To appear in the 31st International Symposium on Computer Architecture (ISCA 31), June 2004

Exploiting Resonant Behavior to Reduce Inductive Noise

Michael D. Powell and T. N. Vijaykumar
School of Electrical and Computer Engineering, Purdue University
{mdpowell,vijay}@purdue.edu

Abstract

Inductive noise in high-performance microprocessors is a reliability issue caused by variations in processor current (di/dt) which are converted to supply-voltage glitches by impedances in the power-supply network. Inductive noise has been addressed by using decoupling capacitors to maintain low impedance in the power supply over a wide range of frequencies. However, even well-designed power supplies exhibit (a few) peaks of high impedance at resonant frequencies caused by RLC resonant loops. Previous architectural proposals adjust current variations by controlling instruction fetch and issue, trading off performance and energy for noise reduction. However, the proposals do not consider some conceptual issues and have implementation challenges. The issues include requiring fast response, responding to variations that do not threaten the noise margins, or responding to variations only at the resonant frequency while the range of high impedance extends to a resonance band around the resonant frequency. While previous schemes reduce the magnitude of variations, our proposal, called resonance tuning*, changes the frequency of current variations away from the resonance band to a non-resonant frequency to be absorbed by the power supply. Because inductive noise is a resonance problem, resonance tuning reacts only to repeated variations in the resonance band, and not to isolated variations. Reacting after a few repetitions allows more time for the response and reduces unnecessary responses, decreasing performance and energy loss.*

1 Introduction

Inductive noise in high-performance microprocessors is a reliability issue caused by variations in processor current (di/dt) which are converted to supply-voltage glitches by impedances in the power-supply network. Lower supply voltages, higher power dissipation, and low-power techniques such as clock gating aggravate the problem by reducing absolute noise margins, increasing the chip current, and increasing the magnitude of current swings.

Circuit techniques address inductive noise by using a hierarchy of on-die, on-chip, and off-chip decoupling capacitors (d-caps) to maintain low impedance over a wide range of frequencies. However, it is difficult to cancel the impedances *between* the levels of the d-cap hierarchy. Even well-designed power supplies exhibit (a few) peaks of high impedance caused by resonant loops due to the wires, inductances, and capacitances between and in adjacent levels. Two such peaks, called *medium-frequency* and *low-frequency* peaks, are usually in the range of tens to hundreds of megahertz and a few megahertz, respectively [8,14]. Current variation at the *resonant frequencies* can cause supply-voltage glitches beyond the noise margins, which are typically around 5% of V_{dd} [10].

Recently, a few architectural techniques have been proposed to address inductive noise. Because current variations are the root cause of the problem, the techniques adjust chip current such that voltage variations fall within noise margins. The techniques adjust current by controlling instruction fetch and issue, trading off performance and energy for noise reduction. The techniques either directly measure the *magnitude* of voltage variations [10], or indirectly infer voltage variations by measuring [8] or estimating [14] the *magnitude* of current variations. Note that because inductive noise is a reliability problem, reducing average di/dt is not sufficient and absolute guarantees are needed.

Unfortunately, these magnitude-based techniques do not consider some critical conceptual issues and have substantial implementation challenges. [10] relies on sensing variations off a threshold in the supply voltage to indicate imminent noise-margin violations. **(1) False alarms:** Even if the threshold is close to the violation point, most variations do not escalate to violations because they either are caused by non-resonant current variations which are absorbed by the power supply, or are resonant echoes from past variations due to *ringing* at the resonant frequency. [10] does not distinguish between such spurious variations and true resonance, and reacts unnecessarily incurring performance and energy loss. **(2) Need for fast reaction:** To reduce false alarms, the threshold must be close to the violation point. Then, however, violations may *immediately* follow small variations off the threshold, requiring quick reaction, as acknowledged in [10]. [10] requires fine-grain sensors to classify quickly and accurately small voltage deviations as being either small enough to ignore or large enough to justify immediate reaction. As supply voltages reduce, and absolute noise margins shrink, implementing voltage sensors to classify quickly ever smaller deviations (less than 20mV) may be difficult (e.g., at scaled V_{dd}, current sensing instead of voltage sensing is being considered for SRAM sense-amplifiers [20]). In addition, wire delays in obtaining data from sensors spread across the die and in sending clock-gate signals to stall upon a threat put pressure on the sensor response times in [10].

