Instruction-Driven Clock Scheduling with Glitch Mitigation

Gu-Yeon Wei, David Brooks, Ali Durlov Khan and Xiaoyao Liang
School of Engineering and Applied Sciences, Harvard University
33 Oxford St., Cambridge, MA 02138
{guyeon,dbrooks,adkhan,xliang}@eecs.harvard.edu

ABSTRACT

Instruction-driven clock scheduling is a mechanism that minimizes clock power in deeply-pipelined datapaths. Analysis of realistic processor workloads shows a preponderance of bubbles persist through pipelines like the floating point unit. Clock scheduling ostensibly adapts pipeline depth with respect to bubbles in the instruction stream without performance loss. Unfortunately, shallower pipelines (i.e. longer pipe stages) are prone to larger amounts of glitches propagating through logic, increasing dynamic power. Experimentally measured results from a 130nm FPU test chip with flexible clocking capabilities show a super-linear increase in glitch-induced dynamic power for shallower pipelines. While higher glitch power can severely diminish the power savings offered by clock scheduling, judicious clocking of intermediate stages offers glitch mitigation to recover power savings for worst-case scenarios. Detailed analysis of clock scheduling applied to a FPU in a POWER4-like processor running realistic workloads shows an average net power savings of 15% compared to an aggressively clock-gated design.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Design studies

General Terms

Design

1. INTRODUCTION

Efficient energy utilization and management is critical in modern microprocessor designs, constrained by a maximum power budget to keep cooling and power delivery costs in check. While the pursuit of ever-higher clock frequencies has been tempered in recent years with heavier reliance on parallelism for continued performance gains, clocks still consume a sizeable fraction of the overall power budget in synchronous machines with deep pipelines [1]. Moreover, simple reliance on technology scaling to reduce power and improve performance looks to offer diminishing returns in nanoscale technologies. This motivates computer architects and circuit designers to uncover inefficiencies in traditional synchronous designs and squeeze out power savings wherever possible. To continue the power management effort, this paper investigates clock scheduling, a clock-power reduction scheme for deeply-pipelined datapaths, that ostensibly changes pipeline depth with respect to cycle-level variability in the instruction stream and combats the rise in glitch-induced dynamic power in the combinational logic to maximize overall power savings.

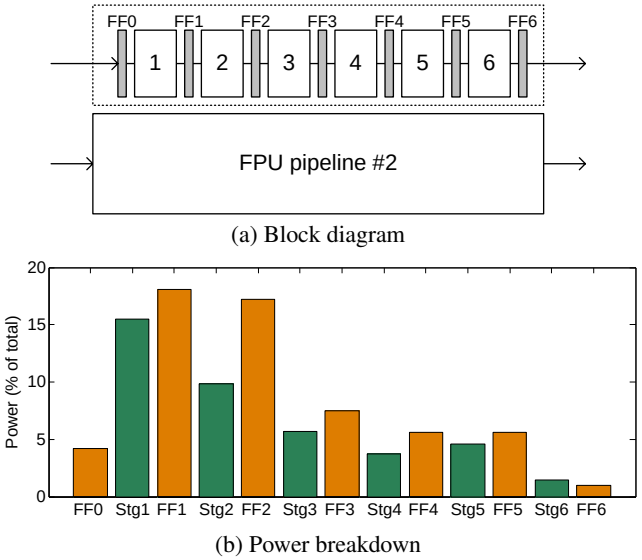
The high cost of clock power is well known and understood, which has led to implementations of clock gating at various levels of granularity—from the block down to the stage level. Traditional clock gating is based on the recognition that utilization patterns of functional units within a processor can vary widely with respect to running workloads. Hence, gating the clock at the unit level greatly reduces needless switching. Finer, stage-level clock gating can further reduce power by accommodating fluctuations in cycle-level activity patterns due to intermittent stalls resulting from memory latencies, branch mispredictions, etc. The added complexity of implementing fine-grained gating in the clock-distribution network is more than justified by the power savings offered. In recent years, researchers have proposed further extensions to stage-level clock gating by leveraging the intermittent stalls or bubbles flowing through deep datapath pipelines to avoid clocking of intermediate latch stages [2], thereby aggregating multiple shorter pipe stages into fewer longer stages, i.e. shallower pipeline depth. While such an approach offers significant clock-power savings for long sequences of bubbles between valid data flowing through a pipeline, the longer combinational logic blocks are much more susceptible to glitch (i.e. spurious signal transitions) propagation that diminish overall power savings. Hence, the benefits of such transparent pipelining schemes must be evaluated in the context of realistic workloads to assess the clock-power savings relative to increases in glitch power.

