# Deterministic Clock Gating for Microprocessor Power Reduction

Hai Li, Swarup Bhunia, Yiran Chen, T. N. Vijaykumar, and Kaushik Roy
*1285 EE Building, ECE Department, Purdue University*
*<hl, bhunias, yc, vijay, kaushik>@ecn.purdue.edu*

## 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. Pipeline balancing (PLB), a previous technique, is essentially a methodology to clock-gate unused components whenever a program's instruction-level parallelism is predicted to be low. However, no non-predictive methodologies are available in the literature for efficient clock gating. This paper introduces deterministic clock gating (DCG) 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. Our experiments show an average of 19.9% reduction in processor power with virtually no performance loss for an 8-issue, out-of-order superscalar processor by applying DCG to execution units, pipeline latches, D-Cache wordline decoders, and result bus drivers. In contrast, PLB achieves 9.9% average power savings at 2.9% performance loss.*

## 1. 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. Considering all the clock signals, the total clock power is usually a substantial 30-35% of the microprocessor power [3].

*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 used for speed and area advantages over static logic.

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 block that the clock-gating control circuitry is almost as large as the block itself, incur large overhead. This overhead may result in power dissipation *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 some 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 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. 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 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.

DCG's deterministic methodology has three key advantages over PLB's predictive methodology: (1) PLB's ILP prediction is not 100% accurate; if the predicted ILP is lower than the actual ILP, PLB ends up

This research was sponsored in part by DARPA PAC/C, by Intel Corporation, and by Semiconductor Research Corporation.

clock-gating useful blocks and incurs performance loss. If the predicted ILP is higher than the actual ILP, PLB leaves unused blocks not clock-gated and incurs lost opportunity. In contrast, DCG *guarantees* no performance loss and no lost opportunity for the blocks whose usage can be known in advance. (2) PLB's clock-gating granularities, both circuit granularity and time granularity, are coarse; PLB's circuit granularity is a cluster (i.e., *many* back-end stages from register read through writeback are considered *together*). PLB's time granularity is a 256-cycle window (i.e., clusters stay clock-gated for 256-cycle windows). In contrast, DCG clock-gates at finer granularities of a few (1-2) cycles and smaller circuit blocks (execution units, D-cache address decoders, result bus drivers, and pipeline latches). Because DCG's blocks are still substantially larger than the few gates added for clock gating, DCG amortizes the overhead. PLB's coarser granularity makes it less effective and less flexible than DCG, which is a general technique applicable to non-clustered microarchitectures. (3) While PLB's prediction heuristics (FSMs and thresholds) have to be fine-tuned, DCG uses no extra heuristics and is significantly simpler.

Using Wattch [2] and a subset of the SPEC2000 suite [8], we show that DCG saves on average 19.9% 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].

This paper makes the following contributions:

- Although some commercial processors may use some form of clock gating, there is no literature on their methodology. This paper fills this gap by proposing DCG, presenting the issues, and evaluating the deterministic methodology.
- This is the first paper to show that a deterministic clock-gating methodology is better than a predictive methodology such as PLB.
- DCG not only achieves more power savings than PLB, but also incurs no performance loss compared to PLB's modest degradation, while being simpler.

The remaining sections are organized as follows. Section 2 describes basic clock gating, and identifies the out-of-order-issue pipeline stages to which we apply DCG. Section 3 presents implementation details for each pipeline stage. In section 4, we describe our experimentation methodology. Section 5 presents the results and compares DCG and PLB. Section 6 discusses related work and Section 7 concludes the paper.

## 2. Deterministic clock gating

### 2.1. Principle of clock gating

The clock network in a microprocessor feeds clock to sequential elements like 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 circuit's capacitances when the circuit is idle, and saves the circuit's clock power.

Figure 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 figure 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 AND gate's capacitance itself is much smaller than $C_g$, there is a net power saving.

