DCG: Deterministic Clock-Gating for Low-Power Microprocessor Design

Hai Li, Swarup Bhunia, Yiran Chen, Kaushik Roy, *Fellow, IEEE*, and T. N. Vijaykumar, *Member, IEEE*

***Abstract*—With the scaling of technology and the need for higher performance and more functionality, power dissipation is becoming a major bottleneck for microprocessor designs. Because clock power can be significant in high-performance processors, we propose a deterministic clock-gating (DCG) technique which effectively reduces clock power. DCG is based on the key observation that for many of the pipelined stages of a modern processor, the circuit block usage in the near future is known a few cycles ahead of time. Our experiments show an average of 19.9% reduction in processor power with virtually no performance loss for an eight-issue, out-of-order superscalar by applying DCG to execution units, pipeline latches, *D*-cache wordline decoders, and result bus drivers.**

***Index Terms*—Deterministic clock-gating (DCG), superscalar microarchitecture.**

I. Introduction

PRESENT-DAY, general-purpose microprocessor designs are faced with the daunting task of reducing power dissipation since power dissipation is quickly becoming a bottleneck for future technologies. Lowering power consumption is important for not only lengthening battery life in portable systems, but also improving reliability, and reducing heat-removal cost in high-performance systems.

Clock power is a major component of microprocessor power mainly because the clock is fed to most of the circuit blocks in the processor, and the clock switches every cycle. Table I shows the published power breakdowns for the Intel Pentium Pro and Alpha 21 264 [2]. Note that the total clock power is accounted for in different ways for the two processors. Pentium Pro reports that the global clock (i.e., the global clock distribution tree, *not* including pipeline latches and functional units) contributes 7.9% to the total processor power. In contrast, Alpha 21 264 reports that the total (i.e., global + local) clock power is 34.4% of the overall power consumption of the processor. Hence, total clock power is a substantial component of total microprocessor power dissipation.

Clock-gating is a well-known technique to reduce clock power. Because individual circuit usage varies within and across applications [1], not all the circuits are used all the time, giving rise to power reduction opportunity. By ANDing the clock with a gate-control signal, clock-gating essentially disables the clock to a circuit whenever the circuit is not used, avoiding power dissipation due to unnecessary charging and discharging of the unused circuits. Specifically, clock-gating targets the clock power consumed in pipeline latches and dynamic-CMOS-logic circuits (e.g., integer units, floating-point units, and wordline decoders of caches) used for speed and area advantages over static logic.

TABLE I
POWER BREAKDOWNS FOR INTEL PENTIUM PRO AND ALPHA 21 264

Intel Pentium Pro		Alpha 21264	
Instruction Fetch	22.2%	Caches	16.1%
Register Alias Table	6.3%	Issue Logic	19.3%
Reservation Stations	7.9%	Memory Management	8.6%
Reorder Buffer	11.1%	FP Exec. Unit	10.8%
Integer Exec. Unit	14.3%	Integer Exec. Unit	10.8%
Data Cache	11.1%	Total Clock Power	34.4%
Memory Order Buffer	6.3%		
FP Exec. Unit	7.9%		
Global Clock	7.9%		
Branch Target Buffer	4.7%		

Effective clock-gating, however, requires a methodology that determines which circuits are gated, when, and for how long. Clock-gating schemes that either result in frequent toggling of the clock-gated circuit between enabled and disabled states, or apply clock-gating to such small blocks that the clock-gating control circuitry is almost as large as the blocks themselves, incur large overhead. This overhead may result in power dissipation to be *higher* than that without clock-gating. While the concept of circuit-level clock-gating is widely known, good architectural methodologies for effective clock-gating are not. Pipeline balancing (PLB) is a recent technique, which essentially outlines a predictive clock-gating methodology [1]. PLB exploits the inherent variation of instruction level parallelism (ILP) even *within* a program. PLB uses heuristics to predict a program's ILP at the granularity of 256-cycle window. If the degree of ILP in the next window is predicted to be lower than the width of the pipeline, PLB clock-gates a cluster of pipeline components during the window.

In contrast to PLB's predictive methodology, we propose a deterministic methodology. *Deterministic clock-gating (DCG)* is based on the key observation that for many of the pipeline stages in a modern processor, a circuit block usage in a specific cycle in the near future is *deterministically* known a few cycles ahead of time. DCG exploits this advance knowledge to clock-gate the unused blocks. In particular, we propose to clock-gate execution units, pipeline latches of back-end stages after issue, L1 *D*-cache wordline decoders, and result bus drivers. In an out-of-order pipeline, whether these blocks will be used is

Manuscript received February 28, 2003; revised July 1, 2003. This work was supported in part by Defense Advanced Research Projects Agency (DARPA) PAC/C, in part by Intel Corporation, and in part by Semiconductor Research Corporation.

The authors are with the Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47906 USA (e-mail: hl@ecn.purdue.edu; bhunias@ecn.purdue.edu; yc@ecn.purdue.edu; kaushik@ecn.purdue.edu; vijay@ecn.purdue.edu).

Digital Object Identifier 10.1109/TVLSI.2004.824307

known at the *end of issue* based on the instructions issued. There is *at least one cycle* of register read stage between issue and the stages using execution units, *D*-cache wordline decoder, result bus driver, and the back-end pipeline latches. DCG exploits this one-cycle advance knowledge to clock-gate the unused blocks without impacting the clock speed (Section III).

DCG has the following key features; 1) DCG is based on actual usage of circuit blocks and not on predictions. Therefore, DCG avoids performance loss due to mispredictions causing circuits to be gated when they are needed, and lost opportunity due to mispredictions causing circuits not to be gated when they are idle. 2) DCG clock-gates at fine granularities of a few (1–2) cycles on small circuit blocks (execution units, *D*-cache decoders, result bus drivers, and pipeline latches). The fine granularity enables flexible gating of individual pipeline stages without the all-or-nothing restriction of gating the entire pipeline backend, making DCG effective. However, DCG's blocks are still substantially larger than the few gates added for clock-gating, allowing DCG to amortize the overhead. 3) DCG is a simple technique requiring no fine-tuning of thresholds, and is general enough to be applicable to clustered and nonclustered microarchitectures.

Using Wattch and a subset of the SPEC2000 suite, we show that (neglecting leakage power), DCG saves on average 20.3% of total processor power and power-delay for an 8-issue, out-of-order processor with virtually no performance impact. In contrast, PLB achieves 9.9% average power savings and 7.2% average power-delay savings, while incurring 2.9% performance loss, which are in line with [1]. If we assume leakage power accounts for 10% of dynamic power, on an average 18.3% improvement in total processor power is obtained for DCG while PLB the improvement is 8.9%.

This paper makes the following contributions.

- There is no literature on clock-gating methodology. This paper fills this gap by proposing DCG, presenting the issues, and evaluating the deterministic methodology.
- DCG not only achieves large power savings, but also incurs no performance loss, while being simple.