This paper introduces instruction-driven clock scheduling applied to deep pipelines like the floating point unit (FPU) in modern microprocessors to reduce clock power. For example, clocks consume $\sim$60% of the power in IBM POWER4's FPU [3]. While conceptually similar to transparent pipelining in [2], we consider clock scheduling for flip-flop (FF) based pipelines and propose the circuit-level modifications to FF clocking that are required. In addition to glitch reductions that result from constraints imposed by the modified FF clocking, we also propose a simple and yet effective glitch mitigation scheme that judiciously inserts clock edges to block glitch propagation. We rely on experimentally measured results from a 130nm 6-stage FPU test chip to show that glitch power grows with pipe stage depth, potentially compromising the power-saving promises of clock scheduling. Through detailed analysis of realistic workloads running on FPUs, we show that clock scheduling offers net power savings and glitch mitigation can improve savings.

The contributions of this work are summarized as follows:

1. Detailed analysis of realistic workloads (e.g., *SPECfp*) verifies the potential for clock-power savings in the FPU of microprocessors like IBM's POWER4.
2. We propose clock scheduling, applied to flip-flop based designs with modifications made to the clocks, and demonstrate the extent of clock-power savings possible with respect to bubbles in the instruction stream.
3. Experimentally measured results from a 130nm FPU test chip demonstrate a super-linear relationship between increases in dynamic power due to glitches and pipeline depth.
4. We evaluate clock scheduling and glitch mitigation in the context of realistic workloads and demonstrate average net power savings of 15% can be achieved compared to an aggressively clock-gated design.

ISLPED'08, August 11–13, 2008, Bangalore, India.



(a) Block diagram

(b) Power breakdown

Figure 1: FPU pipeline characteristics assumed in the case study.

The rest of the paper is organized as follows. Before describing and evaluating the potential benefits of clock scheduling, Section 2 presents a case study to show the preponderance of bubbles that persist through the FPU in a processor like IBM's POWER4 running *SPECfp* benchmarks. Driven by the high clock power seen and the apparent potential for power savings, Section 3 presents the details of clock scheduling. However, we must temper our enthusiasm by understanding the impact of glitches as described in Section 4. Luckily, evaluation of clock scheduling with glitch mitigation in Section 5 shows net power savings are still possible for FPUs. Section 6 provides an overview of related work before closing the paper with concluding thoughts in Section 7.

2. CASE STUDY: 6-STAGE FPU

Clock scheduling is proposed on the premise that typical workloads exhibit a high degree of cycle-level variations in activity due to intermittent bubbles that persist in the instruction streams. In order to verify this premise, this section presents a brief case study of the characteristics of workloads found in the *SPECfp* benchmark suite running on a FPU like that found in IBM's POWER4.

We assume a FPU that consists of two parallel pipelines with each consisting of 6 pipeline stages divided by FFs, as shown in Fig. 1(a). Data enters the pipelines through FF0. Detailed power analysis of the RTL shows that ∼60% of the total power consumed in the FPU can be attributed to the clocks with logic switching factor of ∼16% for random data. A fully-active power consumption breakdown of the FPU per FF and logic stage is shown in Fig. 1(b). The top heavy distribution of power consumption can be attributed to highly parallel structures (e.g., Wallace Tree) towards the front of the block. We will see later that such attributes are desirable for clock scheduling and glitch mitigation.

To augment the FPU power dissipation breakdowns, we collect architectural utilization to determine potential benefits from clock scheduling. We use the Turandot [4] processor simulator to model a POWER4-like processor with two parallel 6-stage FPU pipelines. We simulate 100M-instruction traces of the *SPECfp* benchmark suite. Fig. 2 presents a stacked bar graph showing the distribution of contiguous bubbles observed. The figure shows 50% of consecutive fp instructions have one or more bubbles between them. Clock scheduling can utilize these bubbles to reduce clock power.

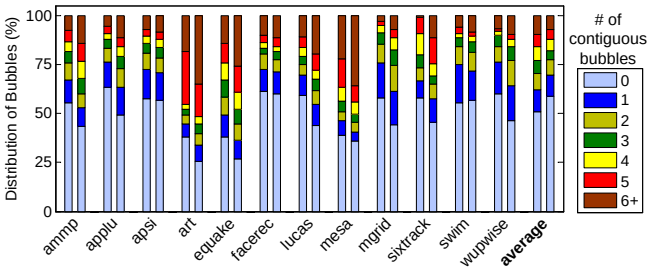


Figure 2: Distribution of contiguous bubbles found in two parallel 6-stage FPU pipelines found in a POWER4-like processor.