A schematic of a dynamic logic cell is shown in Figure 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 *pre-charge* phase of the clock, $C_L$ charges through the PMOS pre-charge transistor and during the *evaluate* phase, it discharges or retains value depending on the input to the pull-down logic (shown as "PDN" in the figure). 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 "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

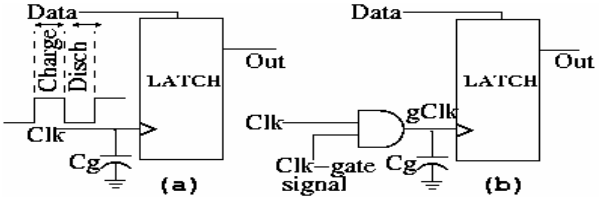

**Figure 1. Clock gating a latch element**

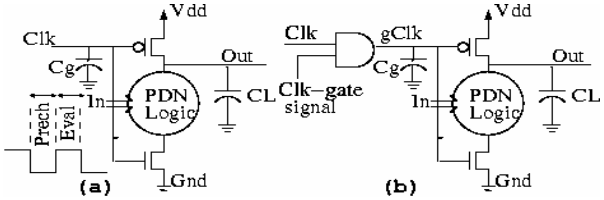

**Figure 2. Clock gating a dynamic logic gate**

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 $C_L$.

Figure 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 reduces 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.

## 2.2. Overview of DCG in a microprocessor

In this section, we analyze the opportunity of deterministic clock gating (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.

Figure 3 depicts the general pipeline model for a superscalar processor [3]. The pipeline consists of 8 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 Figure 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.

### 2.2.1. 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 vs. 64 bits) 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.

We cannot clock-gate the latches following fetch and decode because before decode we do not know which

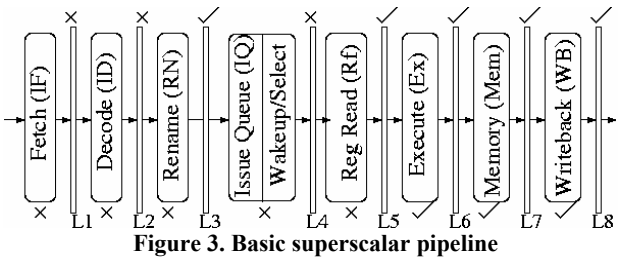


**Figure 3. Basic superscalar pipeline**

instruction is useful and which is useless. In [5], the authors propose a branch prediction confidence estimation method to reduce power dissipation due to often-mispredicted branches. 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.

At the end of decode, we can determine how many of the instructions, out of those fetched, are in the predicted path. 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. Hence, 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 setup the clock-gate control of the rename latch.

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.

### 2.2.2. 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-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. [1] clock-gates the issue queue using its predictive scheme. [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 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.

# 3. Implementation of DCG

## 3.1. 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.

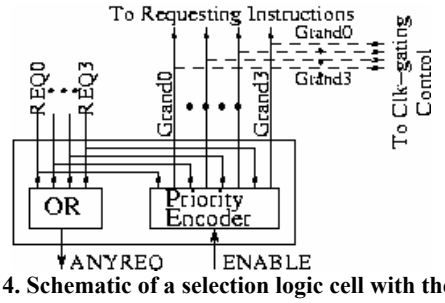


**Figure 4. Schematic of a selection logic cell with the clock gate signals extracted from it**

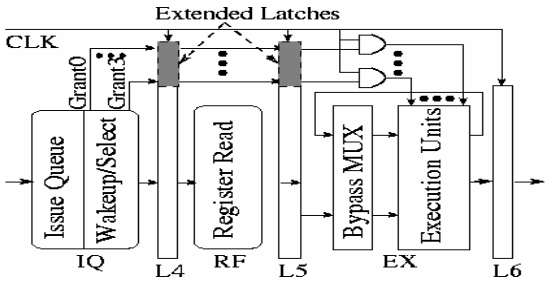


**Figure 5. Clock gating of the execution units**

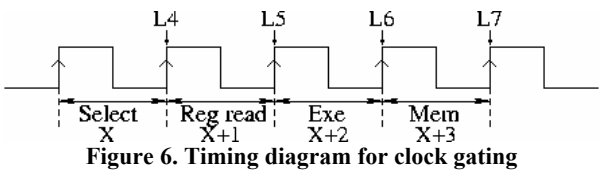


**Figure 6. Timing diagram for clock gating**

Figure 4 shows the schematic of selection logic associated with one type of execution units [6]. 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.