The rest of the paper is organized as follows. Section II discusses related work. Section III describes basic clock-gating, and identifies the out-of-order-issue pipeline stages to which we apply DCG. Section IV presents implementation details for each pipeline stage. In Section V, we describe our experimentation methodology. Section VI presents the results and Section VII concludes the paper.

II. Related Work

As mentioned before, pipeline balancing [1] is a predictive methodology to clock-gate unused components whenever the ILP is predicted to be low. Folegnani *et al.* proposed a deterministic scheme to reduce the issue queue power [6]. Manne *et al.* reduced energy without major performance penalty by restricting instruction fetch when the machine is likely to mispredict [5]. Brooks *et al.* discussed value-based clock-gating [9] and operation packing in integer arithmetic logic units (ALUs) [10]. Several papers have proposed schemes to reduce cache energy and power [11]–[15].

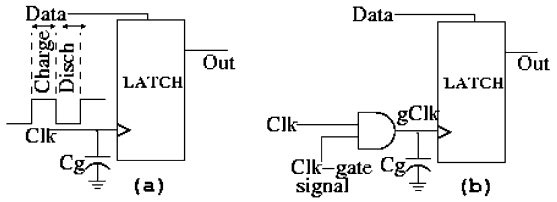

Fig. 1. Clock-gating a latch element.

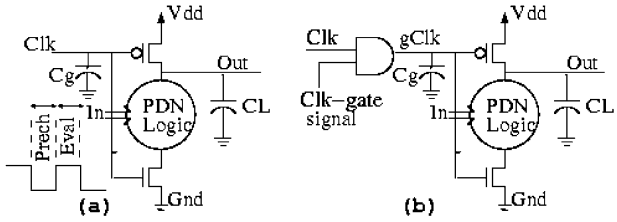

Fig. 2. Clock-gating a dynamic logic gate.

StrongArm used the conditional clocking in [18]. The Alpha 21 264 processor implemented clock–gating in floating-point functional units [3].

Popular technology approaches to reduce power include technology scaling, supply voltage change, frequency scaling, and leakage reduction technique by changing transistor threshold voltage V_{th}, etc. [16].

Circuit-level clock-gating focuses on clock-gating finite state machines (FSM) [17]. The limitation of gated-clock FSM is that its power saving heavily depends on the FSM characteristics. However, the approach is not effective for general-purpose microprocessor pipelines.

III. Deterministic Clock Gating

A. Principle of Clock-Gating

The clock network in a microprocessor feeds clock to sequential elements like flip-flops and latches, and to dynamic logic gates, which are used in high-performance execution units and array address decoders (e.g. *D*-cache wordline decoder). At a high level, gating the clock to a latch or a logic gate by ANDing the clock with a control signal prevents the unnecessary charging/discharging of the capacitances when the circuit is idle, and saves the circuit's clock power.

Fig. 1(a) shows the schematic of a latch element. C_g is the latch's cumulative gate capacitance connected to the clock. Because the clock switches every cycle, C_g charges and discharges every cycle and consumes significant amount of power. Even if the inputs do not change from one clock to the next, the latch still consumes clock power. In Fig. 1(b), the clock is gated by ANDing it with a control signal, which we refer as *Clk-gate signal*. When the latch is not required to switch state, *Clk-gate signal* is turned off and the clock is not allowed to charge/discharge C_g, saving clock power. Because the latches of an operand (32 or 64 b) can be driven by an AND gate, the capacitance of the AND gate itself is much smaller than the sum of multiple C_g of these latches. Hence, we can get a net power saving.

Now, let us consider a dynamic logic cell, the schematic of which is shown in Fig. 2(a). C_g is the effective gate capacitance that appears as a capacitive load to the clock, and C_L is the

capacitive load to the dynamic logic cell. Similar to the latch, the dynamic logic's C_g also charges and discharges every cycle and consumes power.

In addition to C_g, C_L also consumes power: at the *precharge* phase of the clock, C_L charges through the pMOS precharge transistor and during the *evaluate* phase, it discharges or retains value depending on the input to the pull-down logic (shown as "PDN" in Fig. 2). Whether C_L consumes power or not depends on *both* the current input and previous output. There are two cases: (1) If C_L holds a logic "1" at the end of a cycle, and the next cycle output evaluates to a "1," then C_L does not consume any power: Precharging an already-charged C_L does not consume power unless there are leakage losses (which we do not consider in this paper). Because the next output is a "1," there is no discharging. (2) If C_L holds a "0" at the end of a cycle, C_L consumes precharge power, *irrespective* of what the inputs are in the next cycle. Even if the input does not change, this precharge power is consumed. If the next output is a "1," no discharging occurs; otherwise, more power is consumed in discharging C_L.

Fig. 2(b) shows the same cell with gated clock. If the dynamic logic cell is not used in a cycle, *Clk-gate signal* prevents *both* C_g and C_L from switching in the cycle. While clock-gating latches reduce only unnecessary clock power due to C_g, clock-gating dynamic logic reduces unnecessary dissipation of not only the clock power due to C_g, but also the dynamic logic power due to C_L. Here also, because the AND gate's capacitance itself is much smaller than $C_g + C_L$, there is a net power saving. Moreover, a single AND gate can be used to gate the clock to a large number of dynamic logic cells.

B. Overview of DCG in a Microprocessor

In this section, we analyze the opportunity of DCG in different parts of a superscalar microarchitecture. DCG depends on two factors: 1) opportunity due to existence of idle clock cycles (i.e., cycles when a logic block is not being used) and 2) advance information about when the logic block will not be used in the future.

Fig. 3 depicts the general pipeline model for a superscalar processor [4]. The pipeline consists of eight stages with pipeline latches between successive stages, used for propagating instruction/data from one stage to the next. While we clock-gate the stages and pipeline latches marked with a "tick mark" in Fig. 3, we do not clock-gate the stages and latches with a "cross mark" due to lack of opportunity and/or advance information. Next, we explain why we do or do not clock-gate each individual pipeline latch and stage.

1) DCG for Pipeline Latches: Pipeline latches unconditionally latch their inputs at every clock edge, resulting in high power dissipation. As the technology scales down, deeper pipeline stages with more latches are used. Furthermore, the data width (e.g., 32 versus 64 b) also increases with microprocessor evolution. Consequently, the ratio of the latch power to the total processor power increases. Because most of the stage latches have some idle cycles, clock-gating the latches during these cycles can substantially save processor power. We now analyze each of the stages to determine if an idle cycle for the stage can be known in advance.

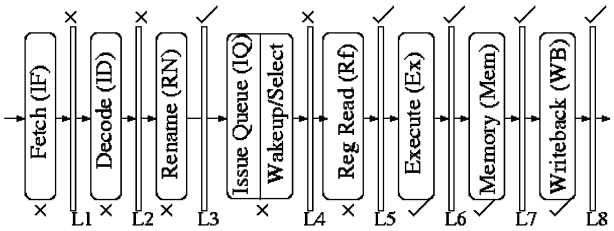

Fig. 3. Basic superscalar pipeline.