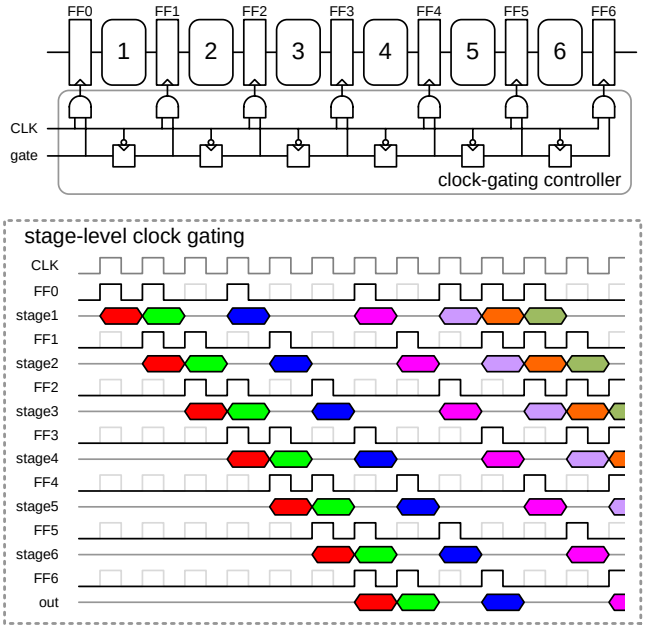


Figure 3: Block and timing diagrams of conventional clock gating.

3. CLOCK SCHEDULING

Clock scheduling extends the power-savings opportunities offered by simple stage-level clock gating by ostensibly reconfiguring the pipeline depth of a datapath. This approach is similar to the transparent pipelining scheme proposed by Jacobson [2], but we extend the approach with circuit modifications required for flip-flop based designs. Moreover, one of the major drawbacks of increasing pipeline stage delay (i.e. shallower pipelines) is increased susceptibility to glitch propagation and associated power consumption that can potentially wipe out all of the clock power savings. Consequently, clock scheduling keeps track of bubbles in the instruction stream to maximize clock power savings while judiciously inserting clock edges to block glitch propagation in order to maximize power savings. This section first provides an overview of clock scheduling applied to a 6-stage FPU pipeline datapath to highlight the potential to save clock power as a function of bubbles that separate valid data flowing through.

To establish a comparison point, Fig. 3 illustrates the basic circuitry and timing diagrams for conventional stage-level clock-gating in a 6-stage, positive edge-triggered FF-based pipeline. The clock gating control circuit is straightforward—shifting a *gate* signal along the pipeline corresponding to bubbles in the instruction stream. The FFs in the controller must trigger off of the negative clock edge in order to properly set up the gating signal prior to each clock pulse. The timing diagram shows how the clock for each FF only fires to sequence valid data through the pipeline and gated during bubbles.

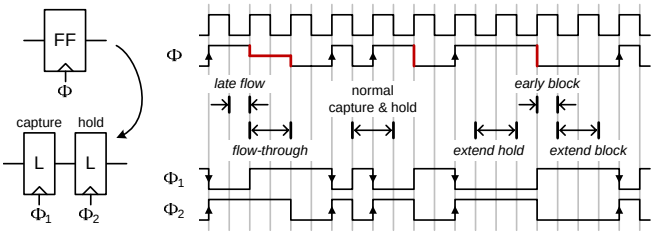


Figure 4: Modified FF clocking to enable clock scheduling.

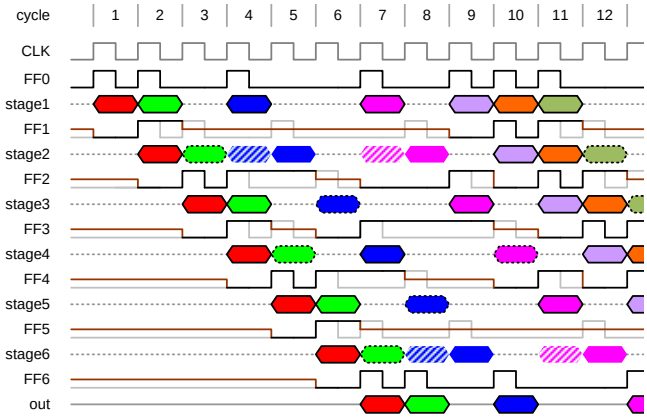


Figure 5: Timing diagrams of instruction-driven clock scheduling.

The hexagonal boxes correspond to data out of each combinational logic stage. While this stage-level clock gating can significantly reduce clock power consumption, there are additional clock power savings that can be gained by modifying the clock signals and FFs to allow data flow through.