[8] and [14] avoid *some* of these problems by using chip current (rather than voltage) as an indicator, but create other problems. **(3) Difficult current estimates and real-time calculations:** Both assume accurate a-priori estimates of chip current to infer voltage. Unfortunately, it is hard to obtain accurate current estimates. [8] uses the a-priori current estimates and

performs real-time convolution to compute future chip voltage. Unfortunately, computing convolution quickly enough to prevent noise-margin violations may be difficult to implement, as acknowledged in [8]. **(4) Resonance in a band of frequencies, not just one frequency:** [14] controls current variations at exactly the resonant frequency. [14] does not consider the fact that the range of high-impedance extends to a *resonance band* around the resonant frequency and violations can occur due to current variations *anywhere* within the band. [14]'s implementation increases processor complexity by requiring the issue queue to determine how many of each type of instruction may be issued each cycle without violating the inductive noise constraint. Extending damping to cover the entire resonance band, instead of just the resonant frequency, would add to this complexity and increase performance and energy degradation

To address these problems, we propose *resonance tuning* to control inductive noise. While previous proposals reduce the *magnitude* of current or voltage variations, we focus on changing the *frequency* of current variations. Resonance tuning is based on two key observations: First, current variations only in the resonance band are problematic; other variations are absorbed by the power distribution network. Second, inductive noise is a problem of circuit resonance from *repeated* current variations in the resonance band; variations in isolation do not build up to violations.

Our first observation has two implications. **(1) Target resonance band:** We change the frequency of current variations *away from the resonance band to non-resonant frequencies*, to be absorbed by the power supply. In contrast, [10]'s target is too wide in that [10] reacts to all voltage variations regardless of whether they occur in the resonance band. [14]'s target is too narrow in that [14] reduces current variations only at *the* resonant frequency. **(2) Sense current, not voltage:** We monitor processor current, and not voltage. Monitoring processor current is better than monitoring voltage because there is no ringing in processor core current, as we explain later. We identify current variations by directly sensing the processor current, without a priori estimates. Therefore, we avoid false alarms of [10] due to ringing and difficult current estimations of [14].

Our second observation has three implications. **(3) True resonance, not spurious variations:** We respond only to *repeated* current variations in the resonance band and not to *individual* variations as do [10] and [14]. Thus, resonance tuning alters nascent resonance before it builds up to a violation. **(4) Slow detection suffices:** We use simple counting of coarsely-identified current-variation events to detect a threat. In contrast, [8] uses full-blown convolution, and [14] needs accurate current magnitude estimates. Because typical resonance periods are over tens of cycles and will be *more* in future technologies, slow sensors suffice for us now and in the future. In contrast, [10] does not exploit resonant behavior and instead relies on fast, accurate sensors. Because nascent resonance takes a few repetitions to build up and many resonant events die before enough repetitions to trigger our response, we raise fewer false alarms than [14] or [10] and incur less performance and energy penalties. **(5) Gentle reaction suffices:** Because of the lenient timing, we do not need fast reaction either. We use a two-tiered response of minor and major perturbations in the pipeline resource usage (e.g., stalling a few issue widths and complete stall) to tune-out current variations while they are still nascent. Our response is significantly simpler than that of [14] which complicates the issue queue. Our first-tier response is gentle. In contrast, [10]'s detection requires harsh reaction (e.g, turning on and off all functional units and the d-cache), and [14] enforces strict bounds on the number and type of instructions that can issue each cycle.

Our simulations using SPEC2000 show that resonance tuning, [10], and [14] incur 5-9%, 19-46%, and 17-26% energy-delay penalty and 4-8%, 11-24%, and 15-24% performance degradation, respectively, over ranges of representative configurations.

