flautner et al., "Drowsy Caches: Simple Techniques for Reducing Leakage Power", *ISCA*, 2002.
[13] Z. Hu et al., "Microarchitectural Techniques for Power Gating of Execution Units", *ISLPED*, 2004.
[14] D. Brooks and M. Martonosi, "Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance", *HPCA*, 1999.
[15] Predictive Technology Model [Online] http://ptm.asu.edu/
[16] Hotspot 5.0: Temperature Modeling Tool [Online] http://lava.cs.virginia.edu/HotSpot/