Flip-flops typically impose hard barriers that only allow data to cross the boundary on a clock edge. Hence, in order to enable a flow-through mode, one can either add a bypass path with the aid of a MUX or modify the clock signals that drive the FF. We consider the second alternative. FFs are typically composed of two back-to-back latches clocked off of complementary clock signals. The first latch (often called the master) captures incoming data and the subsequent latch (often called the slave) holds the captured data. By breaking up the complementary clocks into two independently controlled clock phases, $\Phi 1$ and $\Phi 2$, we can enable a number of different operating modes for the original FF as shown in Fig. 4. To facilitate understanding of subsequent timing diagrams in this paper, it is important to understand these different modes. The normal capture and hold mode occurs on the rising edge of the clock. The *flow-through* mode, corresponding to a mid-level horizontal line for Φ, has both latches in transparent mode and data can flow through. To prevent premature propagation of data through the modified FF when entering and exiting *flow-through*, $\Phi 1$ and $\Phi 2$ must be carefully controlled via *late flow* and *early block* modes. The *extend hold* and *extend block* modes also prevent data race-through conditions. It is worth noting that these modes slightly modify the timing. For instance, the transition from *late flow* to *flow-through* transfers data off of $\Phi 1$ as opposed to $\Phi 2$, which can introduce an additional data-to-Q delay through the second latch.

With modified-FF clocking in place, we now investigate how it can be used to reduce clock power for data flowing through a 6-stage pipeline with interspersed bubbles. Fig. 5 illustrates a timing diagram with clock scheduling. The clock transitions required for stage-level clock gating are shown in grey for each FF. Whenever a mid-level horizontal line crosses through a grey clock pulse,

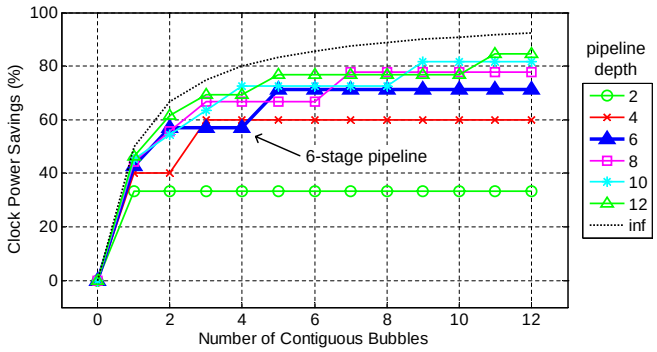


Figure 6: Clock power savings vs. number of contiguous bubbles and pipeline depth.

clock power is saved. In addition to the modified clock signals for each pipe stage, the output of each combinational logic stage (hexagonal boxes) is further annotated to illustrate different operating modes. Hexagons with solid black border edges correspond to logic stages that follow a FF operating in normal capture and hold mode. Hexagons with dotted border edges follow partially clocked FFs that transition from *late flow* to *flow-through* mode. This *late flow* mode prevents upstream data from racing through and corrupting downstream data. For example, at cycle 2, the *late flow* prevents new data entering stage 1 from clobbering data in stage 2. Hexagons with hash marks correspond to stages that suffer glitches. At cycle 4, FF1 is in *flow-through* mode and data that enters stage 1 can propagate straight through to stage 2. Consequently, the combinational logic in stage 2 can start transitioning (or glitching) and consume dynamic power (glitch power). Hexagons without border edges drawn correspond to normal logic propagation in a particular stage that follows a FF in *flow-through* mode. Notice that such hexagons are preceded by hashes and, hence, can also suffer from glitches prior to settling to a final value, denoted by the dotted horizontal lines. An example of why we implement the *early block* mode can be seen at cycle 7 of FF2. If FF2 were to transition from *flow-through* mode to normal capture and hold mode in the middle of cycle 7, data can race ahead to stage 3. While such a condition would not compromise proper functionality, it would introduce additional glitches. The *extend block* mode connects the *early block* to the next normal capture and hold mode. Lastly, an example of *extend hold* can be seen across cycles 6 and 7 of FF4. While FF4 could have entered *flow-through* mode at cycle 7, *extend hold* prevents unnecessary glitches in stage 5. A similar observation can be made for FF3 across cycles 8 and 9. These modifications made to the clock signals for a FF not only enable *flow-through* mode, but also ensure race-free data sequencing and prevent unnecessary glitch propagation. Unfortunately, we will see later that glitches can still be a problem.

The control circuitry required to implement these different modes is more complex than the simple shift register required for stage-level clock gating. The mode of each clock depends on all of the data and bubbles in flight through the datapath. Hence, the simple AND gate for the clock gating signal must be augmented with inputs corresponding to bubbles flowing through each of the pipeline stages. The proposed modifications to the clocking for the FF-based design can be translated to also work for latch-based pipelines with complementary clocks.