Instructions are fetched from the instruction cache every cycle. The instructions are then decoded, checked for dependences, renamed, and deposited in an instruction window. For a branch, the instructions on the predicted path will be taken before the branch is resolved. At the end of decode, we can determine how many of the instructions are in the predicted path out of those fetched. That is, if the third instruction in a fetched block is a branch and the branch is predicted to be taken then the instructions from the fourth instruction to the end of the fetched block are thrown away. Only the first three instructions enter the rename stage. Unfortunately, we cannot clock-gate the latches following fetch and decode because before decode we do not know how many instructions are in the fetched path. However, we can determine the number of instructions that will enter the rename stage at the end of decode and clock-gate the unnecessary parts of the rename latch. We have the entire rename stage to set up the clock-gate control of the rename latch. In [5], the authors propose a branch prediction confidence estimation method to reduce power dissipation due to often-mispredicted branches. In our method, however, we stick to purely deterministic means of realizing clock-gating without performance loss, and do not apply any confidence methods, which come at the cost of performance loss.

Because we can identify which and how many instructions are selected to issue only at the very end of issue, we do not have enough time to clock-gate the issue latch. We can clock-gate the latches for the rest of the pipeline stages [i.e., register read (Rf), execute (Ex), memory access (Mem) and writeback (WB)]. At the beginning of the each of the stages we know how many instructions are entering the stage, and we can exploit the time during the stage to set up the clock-gate control for these latches.

2) DCG for Pipeline Stages: Fetch stage uses the decoders in the instruction cache and decode stage uses instruction decoder, both of which are often implemented with dynamic logic circuits. However, we cannot clock-gate fetch and decode logic, because fetch and decode occur almost every cycle. We do not know which instructions are useless until we decode them, which is too late to clock-gate the decode stage. Rename stage consumes little power and so we do not consider rename stage for clock-gating.

The issue stage consists of the issue queue, which uses an associative array and a wakeup/select combinational logic. There are many papers on reducing the issue queue power. Ref. [1] clock-gates the issue queue using its predictive scheme. Ref. [6] proposes a scheme in which issue queue entries that are either deterministically determined to be empty, or deterministically known to be already woken up, are essentially clock-gated. Because [6] already presents a deterministic method to clock-gate

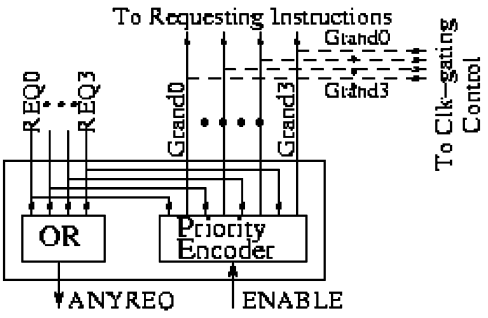

Fig. 4. Schematic of a selection logic cell with the clock-gate signals extracted from it.

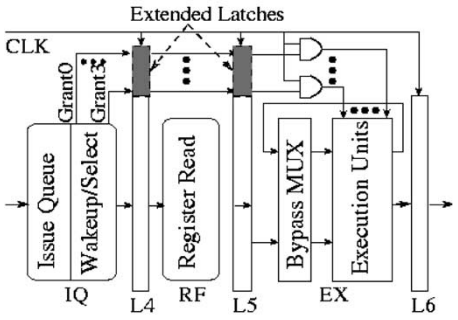

Fig. 5. Clock-gating of the execution units.

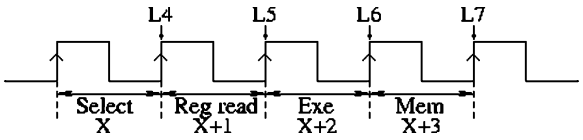

Fig. 6. Timing diagram for execution units clock-gating.

the issue queue, we do not explore applying DCG to the issue queue.

Register read stage consists of a register file implemented using an array. However, only at the very end of issue, we know how many instructions are selected and are going to access the register file in the next cycle. Hence, there is no time to clock-gate the register file.

We can clock-gate the execution units, which are often implemented with dynamic logic blocks for high performance. Based on the instructions issued, we deterministically know at the end of issue which unit is going to be used in the cycle after the register read stage. Hence, we can clock-gate the rest of the unused execution units, by setting the clock-gate control during the read cycle. Modern caches use dynamic logic for wordline decoding and the writeback stage uses result bus driver to route result data to the register file. Instructions that enter the execute stage go through the memory and writeback stages. We can use the same clock-gate control used in execute to clock-gate the relevant logic in these stages. The control signal needs to be delayed by one and two clock cycle(s), respectively, for the memory and writeback stages.

IV. Implementation of DCG

A. Execution Units

At the end of instruction issue, we know which execution units will be used in the execute stage, a few cycles into the future. The selection logic in a conventional issue queue not only selects which instructions are to be issued based on execution unit availability, but also matches instructions to execution unit. Hence, we leverage the selection logic to provide information about which execution units will remain unused and clock-gate those units.

Fig. 4 shows the schematic of selection logic associated with one type of execution units (e.g., either integer ALU, or floating-point adder, or floating-point multiplier, etc.) [4]. The request signals (REQ) come from the ready instructions once the wakeup logic determines which instructions are ready. The selection logic uses some selection policy to select a subset of the ready instructions, and generates the corresponding grant signals (GRANT). In our implementation, we send the GRANT signals to the clock-gate control.

Fig. 5 shows the pipeline details of the control. Because instructions selected in cycle X use the execution units in cycle $X + 2$ (as shown in Fig. 6), we have to pass the GRANT signals down the pipeline through latches for proper timing of clock-gating. We extend the pipeline latches for the issue and read stages by a few extra bits to hold the GRANT signals. We note that the gated clock line (output of the AND gates in Fig. 5) that feeds the execution units may be skewed a bit because of the delay through the latch and the AND gate. This skew affects only the *precharge* phase and not the *evaluate* phase. Therefore, DCG is likely not to lengthen execution unit latencies.

The control for clock-gating execution units is simple and the overhead of the extended latches and the AND gates is small compared to the execution units (e.g., 32- or 64-b carry-look-ahead adders) themselves. Therefore, the area and power overhead of the control circuitry are easily amortized by the significant power savings achieved.

If execution units keep toggling between gated and nongated modes, the control circuitry keeps switching, resulting in an increased overhead due to the power consumed by the control circuitry. To alleviate this problem, we apply *sequential priority policy* for execution units: Among the execution units of the same type, we statically assign priorities to the units, so that the higher priority units are always chosen to be used before the lower priority units. Thus, most of the time the (lower) higher priority units stay in (gated) nongated mode, minimizing the control power overhead.