Figure 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 Figure 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 Figure 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-bit 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 non-gated modes, the control circuitry keeps switching, resulting in an increased overhead due to the power consumed by the control circuitry. Current charging and discharging may also cause large di/dt noise in the supply line. To alleviate these problems, 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) non-gated mode, minimizing the control power overhead. As described in [3], this *sequential priority policy* is easy to implement and does not affect overall performance.

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

Figure 7 shows the clock-gating control for the stages following issue queue. The outputs of the extended latches carrying the one-hot encoding are AND'ed 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 x number of operands per instruction x operand width bits, e.g., 8 x 2 x 64 = 1024 bits) themselves, and clock drivers, respectively. Hence, the impact of the extra control logic on area and power is not significant.

### 3.3. 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 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 X+3, allowing DCG to work.

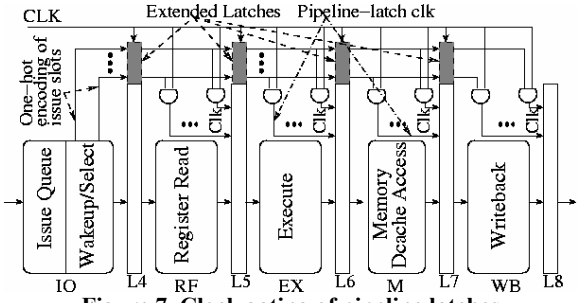


**Figure 7. Clock gating of pipeline latches**

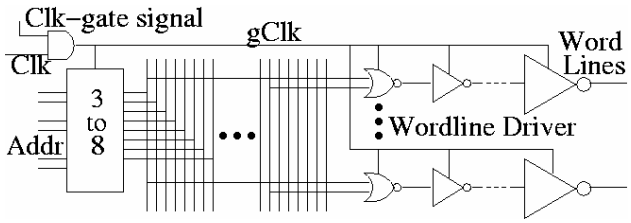


**Figure 8. Clock gating of D-cache decoder**

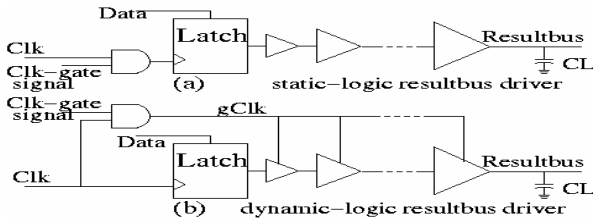


**Figure 9. Clock gating of result bus driver**

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.

Figure 8 shows the schematic of a port decoder, which is implemented in three stages [7]. In the first stage, a set of NAND gates is used to implement 3x8 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 3x8 decoders and the NOR gates are usually implemented in dynamic logic due to speed and area advantage and hence, can be clock-gated. In our experiments, we modified Wattch [2] to incorporate dynamic logic port decoder for D-cache.

### 3.4. 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.

Figure 9 shows the schematic of clock-gating the result bus driver. Here, $C_L$ is the load capacitance rising from the result bus. In Figure 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 Figure 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.

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 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.

# 4. Experimental methodology

## 4.1. Architectural simulation

We modified Wattch to perform our simulations [2]. We assume that the execution units, D-cache wordline decoders and result bus drivers use dynamic logic for performance and area considerations. We estimate overall processor energy using Wattch scaled for a 0.18μm technology. The baseline processor configuration is summarized in Table 1.

We use pre-compiled Alpha Spec2000 binaries [8] to analyze DCG's performance and power. We use ref inputs, fast-forward 2 billion instructions, and simulate 500 million instructions.

## 4.2. DCG power calculation

For each of the execution units, pipeline latches, D-Cache wordline decoders and result bus drivers, the circuit's power is added if the circuit is not clock-gated. If the circuit is clock-gated in a cycle, zero power is added. Thus, we assume that there is no leakage loss.

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.

## 4.3. 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 non-clustered 8-wide issue out-of-order superscalar shown in Table 1. In our PLB implementation, we use the *same* clock-gating granularity as the PLB paper, except our pipeline is not clustered. Accordingly, there are three possible issue widths: 8-wide issue, 6-wide issue and 4-wide issue. 8-wide issue is the normal mode, while 6-wide and 4-wide are used for low-power mode. When the predicted *IPC* (Instructions per Cycle) is low, the machine transfers to 6-wide or 4-wide issue.