Before delving more deeply into the detrimental effects of glitches, Fig. 6 plots the clock power savings offered by clock scheduling, relative to stage-level clock gating, vs. the number of contiguous bubbles between valid data and pipeline depth. The plot shows that

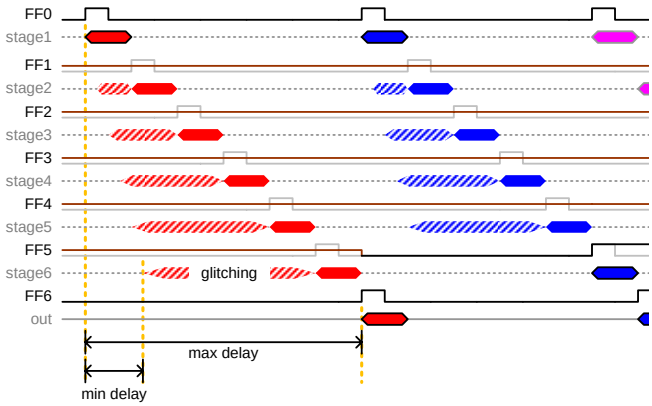


Figure 7: Timing diagram illustrating glitch propagation.

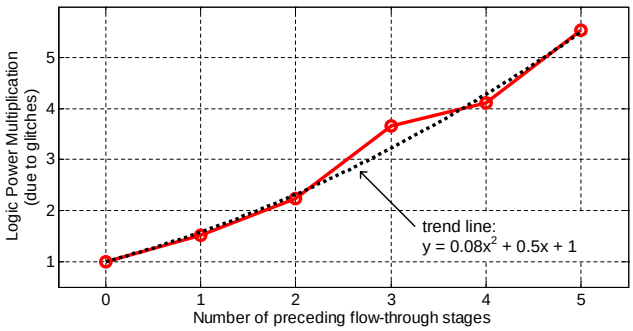


Figure 8: Measured glitch power in 130nm 6-stage FPU test chip.

clock power savings saturates once the number of contiguous bubbles exceeds the pipeline depth. Furthermore, clock power savings improves when clock scheduling is applied across more pipeline stages. While the plot may even suggest that deeper pipelining for a particular function also offers more savings, it is important to keep in mind that the total number of FFs would also grow and overall clock power may increase. Hence, it is important to consider a wide variety of factors when choosing the optimal pipeline depth as described in [5], where clock scheduling is one aspect.

4. MITIGATING GLITCHES

While clock scheduling can offer up to 70% savings in clock power for a 6-stage pipeline, glitch propagation unfortunately grows with the number of contiguous bubbles in the pipeline, possibly wiping out most if not all of the clock power savings achieved. This motivates a simple glitch mitigation strategy that can significantly reduce glitch propagation for the worst offenders.

To illustrate this glitch problem, Fig. 7 presents a timing diagram of valid data flowing through a 6-stage pipeline separated by long strings of contiguous bubbles. Assuming a long string of bubbles preceding the first valid data shown in the plot, clock scheduling dictates FF1 through FF5 ought to be in *flow-through* mode, constituting a long piece of combinational logic. As the length of the combinational logic grows, the gap between the minimum and maximum delay paths grows. In this example, stage 6 can start to glitch well before it settles out to a final value. Moreover, glitches get progressively worse towards the end of the pipeline. Unless wavepipelining can be efficiently employed, increasing glitch power must be carefully modeled and addressed.

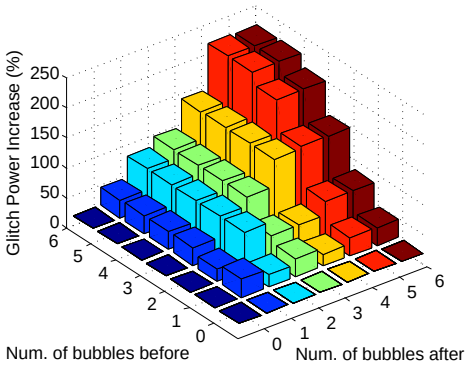
In order to better understand how glitches can lead to high power overheads, we rely on experimentally measured data from a 6-stage FPU test chip implemented in a UMC 130nm CMOS process [6]. While this FPU employs latch-based clocking with complementary clocks, the clock to each of the latches has flexible control such that the FPU can be configured to operate with different pipeline depths. Each stage of logic was designed using a standard synthesis, place and route CAD flow with optimizations focusing on performance and balancing the delay between stages. Static timing analysis for each of the six stages shows that >95% of all delay paths through the combinational logic have delays between 50% and 100% of the maximum delay path. In other words, the difference between the minimum and maximum delay paths is approximately one half of the clock period for a vast majority of the paths. By measuring the power of each stage in the FPU while changing the number of preceding flow-through stages, we can assess the impact of logic depth on glitch power. Fig. 8 plots the multiplicative increase in logic power vs. the number of preceding flow-through stages, and shows a super-linear relationship. For example, stage 6 of the pipeline suffers a >5× increase in dynamic power (for that stage) due to glitches if all six stages of the FPU operate as a single pipeline stage as opposed to being clocked separately.