B. Pipeline Latches

We clock-gate pipeline latches at the end of rename, register read, execute, memory, and writeback stages. For rename, the number of clock-gated latches in any cycle is known from the decode stage in the previous cycle. For latches in the other stages, the number of clock-gated latches in any cycle is known from the issue stage. We augment the issue stage to generate a one-hot encoding of how many instructions are issued every cycle. The encoding has a "0" to represent an empty issue slot, and a "1" to represent a full issue slot for an issued instruction, for all the issue slots of the pipeline. Much like the execution

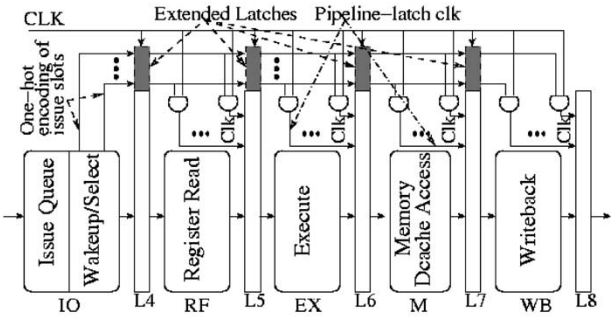

Fig. 7. Clock-gating of pipeline latches.

units, the clock the one-hot encoding is passed down the pipe via extended pipeline latches.

Fig. 7 shows the clock-gating control for the stages following issue queue. The outputs of the extended latches carrying the one-hot encoding are ANDed with the clock line to generate a set of gated clock inputs for pipeline latches corresponding to individual issue slots. Note that the clock line for the extra latches themselves is not gated.

Extensions to the pipeline latches and the extra AND gates for the control are small compared to the pipeline latches (containing issue-width $\times$ number of operands per instruction $\times$ operand width bits, e.g., $8 \times 2 \times 64 = 1024$ b) themselves, and clock drivers, respectively. Hence, the impact of the extra control logic on area and power is not significant.

C. D-Cache Wordline Decoder

D-cache wordline decoders are clock-gated using the load/store issue information; similar to the way the pipeline latches are gated. The number of load/store instructions issued in a cycle is one-hot encoded and passed down the pipeline through some extra latches added to the regular pipeline latches. A load instruction issued at cycle X uses the *D*-cache in cycle $\mathrm{X}+3$. The load/store queue does not delay the load; the load accesses the cache and the queue simultaneously. Therefore, in cycle X, the one-hot encoding deterministically identifies how many ports would be used in cycle $\mathrm{X}+3$, allowing DCG to work.

Stores, however, may be delayed in the load/store queue waiting until commit, so that the timing of store accesses to the cache may not be pre-determinable. Depending upon the load/store queue details, there are two possibilities. 1) An upcoming store access may be known in the previous cycle, giving time for the clock-gate control to be set up. 2) If no advance knowledge is available, the store may have to be delayed by one cycle to allow for clock-gate control set up. Because stores, unlike loads, do not produce values for the pipeline, this delay will result in virtually no performance loss.

If in one cycle, we find that the number of loads and stores to use the *D*-cache in the next cycle is less than the number of ports, we clock-gate the ports which are unused in the next cycle. As before, the amount of extra logic for controlling the clock is small compared to the large wordline decoders.

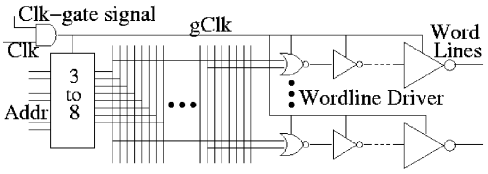

Fig. 8. Clock-gating in the *D*-cache decoder structure.

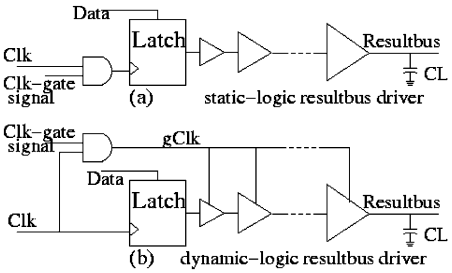

Fig. 9. Clock-gating of result bus driver.

Fig. 8 shows the schematic of a port decoder, which is implemented in 3 stages [7]. In the first stage, a set of NAND gates is used to implement 3–8 decoders. The second stage consists of a large number of NOR gates equal to the number of rows and the third stage consists of wordline drivers. The 3–8 decoders and the NOR gates are usually implemented in dynamic logic due to speed and area advantage and, hence, can be clock-gated.

D. Result Bus Driver

To route the results to the register file, writeback stage drives large capacitive load arising from the result bus. When the input of the result bus transitions back and forth between the two logic levels, power is consumed to charge/discharge the load capacitance of result bus driver.

Fig. 9 shows the schematic of clock-gating the result bus driver. Here, C_L is the load capacitance rising from the result bus. In Fig. 9(a), the result bus driver uses static logic, and clock-gating is implemented at the pipeline latch which feeds the driver. While the result bus is not used, *Clk-gate signal* isolates the input data from the result bus. Hence, C_L is not charged/discharged even if the input switches spuriously. A clock-gating schematic for result bus driver using dynamic logic is shown in Fig. 9(b). Here, clock-gating can be implemented directly to the result bus drivers. If the result bus is not used in a particular clock cycle, *Clk-gate signal* prevents C_L from switching, and reduces power. As mentioned before, the AND gate's capacitance itself is much smaller than the total gate and drain capacitance of dynamic-logic result bus drivers, and hence, there is a net power saving.

Result bus drivers in writeback stage are clock-gated by using the similar way as the pipeline latches. The number of instructions executed in a cycle is one-hot encoded and passed down the pipeline through some extra latches added to the pipeline latches. The instruction executed in cycle X goes though writeback stage at cycle $\mathrm{X}+2$. So the execution units' control signals can be used but need to be delayed by two cycles. The amount of extra logic is small compared to the large result bus driver.

TABLE II
BASELINE CCONFIGURATION OF SIMULATED PROCESSOR

Processor	8-way issue, 128 RUU, 64 LSQ, 6 integer ALUs, 2 integer mul/div units, 4 FP ALUs, 4 FP mul/div units
Branch Prediction	8K/8K/8K hybrid predictor; 32-entry RAS, 8192-entry 4-way BTB, 8 cycle misprediction penalty
Caches	64KB 2-way 2-cycle I/D L1, 2MB 8-way 12-cycle L2, both LRU
Main Memory	Infinite capacity, 100 cycle latency; Split transaction, 32-byte wide bus

V. EXPERIMENTAL METHODOLOGY

A. Architectural Simulation

We modified Wattch [2] to perform our simulations. [2] reported that Wattch is within 20% of Alpha and Pentium power dissipation in terms of absolute accuracy. However, Wattch is fairly accurate in estimating the relative contribution of various circuit blocks (similar to the breakdown shown in Table I). Because we show relative improvements we believe that our results are fairly accurate.