**Table 1. Baseline processor configuration**

| | |
|---|---|
| **Processor** | 8-way issue, 128-entry windows, 64-entry load/store queue, 6 integer ALUs, 2 integer multiply/divide units, 4 floating point ALUs, 4 floating point multiply/divide units |
| **Branch prediction** | 2-level, 8192-entry in first level, 8192-entry in second level, 4B history; 32-entry RAS, 8192-entry 4-way BTB, 8 cycle mispredict 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 |

To be consistent with experiments reported in [1], we choose *issue IPC* ($I_{IPC}$ – Instructions issued per cycle) as the primary trigger, and *floating point issue IPC* ($FP_{IPC}$ – floating point instructions issued per cycle) and *mode history* (used to reduce spurious transitions between two issue modes) as the secondary trigger. We also choose the same state machine and threshold values for triggering as in [1]. Based on the experiments of [1], the sampling window size is 256 cycles.

In 4-wide issue mode, we disable half of issue slots, including 3 integer ALU units, 1 integer multiply/divide unit, 2 FPUs, 2 FP multiply/divide units, and 1 memory port. In 6-wide issue mode, 1 integer ALU unit, 1 FPU and 1 FP multiply/divide unit are disabled. Cache ports are left intact because memory bandwidth is important for both integer and FP performances.

Reducing the issue width from 8 to 6 and 4 results in some performance loss. PLB does not reduce the issue width below 4 because of high performance degradation that ensues.

The PLB paper reduced power for *only* the execution units and issue queue. But 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 and result bus 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.

The PLB paper based its power calculation on estimating power consumption percentage from execution units and issue queue for the Alpha 21464 and multiplying by usage coefficients to get power saving for 4-wide and 6-wide issue. To allow for direct comparison between PLB and DCG, we use Wattch to calculate power saving for PLB-orig and PLB-ext, as we do for DCG. However, our PLB numbers are in line with those in [1].

## 4.4. Optimal number of integer ALU units

Usually a superscalar processor is designed with the same number of integer ALUs as its issue. 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 ALUs 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 8-wide issue processor, we observed the effect of reducing number of integer units on processor performance starting with 8 integer units. In the worse case, the relative performance is 98.8% with 6 integer ALU units and 92.7% with 4 integer units. Although a configuration with 4 units should dissipate less power than one with 6 units, the former suffers significant performance loss. With respect to both power and performance 6 integer units seem to be optimal for 8-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.

# 5. Results

In this section, we present power and performance results obtained from Wattch simulation for both DCG and PLB methods. We compare the effectiveness of DCG with that of PLB in section 5.1. The result shows that DCG not only achieves higher power savings than PLB, but also incurs no performance loss due to its deterministic nature, while PLB incurs some performance loss. We isolate the power saving for execution units, pipeline latches, D-cache wordline decoders and result bus drivers and present them in sections 5.2 through 5.5. Finally, in section 5.6, we discuss the effectiveness of DCG for future generation processors with deeper pipelines.

## 5.1. Effectiveness of DCG

In this section, we compare power savings and power-delay saving of DCG against that of PLB.

Recall from section 4.3 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 non-predictive nature and the finer granularity of DCG, and not from the choice of which pipeline components to clock gate.

In Figure 10, we plot the total power savings 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. DCG achieves average savings of 20.9% and 18.8% for integer and floating-point (FP) programs, respectively. In contrast, corresponding savings for PLB-orig are 6.3% and 4.9%, falling considerably behind DCG. Our PLB-orig numbers are in line with those in [1]. PLB-ext improves a little upon PLB-orig and saves 11.0% and 8.7% power on average. The lower savings of PLB-ext compared to that of DCG clearly shows that DCG's deterministic methodology is superior to PLB's predictive methodology, for clock gating the same pipeline components.

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 easily 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).