It is important to note that glitches in combinational logic are intrinsically tied to the underlying logic structure and, hence, varies from one design to another. Specifically, the propagation of glitches depends on the difference between the minimum and maximum delay paths in the logic, the distribution of delay paths, divergence and convergence of paths, etc. The experimental results presented above merely show that relatively complex designs, implemented via standard CAD flows, can exhibit considerable increases in glitch-induced power as logic depth grows. With that caveat in mind, we rely on the observed power vs. logic depth trend to later evaluate the merits of clock scheduling.

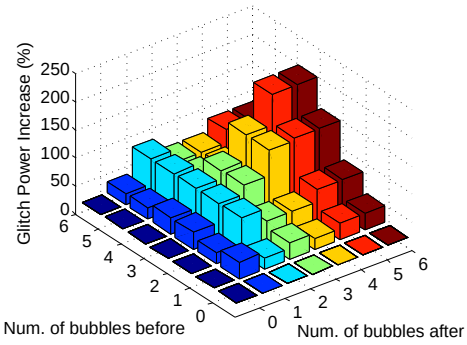
Interestingly, glitch propagation in a pipeline with clock scheduling depends on the number of bubbles that not only precede, but also follow valid data through the pipeline. By applying the trend line observed in Fig. 8, Fig. 9(a) presents a 3D plot of glitch power increase vs. the number of bubbles that precede and follow valid data. Back-to-back instructions do not suffer additional glitch power penalties (0% power increase), but also cannot benefit from clock scheduling. The worst-case condition is for isolated valid data flanked by six or more bubbles. Recognizing that the delay difference between minimum and maximum delay paths grows as the combinational logic gets longer, we propose a simple glitch mitigation scheme that judiciously clocks an intermediate FF in the pipeline for instructions preceded by five or more bubbles as shown in Fig. 10. The glitches in the last two stages (stage 5 and 6) are significantly reduced at the expense of lower clock power savings. Fig. 9(b) shows this simple glitch mitigation scheme can cut down glitch power for these worst-case conditions. While FF3 works best for the FPU in this paper, the choice of the FF for glitch mitigation depends on clock and logic power distribution across the pipeline. Later, we shall see that certain benchmarks with a preponderance of six or more bubbles greatly benefit from this glitch mitigation scheme.

5. RESULTS AND ANALYSIS

Based on our understanding of clock scheduling and the impact of glitches, we now turn back to the FPU case study in Section 2 in order to investigate the potential merits of clock scheduling. We can evaluate clock scheduling applied to the FPU in a processor like the POWER4 by combining architectural simulation results summarized by the per-benchmark bubble distribution plot in Fig. 2 with the clock-power savings trend in Fig. 6. Power savings is determined by appropriately scaling each FF's clock power, broken down for the FPU in Fig. 1(b). Furthermore, we combine the glitch



(a) Without glitch mitigation



(b) With glitch mitigation

Figure 9: Logic power increase due to glitch propagation vs. number of bubbles before and after valid data.

power vs. bubble relationship in Fig. 9 with the simulated bubble distributions to determine the resulting glitch-induced power penalties, which again assumes the glitch power vs. logic (or pipeline) depth trends measured from an experimental test chip.[1] The first and last FFs, FF0 and FF6, are assumed to always clock in and out valid data, but gated during bubbles. Since fine-grained, stage-level clock gating is commonplace, we assume its corresponding clock and logic power to be the baseline for our study. In other words, all of the results in this section demonstrate the power savings in a FPU achieved over conventional stage-level clock gating.

To first determine the clock power savings offered by clock scheduling, Fig. 11 plots the percentage of clock power savings across several different workloads from the *SPECfp* benchmark suite with and without glitch mitigation. By enabling the *flow-through* mode in the FFs, clock power savings ranges from 27% to 63%. At the high end, workloads such as *art* have ∼50% of the instructions separated by five or more contiguous bubbles, leading to large clock power savings. Unfortunately, this high clock power savings also comes with high glitch-induced power penalties as shown in the same plot, a 45% increase in logic power for *art*. Recognizing that longer logic depth leads to more glitches, simple glitch mitigation can greatly reduce the logic power increase for a modest reduction in clock power savings. The clock power savings, averaged across all benchmarks, reduces from 43% to 40% with glitch mitigation turned on. In contrast, the average logic power overhead due to glitches reduces from 17% to 11%.

By combining the cost and benefit results above, Fig. 12 plots the overall power savings for the FPU with and without glitch mitigation turned on. The plot shows overall savings as high as 36% may

[1]While the experimental test chip and the FPU in this analysis are both 6-stage pipelines, they are not the same design.