Wattch used dynamic logic for power estimation of execution units [2]. Cacti and Wattch used static logic for *D*-cache wordline decoder [2], [7]. We modified the code to use dynamic logic. We estimate overall processor energy using Wattch scaled for a 0.18-μm technology. The baseline processor configuration is summarized in Table II.

We use precompiled Alpha Spec2000 binaries [8] to analyze DCG's performance and power. We use ref inputs, fast-forward two billion instructions, and simulate five hundred million instructions.

B. DCG Power Calculation

For each of the execution units, pipeline latches, *D*-Cache wordline decoders and result bus drivers, the circuit power is included if the circuit is not clock-gated. If the circuit is clock-gated, zero power is added. This is the ideal case. In practice, it is impossible to totally shut off the power even the clock is gated. There is still leakage loss to be considered. Here, we assume unused units consume 10% of its dynamic power.

There is power overhead associated with DCG's control circuitry. We include the power overhead due to the extended pipeline latches when calculating latch power consumption. Apart from latches, DCG adds some extra AND gates. These AND gates can be designed with minimum size so that their power overhead is negligible, compared to large drivers for the clock tree.

C. Simulation Environment for PLB

For comparison, we also implemented pipeline balancing method (PLB), which is proposed for a clustered pipeline [1]. We adapted PLB to a nonclustered eight-wide issue out-of-order superscalar shown in Table II. In our PLB implementation, we use the *same* clock-gating granularity as [1], except our pipeline is not clustered. Accordingly, there are three possible issue widths: eight-wide issue, six-wide issue and four-wide issue. Eight-wide issue is the normal mode, while six-wide and four-wide are used for low-power mode. When the predicted instructions per cycle (IPC) is low, the machine transfers to six-wide (1/4 of issue slot is disabled) or four-wide issue (half of issue slot is disabled). To be consistent with experiments reported in [1], we used the same state machine and trigger condition for state transition as in [1].

PLB [1] reduced power for *only* the execution units and issue queue. DCG clock-gates execution units, cache, result bus and pipeline latches. If we compare DCG and PLB as they stand, the differences in their effectiveness will include not only the differences in their methodologies but also the fact that they optimize different pipeline components. To isolate the differences in their methodology, we show two versions of the PLB scheme; we show the savings for the original scheme under *PLB-orig*. We have extended PLB to clock-gate pipeline latches, *D*-cache wordline decoder in addition to the execution units and issue queue, in the scheme called *PLB-ext*. For the *D*-cache, we modified the heuristic to reduce the number of ports from 2 to 1 whenever the issue width reduces from 8 to 4. In PLB-ext, we assume that the appropriate number of pipeline latches and result buses are clock-gated whenever the issue width reduces from 8 to 6 to 4. Note that while PLB-orig and PLB-ext clock-gate the issue queue, DCG does not.

D. Optimal Number of Integer ALU Units

Usually a superscalar processor is designed with the same number of integer ALU units as its issue width (e.g., the eight-wide issue processor should have eight integer ALUs). This number is intended to achieve high performance by ensuring that if all ready instructions happen to use integer units, they need not wait. This case may be a rarity for most applications and some integer units may remain unused almost all the cycles. These unused execution units, on the other hand, dissipate similar amount of power as the used ones. Therefore, a processor with as many integer ALU units as its issue width may not be optimal for power and performance together.

Measuring the impact of clock-gating in a processor with redundant execution units may exaggerate the technique's effectiveness. To determine the optimal configuration in terms of the number of integer units for the eight-wide issue processor, we observed the effect of reducing number of integer units on processor performance starting with eight integer units. In the worse case among our benchmarks, the relative performance is 98.8% with six integer ALU units and 92.7% with four integer units. Although a configuration with four units should dissipate less power than one with six units, the former incurs significant performance loss. With respect to both power and performance six integer units seem to be optimal for eight-wide issue processor, we use this configuration in all our experiments. For the other execution units also, we choose the number of units based on power-performance consideration.

VI. RESULTS

In this section, we present power and performance results obtained from Wattch simulation. First, we present results on the effectiveness of DCG. Second, we isolate the power saving for execution units, pipeline latches, *D*-cache wordline decoders and result bus drivers and present them in Sections VI-B–VI-E. Finally, we discuss the effectiveness of DCG for future generation processors with deeper pipelines.

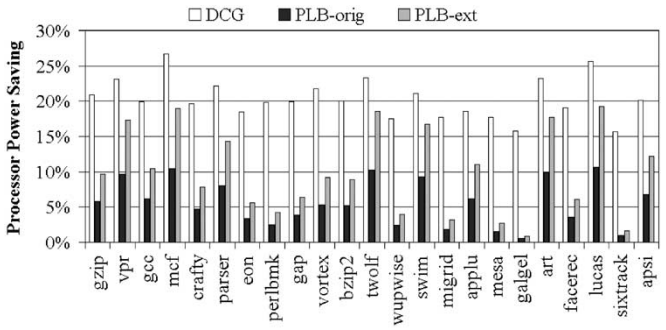

Fig. 10. Total processor power savings.

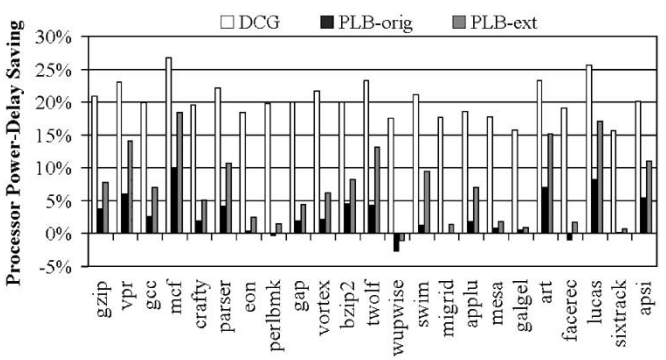

Fig. 11. Total processor power-delay savings.

A. Effectiveness of DCG

In this section, we present the power savings of DCG. Here, we consider power-delay product, which takes into account both power savings and performance loss and is a better metric to compare the effectiveness of the methods.

Recall from Section V that PLB-orig clock-gates only the execution units and issue queue, but PLB-ext clock-gates the issue queue in addition to the same pipeline components as DCG—execution units, pipeline latches, *D*-cache wordline decoders, and result bus. Not considering the issue queue advantage of PLB-ext, the difference between PLB-ext and DCG comes entirely from the nonpredictive nature and the finer granularity of DCG, and not from the choice of which pipeline components to clock-gate.

In Fig. 10, we plot the total power savings in the ideal case achieved by DCG (left bar), PLB-orig (middle bar) and PLB-ext (right bar) as a percentage of the total processor power for the base case processor, which does not implement any clock-gating. Y axis represents power savings computed as a percentage of total power.