Figure 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 Figure 10 with one key

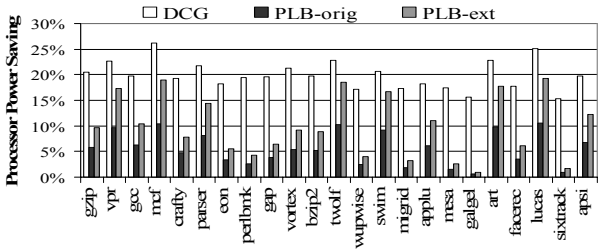


**Figure 10. Total processor power savings**

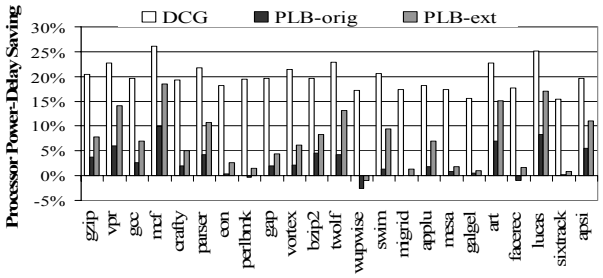


**Figure 11. Total processor power-delay savings**

difference. 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 Figure 10. Because DCG incurs virtually no performance degradation, power-delay saving for DCG is the same as its power saving. 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 a little better than PLB-orig in terms of power-delay (8.3% and 5.9% respectively) since it saves more power by gating more components. To summarize, DCG is a more effective technique than PLB, considering power and delay together.

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. We also show that for each of the components, DCG achieves more savings than PLB-ext, indicating that DCG is uniformly effective across all the pipeline components considered.

## 5.2. Execution units

In this section we discuss power saving in integer and FP units with DCG and PLB-ext.

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 3.1). Hence, we expect to achieve about 65% and 75% power saving in the integer units for the integer and FP benchmarks, respectively. We also expect to save almost all and 77% of the FPUs' power in FPUs for integer and FP benchmarks, respectively.

PLB-ext clock-gates half of the pipeline resources gated when the processor works in 4-wide issue mode. In 6-wide issue mode, we disable 1 integer ALU, 1 FPU, and 1 FP multiply/divide unit, which constitute 1/4 of the total functional units. Hence, PLB-ext can save 50% and 25% of total execution (integer and FP) unit power in 4-wide and 6-wide issue modes, respectively.

Figure 12 shows the power savings in integer execution units by using DCG (left bar) and PLB-ext (right bar). The Y-axis corresponds to power saving obtained as a percentage of total integer units' power for the base case processor. As expected, the total power saving for the integer units is about 72.0% on average. The smaller power saving in PLB-ext can be attributed to PLB's predictive nature and granularity limitation. On average, PLB saves 29.6% of integer execution unit power, which is significantly less than DCG's savings.

Figure 13 shows similar plot for FP execution units. 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. Power saving for FPUs with PLB-ext (right bar) is much less than the saving achieved with DCG. On an average, PLB-ext saves 23.0% of FPUs' power for FP benchmarks while, for some integer programs, DCG saves the entire FPU power. PLB, on the contrary, only gets less than 25% power saving for some integer benchmarks such as *eon* ad *perlbmk*. This is because the trigger condition for PLB for switching issue modes depends on both $I_{IPC}$ and $FP_{IPC}$. Although these benchmarks seldom use FPUs, processor works in 8-wide or 6-wide issue mode because of high $I_{IPC}$. Hence, the FPUs cannot be disabled and power saving in FPUs with PLB is significantly smaller for integer benchmarks than achievable with DCG.