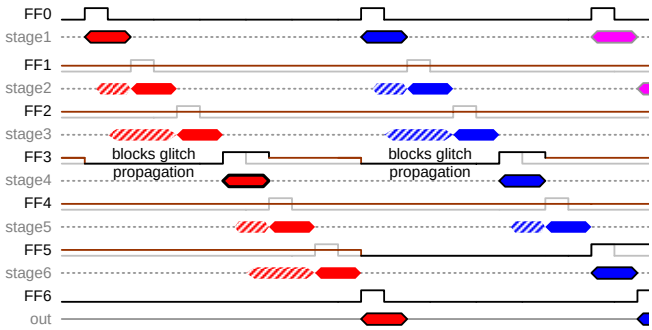


Figure 10: Timing diagram of proposed glitch-mitigation scheme.

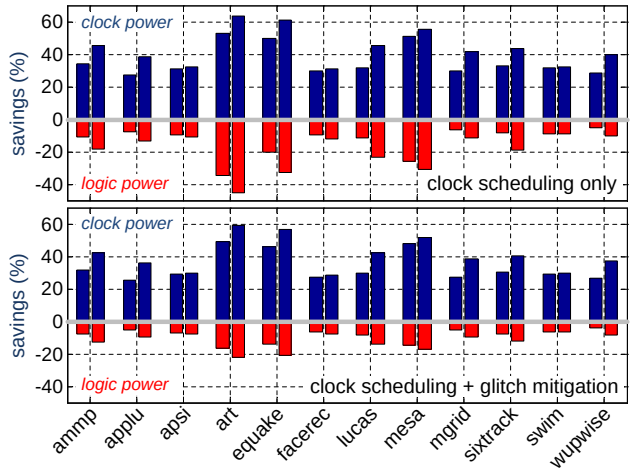


Figure 11: Evaluation of clock power savings and logic glitch power overhead penalty for two parallel FPU pipelines across *SPECfp* workloads.

be possible for one of the pipelines running *art* if glitch penalties are ignored. Unfortunately, the net savings can be considerably smaller with glitch penalties—reduced by a half for *art*. Again, glitch mitigation can recoup glitch penalties to improve the net power savings. It is important to note that glitch mitigation is most effective for benchmarks that suffer high glitch penalties such as *art*, *equake*, and *mesa*. For other benchmarks like *swim*, there is a small reduction in power savings due to a slight increase in clock power. Overall, glitch mitigation improves A more sophisticated glitch mitigation scheme may be able to yield slightly more savings.

Clock scheduling is especially effective in reducing clock power for pipelines with top-heavy power breakdowns similar to the FPU evaluated here. First of all, most of the power savings come from having at least one or two consecutive bubbles between valid data flowing through the pipeline (see Fig. 6). Moreover, with at least two consecutive bubbles, FF1 and FF2 can both operate in *flow-through* mode and they exhibit the highest clock power. Second, glitch-induced power penalties get worse towards later stages of the pipeline. Since FF0 is always clocked in our example, the first stage of logic suffers the least from glitches, but is the logic stage with the highest power. To clearly elucidate these points, Fig. 13 plots the resulting power savings if power distribution across the FFs and logic stages are equal (i.e. even breakdown of power in Fig. 1(b)). Not only are power savings due to clock power reductions reduced compared to those seen in Fig. 12, there can be a net loss in power savings for some workloads without glitch mitigation. For such datapaths where power is evenly distributed across the stages, simple glitch mitigation is not only effective, but critical.

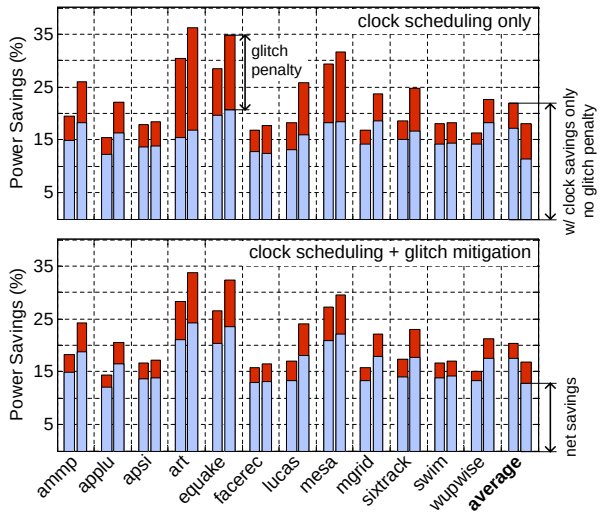


Figure 12: Evaluation of overall power savings without glitch penalties and net power savings with glitch penalties for two parallel FPU pipelines across *SPECfp* workloads.