In the ideal case, leakage power consumption on unused sections need not be considered. DCG achieves average savings of 21.3% and 19.3% for integer and floating-point (FP) programs, respectively. The corresponding savings for PLB-orig are 6.3% and 4.9%. Our PLB-orig numbers are in line with those in [1]. PLB-ext improves upon PLB-orig and saves 11.0% and 8.7% power on average. If we assume leakage power for unused sections is 10% of its dynamic power, DCG achieves average savings of 19.2% and 17.4% for integer and FP programs, respectively. The corresponding savings for PLB-ext are 9.9% and 7.8%, respectively.

DCG achieves the highest savings for *mcf* and *lucas*, because these two programs stall frequently due to unusually high cache miss rates, affording large opportunity for gating.

The difference in power savings between DCG and PLB for a particular program largely relies on the utilization of different execution units in the program. For some integer programs, such as *perlbmk*, power savings achieved by PLB-orig and PLB-ext are much smaller than that achieved by DCG. While these programs have high utilization of the integer units, they seldom use the FP units. These unused FP units can be clock-gated using DCG, but PLB does not clock-gate the units because of PLB's coarser granularity (i.e., the integer units of the corresponding "cluster" are in use, so the FP units are not disabled).

Fig. 11 shows the power-delay savings achieved for the processor by DCG, PLB-orig and PLB-ext methods computed as a percentage of the base case processor's power-delay. Y axis represents the percentage power-delay savings. The corresponding bars follow the same trends as the plot in Fig. 10 with one key difference. Because DCG incurs virtually no performance degradation, power-delay saving for DCG is the same as its power saving. Because of the impact on performance in PLB, power-delay bars for PLB-orig and PLB-ext are shorter than the corresponding power bars in Fig. 10. PLB-orig suffers 2.9% performance loss for both integer and FP programs, delivering 3.5% and 2.0% power-delay savings, respectively. PLB-ext, on the other hand, does better than PLB-orig in terms of power-delay (8.3% and 5.9%, respectively) since it saves more power by gating more components.

In the following sections, we deal with power saving in individual components. We show that DCG's savings comes from all, not any one, of the components.

B. Execution Units

In this section we discuss power saving in integer and FP units when we apply deterministic clock-gating to these units.

In initial simulations, we observed that the utilization of integer execution units for the integer benchmarks is on average 35%, while the FP units have almost no utilization for these programs. On the other hand, for the FP benchmarks, average utilizations of the FP units is about 23% while the integer units are used for about 25% of the cycles on average. DCG allows us to clock-gate an execution unit for *all* its idle cycles (Section IV). Hence, we expect to achieve about 65% power saving in the integer units for the integer benchmarks and 77% saving in the FP units for the FP benchmarks. We also expect to save almost all of the FP units' power for integer benchmarks and about 75% of integer units' power for the FP benchmarks.

PLB-ext clock-gates half of the pipeline resources when the processor works in four-wide issue mode for 50% power savings. In six-wide issue mode, we disable one integer ALU, one FPU, and one FP mul/div unit, which amounts to 25% power savings.

Fig. 12 shows the ideal power savings in integer execution units by using DCG (left bar) and PLB-ext (right bar) for all the benchmarks considered. The Y axis corresponds to power saving obtained as a percentage of total integer units' power for the base case processor. Without considering leakage power

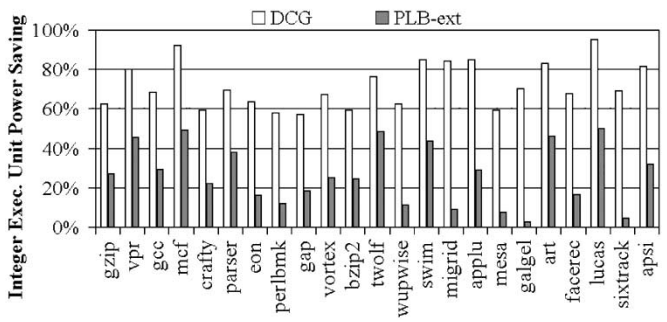

Fig. 12. Integer execution unit power savings.

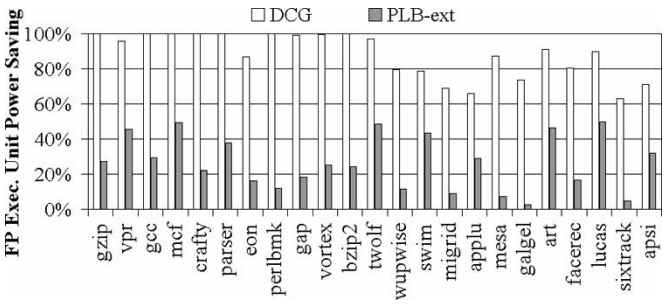

Fig. 13. FP execution unit power savings.

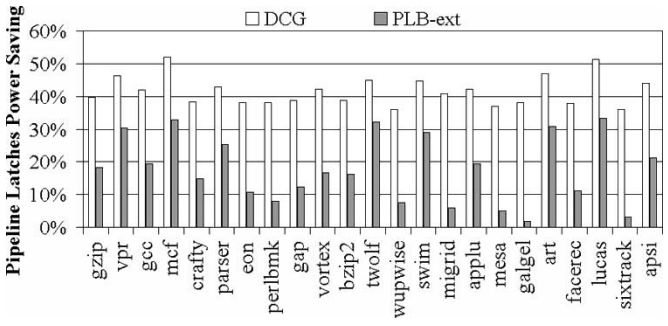

Fig. 14. Pipeline latch power savings.

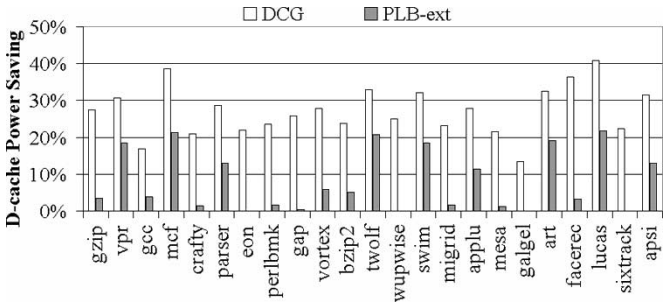

Fig. 15. *D*-cache power savings.

consumption, DCG achieves about 72.0% of the total power savings for the integer units on average. The integer execution unit power savings are 64.8% and 26.4% for DCG and PLB-ext, respectively.

Fig. 13 shows similar plot for FPUs. Y axis represents power savings in the FPUs calculated as the percentage of total power dissipated in the FPUs. DCG (left bar) achieves 77.2% total power saving for FP programs on average, and close to 100% power saving for most integer benchmarks. On an average, PLB-ext saves 26.4% and 23.0% of FPUs' power for integer and FP benchmarks, respectively.