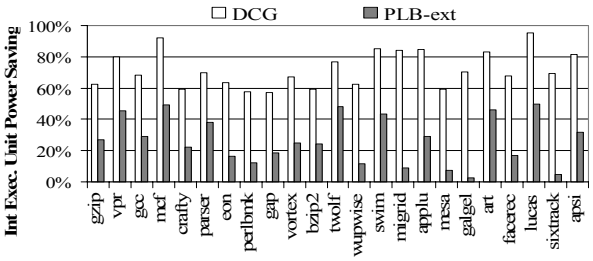


**Figure 12. Integer execution unit power savings**

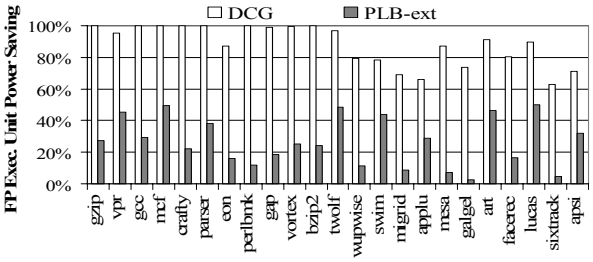


**Figure 13. FP execution unit power savings**

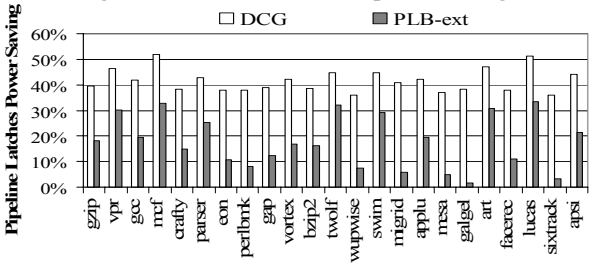


**Figure 14. Pipeline latch power savings**

## 5.3. Pipeline latches

In this section, we discuss power saving for clock gating pipeline latches with both DCG and PLB methods. 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 3.2), we expect to save about 40% of latch power. 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, respectively. This reduction ensures that the pipeline stages have the right number of latches to accommodate for the low-power-modes issue widths. Hence, PLB-ext can save 17% and 33% of pipeline latch power when the processor works in 6-wide and 4-wide issue, respectively.

Figure 14 shows the 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 is 41.6%, as expected. PLB-ext achieves 17.6% power saving in pipeline latches, which is noticeably smaller than the saving obtained using DCG.

*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.

## 5.4. D-cache wordline decoder

In this section, we present power saving results in D-cache wordline decoder. 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.

In PLB-ext, we disable one memory port when the processor switches its issue width from 8 to 4, resulting in 50% of decoder power and 20% of total cache power saving. To avoid undue impact on performance, we keep both the memory ports enabled when reducing issue width from 8 to 6. Hence, the 6-wide issue mode does not contribute to power saving in D-cache decoder.

Figure 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 22.6% power saving on average, which closely matches the expected saving. On the other hand, PLB-ext achieves merely 8.1% savings.

## 5.5. Result bus drivers

In this section, we compare power saving in the result bus drivers for DCG and PLB-ext. 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.

For PLB-ext, we disable 2 (or 4) of the 8 result buses, when the processor changes issue width from 8 to 6 (or 4). Because the number of enabled result buses is the same as the issue width, PLB-ext does not suffer from any power loss in the low power modes. Hence, we can obtain about 25% and 50% result bus power saving for the processor running with issue width 6 and 4, respectively.

Figure 16 shows 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% which is less than DCG's savings. The difference in power saving between the two methods comes from the inherent limitations of PLB-ext.

## 5.6. DCG for deeper pipelines

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, DCG should perform as well or better with deeper pipelines.

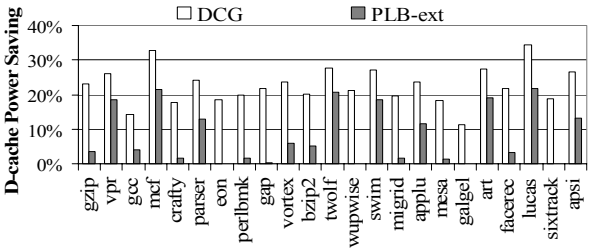


**Figure 15. D-cache power savings**

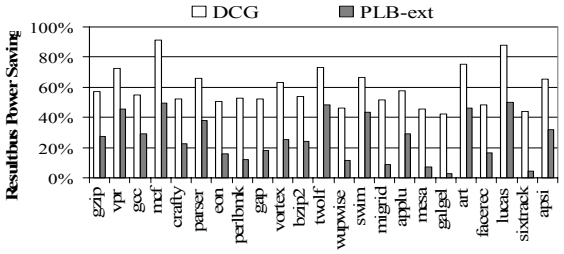


**Figure 16. Result bus power savings**

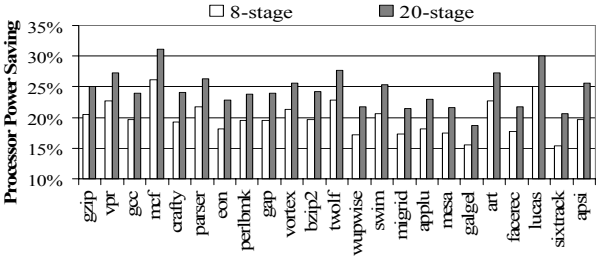


**Figure 17. Processor power saving for deeper pipelines**

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. It is worth noting that PLB method too remains valid for deeper pipelines, but DCG is expected to provide more savings than PLB.