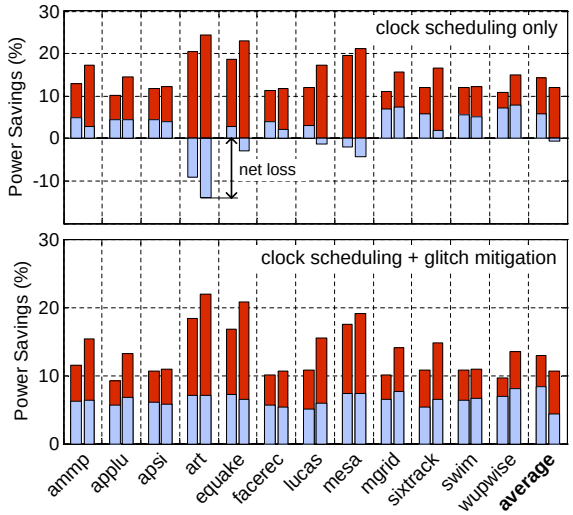


Figure 13: Evaluation of overall power savings assuming a 6-stage FPU pipeline implemented with an even distribution of power across all FFs and logic stages.

6. RELATED WORK

Clock scheduling described in this paper and transparent latching described by Jacobson in [2] are kindred techniques. They both seek to reduce unnecessary clock transitions by enabling a way to let data flow through latches or FFs unhindered when separated by bubbles. Jacobson shows that transparent pipelining offers clock power savings in range of 40-60% and data glitch power can be less than 10% of the clock power savings. This relatively-low glitch penalty is not necessarily a general result of transparent pipelining given the susceptibility to large glitch penalties depending on implementation. In contrast to transparent pipelining with latches, clock scheduling is applied to FF-based pipelines, which requires modifications to enable the *flow-through* mode. We also present a detailed study of how clock scheduling can offer clock-power savings based on an analysis of realistic workloads running on FPUs found in processors like IBM's POWER4, an understanding of FPU structure and power breakdown, and a model of data glitch power based on experimental measurements. While both clock scheduling and transparent pipeline promise significant savings in clock power, we show that data glitch power penalties can be severe and must be carefully accounted for. Lastly, we introduce and evaluate the merits of a simple glitch mitigation technique to trade off limited amounts of clock-power savings for larger reductions in glitch power.

Jacobson et al. extend the original transparent latching paper by providing a more comprehensive discussion of clock gating in [7]. A brief discussion of clock power savings for commercial workloads summarizes the clock power savings that can be achieved, but they do not present any details on the caveats related to glitch power. Hill and Lipasti build upon Jacobson's work by investigating ways to redistribute stalls at the microarchitecture level via slack prediction and maximize the benefits of transparent pipelining [8]. As a study at the microarchitecture level, this paper also ignores glitch penalties. Other work related to collapsing pipelines for power reduction are thoroughly discussed in the aforementioned papers.

7. CONCLUSION

Based on the plethora of bubbles observed to flow through the FPU for realistic *SPECfp* workloads, this paper presents the potential benefits and costs associated with clock scheduling. Clock scheduling is a technique that effectively adapts pipeline depth on the fly in response to bubbles that flow through deeply-pipelined datapaths. While we show significant clock-power savings are possible for a variety of workloads, glitch power penalties are highest for workloads that achieve the highest clock-power savings. We present a simple model of how glitch power increases with logic depth based on experimental measurements of a FPU test chip implemented in 130nm. Recognizing that glitch power gets worse for later stages in the pipeline, simple glitch mitigation schemes can be used to improve overall power savings. Lastly, our evaluations show that top-heavy blocks like a FPU can greatly benefit from clock scheduling, but pipelines with work (and power) evenly distributed across the stages may not see any power savings and designers must be especially vigilant to keep glitch penalties low.

This work is supported by National Science Foundation grants CCF-0429782 and CSR-0720566.

8. REFERENCES

[1] S. Borkar, "Thousand core chips—A technology perspective," in *Proc. DAC*, June 2007.

[2] H. M. Jacobson, "Improved clock-gating through transparent pipelining," in *Proc. ISLPED*, Aug. 2004.

[3] D. Brooks, *et al.*, "New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors," *IBM Journal of R&D*, Nov. 2003.

[4] M. Moudgill, J.-D. Wellman, and J. Moreno, "Environment for PowerPC microarchitecture exploration," in *IEEE Micro*, June 1999.

[5] D. M. Brooks, *et al.*, "Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors," in *IEEE Micro*, Nov/Dec 2000.

[6] X. Liang, D. Brooks, and G.-Y. Wei, "A process variation tolerant floating-point unit with voltage interpolation and variable latency," in *Proc. IEEE International Solid-State Circuits Conference*, Feb. 2008.

[7] H. M. Jacobson, *et al.*, "Stretching the limits of clock-gating efficiency in server-class processors," in *Proc. HPCA-11*, Dec. 2005.

[8] E. L. Hill and M. H. Lipasti, "Stall cycle redistribution in a transparent fetch pipeline," in *Proc. ISLPED*, Aug 2006.