C. Pipeline Latches

In this section, we discuss power saving for clock-gating pipeline latches with DCG. In our simulations, we have observed that the utilization of pipeline latches is about 60% on average. Because DCG allows us to clock-gate a latch for all its unused cycles (Section IV), we expect to save about 40% of latch power with this method. The extra pipeline latches required for implementing control in DCG, are not clock-gated, but account for merely 1% of total latch power. Though this overhead is small, we consider the overhead in all our experiments.

PLB-ext clock-gates 1/4 and 1/2 of the pipeline latches for each stage, when the issue width reduces from 8 to 6 and from 8 to 4 for 17% and 33% power savings, respectively. This reduction ensures that the pipeline has the right number of latches to accommodate for the low-power-modes issue widths.

Fig. 14 shows the ideal latch power savings for DCG (left bar) and PLB-ext (right bar). The Y axis corresponds to the power savings computed as a percentage of total power dissipated in pipeline latches without any clock-gating. The power saving achieved with DCG includes the power overhead due to DCG's extended latches. As expected, the power saving for pipeline latches is 41.6% on average using DCG. PLB-ext achieves 17.6% power saving in pipeline latches. *Mcf* and *lucas* stand out in terms of DCG's savings. Recall that *mcf* and *lucas* stall frequently due to high cache miss rates, affording large opportunity for clock-gating the pipeline latches.

D. D-Cache Wordline Decoder

In this section, we present power saving results in *D*-cache wordline decoder for clock-gating. In DCG, we expect to disable a wordline decoder for almost all the cycles for which the corresponding cache port is not used. Simulations about the utilization of memory ports demonstrate about 40% average usage of a memory port for the processor configuration considered. Because the wordline decoders consume about 40% of total *D*-cache power, we expect to save about 25% of cache power with DCG.

PLB-ext disables one port when the processor switches its issue width from 8 to 4, resulting in 50% of decoder power and 20% of *D*-cache power saving. To avoid undue impact on performance, we keep both the ports enabled in six-wide issue.

Fig. 15 shows the *D*-cache power saving results for DCG (left bar) and PLB-ext (right bar). The Y axis represents power savings as a percentage of total *D*-cache power for the processor with no clock-gating. DCG achieves 27.2% power saving on average, which closely matches the expected saving. After considering leakage power consumption on unused *D*-cache wordline decoder, the *D*-cache power savings are 24.5% and 7.3% for DCG and PLB-ext, respectively.

E. Result Bus Drivers

In this section, we discuss power saving in the result bus drivers. We have observed that the utilization of result bus is about 40% on average. Because we can save power in all the unused cycles, we expect to save about 60% of power consumed in the bus driver using DCG.

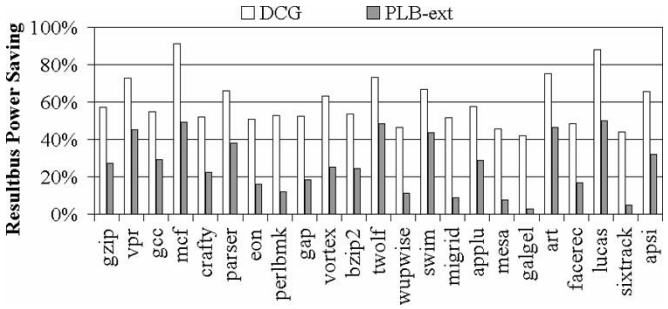

Fig. 16. Result bus power savings.

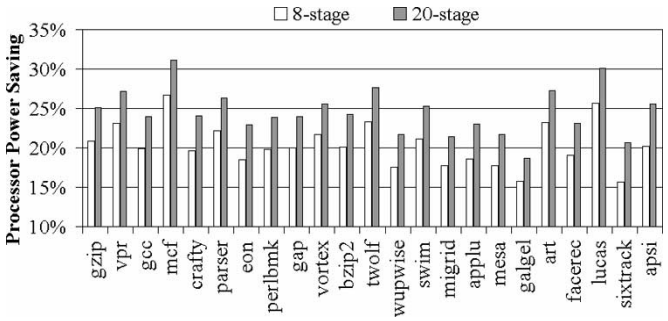

Fig. 17. Processor power saving for deeper pipelines.

For PLB-ext, we disable 2 (or 4) of the eight result buses, when the processor changes issue width from 8 to 6 (or 4) for 25% (or 50%) power savings.

Fig. 16 shows ideal power savings in result bus for DCG (left bar) and PLB-ext (right bar). The Y axis represents power savings as a percentage of total result bus power for the base case processor. DCG achieves 59.6% average power savings, which is according to the expected value. The average power saving with PLB-ext is about 32.2%. The result bus driver power savings are 53.6% and 29.0% for DCG and PLB-ext, respectively, considering total power (dynamic + leakage).

F. DCG for Deeper Pipeline

One important trend in high-performance processor design is to lengthen the processor pipelines to accommodate for higher clock rates. In this section, we discuss the impact of DCG on a deeper pipeline. Because the advance knowledge of resource usage does not change with pipeline depth (e.g., how many ALU units will be used in execute is still known after issue, and if anything there are more cycles between issue and execute), DCG should perform as well or better with deeper pipelines.

All the resources, which we can gate for the baseline architecture, can also be gated for longer pipelines, but the opportunity to gate the extra latches depends on which stages in the basic pipeline are lengthened. In particular, if a new pipeline stage is introduced for any step except fetch, decode, or issue, pipeline latches at the end of those stages can be gated using DCG, making DCG an equally or more effective technique for processors of future generations.

Fig. 17 shows the DCG power savings for a deeper pipeline without considering leakage power consumption. The Y axis corresponds to DCG power savings computed as a percentage of the total processor power for the base case processor, which does not implement any clock-gating. The left and right bars are for an 8- and 20-stage processor, respectively. On average, the 20-stage pipeline achieves 24.5% power savings and that is larger than the 8-stage processor's 20.3% savings.

VII. Conclusions

In this paper, we introduced a deterministic clock-gating (DCG) methodology based on the key observation that for many of the stages in a modern pipeline, a circuit block's usage in a specific cycle in the near future is deterministically known a few cycles ahead of time. Using this advance information, DCG clock-gates unused execution units, pipeline latches, *D*-Cache port decoders, and result bus drivers. Results show that DCG is very effective in reducing clock power in high performance microprocessors.

As high-performance microprocessor pipelines get deeper and power becomes a more critical factor, we have shown that DCG's effectiveness and simplicity will continue to be important.

References

[1] R. I. Bahar and S. Manne, "Power and energy reduction via pipeline balancing," in *Proc. 28th Int. Symp. Computer Architecture (ISCA)*, July 2001, pp. 218–229.

[2] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *Proc. 27th Int. Symp. Computer Architecture (ISCA)*, July 2000, pp. 83–94.

[3] M. Gowan, L. Biro, and D. Jackson, "Power considerations in the design of the Alpha 21 264 microprocessor," in *Proc. 35th Design Automation Conf. (DAC)*, June 1998, pp. 726–731.