Figure 17 shows the DCG power savings for a deeper pipeline. 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, which is larger than the 8-stage processor's 19.9% savings.

## 6. Related work

As discussed extensively throughout the paper, pipeline balancing is a predictive methodology to clock-gate unused components whenever the ILP is predicted to be low [1]. Brooks *et. al.* discuss value-based clock gating [9] and operation packing in integer ALUs [10]. Clock-gating has been used in FP functional units in commercial processors [3]. Several papers have proposed schemes to reduce cache energy and power [11, 12, 13, 14, 15].

Circuit-level clock gating focuses on clock-gating finite state machines (FSM) [16]. 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.

## 7. Conclusions

In this paper, we introduced deterministic clock gating (DCG) 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.

Pipeline balancing (PLB), a previous technique, is essentially a methodology to clock-gate unused components whenever a program's instruction-level parallelism is predicted to be low. Our experiments show an average of 19.9% reduction in processor power with virtually no performance loss for an 8-wide issue out-of-order superscalar processor by applying DCG. In contrast, PLB achieves 9.9% average power savings and 7.2% average power-delay savings, at 2.9% performance loss.

Because DCG's clock-gating granularity is finer than PLB' cluster-based granularity, DCG achieves more power savings than PLB. DCG's deterministic nature guarantees virtually no performance loss compared to PLB's modest degradation due to its predictive nature. Unlike PLB's heuristics, DCG does not require fine-tuning of any threshold values, making DCG simpler to implement.

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

## Reference


[1] R. I. Bahar, and S. Manne, "Power and energy reduction via pipeline balancing", *In Proc. of 28th Int'l Symp. on Computer Architecture (ISCA)*, Jul. 2001, pp. 218-229.

[2] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations", *In Proc. of 27th Int'l Symp. Computer Architecture (ISCA)*, Jul. 2000, pp. 83-94.

[3] M. Gowan, L. Biro, and D. Jackson, "Power considerations in the design of the Alpha 21264 microprocessor", *In proc. of 35th Design Automation Conference (DAC)*, Jun. 1998, pp. 726-731.

[4] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Complexity-effective superscalar processors", *In Proc. of 24th Annual Int'l Symp. on Computer Architecture (ISCA)*, Jun. 1997, pp. 206-218.

[5] S. Manne, A. Klauser, and D. Grunwald, "Pipeline gating: speculation control for energy reduction", *In Proc. of 25th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 1998, pp. 132-141.

[6] D. Folegnani and A. Gonzalez, "Energy-effective issue logic", *In Proc .of 28th Int'l Symp. on Computer Architecture (ISCA)*, Jul. 2001, pp. 230-239.

[7] G. Rienman and N. Jouppi, "Cacti 2.0: An enhanced access and cycle time model for on-chip caches", http://research.compaq.com/wrl/people/jouppi/CACTI.html

[8] D. Weaver, "Pre-compiled little-endian Alpha ISA SPEC2000. binaries", http://www.eecs.umich.edu/~chriswea/benchmarks/spec2000.html

[9] D. Brooks and M. Martonosi, "Value-based clock gating and operation packing: dynamic strategies for improving processor power and performance", *ACM Transactions on Computer Systems*, May 2000, 18(2), pp. 89-126.

[10] D. Brooks, and M. Martonosi, "Dynamically exploiting narrow width operands to improve processor power and performance", *In Proc. of 5th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Jan. 1999, pp. 13-22.

[11] D. H. Albonesi, "Selective cache ways: On-demand cache resource allocation", *In Proc. of 32nd Annual IEEE/ACM Int'l Symp. on 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. of 1995 In'l Symp. on Low Power Electronics and 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. of 1999 Int'l Symp. on 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. of 30th Annual IEEE/ACM Int'l Symp. on 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. of 34th Annual Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2001, pp. 54-65.

[16] J. C. Monteiro, "Power optimization using dynamic power management", *In Proc. of the XII Symp. on Integrated Circuits and Systems Design (ICSD)*, Sep. 1999, pp. 134-139.