[4] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Complexity-effective superscalar processors," in *Proc. 24th Annu. Int. Symp. Computer Architecture (ISCA)*, June 1997, pp. 206–218.

[5] S. Manne, A. Klauser, and D. Grunwald, "Pipeline gating: speculation control for energy reduction," in *Proc. 25th Int. Symp. Computer Architecture (ISCA)*, June 1998, pp. 132–141.

[6] D. Folegnani and A. Gonzalez, "Energy-effective issue logic," in *Proc. 28th Int. Symp. Computer Architecture (ISCA)*, July 2001, pp. 230–239.

[7] G. Rienman and N. Jouppi. Cacti 2.0: An Enhanced Access and Cycle Time Model or On-Chip Caches. [Online]. Available: http://research.compaq.com/wrl/people/jouppi/CACTI.html

[8] D. Weaver. (2000) Pre-Compiled Little-Endian Alpha ISA SPEC2000. Binaries. [Online]. Available: http://research.compaq.com/wrl/people/jouppi/CACTI.html

[9] D. Brooks and M. Martonosi, "Value-based clock gating and operation packing: dynamic strategies for improving processor power and performance," *ACM Trans. Comput. Syst.*, vol. 18, no. 2, pp. 89–126, May 2000.

[10] ——, "Dynamically exploiting narrow width operands to improve processor power and performance," in *Proc. 5th Int. Symp. High-Performance Computer Architecture (HPCA)*, Jan. 1999, pp. 13–22.

[11] D. H. Albonesi, "Selective cache ways: on-demand cache resource allocation," in *Proc. 32nd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO 32)*, Nov. 1999, pp. 248–259.

[12] C.-L. Su and A. M. Despain, "Cache design trade-offs for power and performance optimization: a case study," in *Proc. Int. Symp. Low-Power Electronics Design (ISLPED)*, 1995, pp. 63–68.

[13] N. Bellas, I. Hajj, and C. Polychronopoulos, "Using dynamic management techniques to reduce energy in high-performance processors," in *Proc. 1999 Int. Symp. Low-Power Electronics and Design (ISLPED)*, Aug. 1999, pp. 64–69.

[14] J. Kin, M. Gupta, and W. H. Mangione-Smith, "The filter cache: an energy efficient memory structure," in *Proc. 30th Ann. IEEE/ACM Int. Symp. Microarchitecture (MICRO 30)*, Dec. 1997, pp. 184–193.

[15] M. Powell, A. Agrawal, T. N. Vijaykumar, B. Falsafi, and K. Roy, "Reducing set-associative cache energy via selective direct-mapping and way prediction," in *Proc. 34th Annu. Int. Symp. Microarchitecture (MICRO)*, Dec. 2001, pp. 54–65.

[16] K. Roy and S. C. Prasad, *Low-Power CMOS VLSI Circuit Design*. New York: Wiley, 2000.
[17] J. C. Monteiro, "Power optimization using dynamic power management," in *Proc. XII Symp. Integrated Circuits Systems Design (ICSD)*, Sept. 1999, pp. 134–139.
[18] J. Montanaro *et al.*, "A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor," *IEEE J. Solid-State Circuits*, vol. 11, pp. 1703–1714, Nov. 1996.

Hai Li received the B.S. degree in electrical engineering and the M.S. degree in microelectronics from Tsinghua University, Beijing, China, in 1998 and 2000, respectively. She is currently working toward the Ph.D. degree in electrical and computer engineering at Purdue University, West Lafayette, IN.

Her research interests include low-power integrated circuits and architecture.

Mrs. Li received a Fellowship from Purdue University, West Lafayette, IN, for the year 2000–2001.

Swarup Bhunia received the undergraduate degree from Jadavpur University, Calcutta, India, and the Master's degree from the Indian Institute of Technology (IIT), Kharagpur. He is currently working toward the Ph.D. degree in the Department of Electrical Engineering, Purdue University, West Lafayette, IN.

He has worked in the EDA industry on RTL synthesis and verification since 2000. His research interest includes defect-based testing, diagnosis, noise analysis, and noise-aware design.

Yiran Chen received the B.S. and M.S. degrees in electronic engineering from Tsinghua University, Beijing, China, in 1998 and 2001, respectively. He is currently working toward the Ph.D. degree in electrical and computer engineering at Purdue University, West Lafayette, IN.

He was with Micron Advanced System Research Lab, Boise, ID, and in Minneapolis, MN, in the summers of 2002 and 2003, respectively, where he worked on optical-interconnect projects. His research interests includes power supply noise analysis, interconnect modeling, low-power circuits and architecture design.

Kaushik Roy (S'83–M'83–SM'95–F'02) received the B.Tech. degree in electronics and electrical communications engineering from the Indian Institute of Technology, Kharagpur, India, and the Ph.D. degree from the Electrical and Computer Eengineering Department, University of Illinois at Urbana-Champaign in 1990.

He was with the Semiconductor Process and Design Center of Texas Instruments, Dallas, where he worked on FPGA architecture development and low-power circuit design. He joined the electrical and computer engineering faculty at Purdue University, West Lafayette, IN, in 1993, where he is currently a Professor. His research interests include VLSI design/CAD with particular emphasis in low-power electronics for portable computing and wireless communications, VLSI testing and verification, and reconfigurable computing. He has published more than 225 papers in refereed journals and conferences, holds 6 patents, and is a coauthor of a book on *Low Power CMOS VLSI Design* (New York: Wiley, 2000).

Dr. Roy received the National Science Foundation Career Development Award in 1995, IBM faculty partnership award, ATT/Lucent Foundation Award, Best Paper Awards at the 1997 International Test Conference, IEEE 2000 International Symposium on Quality of IC Design, IEEE Latin American Test Workshop, and is currently a Purdue University Faculty Scholar Professor. He is on the Technical Advisory Board of Zenasis Inc. and a Research Visionary Board Member of Motorola Labs (2002). He has been on the Editorial Board of IEEE DESIGN AND TEST, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, and IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS. He was Guest Editor for Special Issue on Low-Power VLSI in the IEEE DESIGN AND TEST (1994) and IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS (June 2000), *IEE Proceedings—Computers and Digital Techniques* (July 2002).

T. N. Vijaykumar (S'90–M'98) received the B.E. degree (Hons.) in electrical and electronics engineering and the M.Sc. Tech. degree in computer science from the Birla Institute of Technology and Science, Pilani, India, in 1990, and the M.S. and Ph.D. degrees in computer science from the University of Wisconsin, Madison, in 1992 and 1998, respectively.

He is currently an Assistant Professor in the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN. His research interests include computer architecture, VLSI microarchitecture, low-power microprocessors, fault-tolerant architectures, speculative multithreading, and network processors.

Prof. Vijaykumar received a National Science Foundation (NSF) CAREER Award in 1999. He is a Member of the Association for Computing Machinery (ACM).