In the next section, we discuss resonance in microprocessor packaging. In Section 3 we explain resonance tuning. Section 4 discusses our experimental methodology, and Section 5 contains our results. We discuss related work in Section 6 and conclude in Section 7.

2 Resonance in Microprocessor Packaging

Because our technique attempts to change the frequency of current variations away from the resonance band, we need to know these parameters: (1) what the resonant frequency is, (2) what the resonance band is — i.e., how much does the frequency of the current variations need to be changed so they become non-resonant, (3) when the change should occur, and (4) what should trigger the change. Based on design-time information about the resonant characteristics of the package, we determine these parameters. We primarily discuss medium-frequency resonance and mention low-frequency resonance at the end.

2.1 Medium-frequency Resonance Characteristics

For the purposes of evaluating inductive noise, the microprocessor power-distribution network may be modeled as a second-order resistive, inductive, and capacitive (RLC) circuit with the power supply modeled as a voltage source. We show such a circuit in Figure 1(a). The circuit models the power-supply impedance (R), the inductance of the connections between the die and the chip (L), and the on-die d-caps (C). The CPU circuitry, which consumes current based on the activity in the processor, is modeled as a current source. Components that are ignored in our model, such as the on-chip and off-chip d-caps, generally do not respond to variations at the frequencies of interest. More complex models of the power-distribution network are shown in [6, 8, 9]; however, the second-order model effectively captures resonant behavior and is widely used [10, 9]. We discuss microprocessor resonance in the context of second-order circuit behavior in the resonant loop among the power supply impedance, inductance between the chip and the die, and on-die capacitors.

The circuit characteristics are determined by the values of R, L, and C. For a given microprocessor, these values can be calculated from technology parameters and CAD tools [6]. For

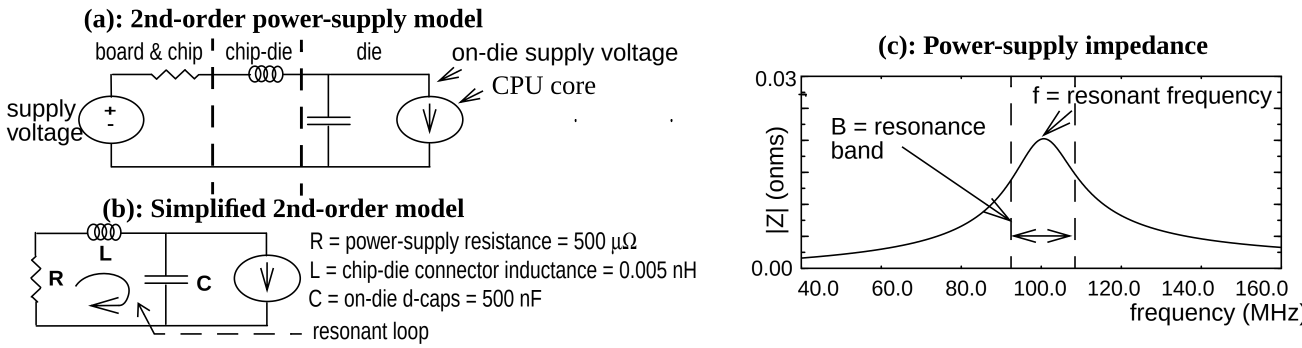

FIGURE 1: Power supply model and impedance for typical values.

our discussion, it is also useful to establish how the parameters scale with technology. Scaling leads to smaller values of R as power-distribution networks endeavor to deliver more current with smaller voltage drop. L stays about the same as it is a characteristic of the connections (usually solder bumps in today's flip-chip packages [16]) between the die and the package while the need for C increases due to the processor having more devices and current [13,15,16].

2.1.1 What is the resonant frequency?

First, we must establish that resonant oscillation is actually a concern. The circuit in the figure is said to be *underdamped* if

$$R^2 < \frac{4L}{C}$$

An underdamped circuit is subject to resonant oscillation. Because technology scaling calls for small R and large C, microprocessor power supplies are and will continue to be underdamped. (Critically damped and overdamped circuits do not satisfy the inequality and do not oscillate.)

Second, we note that the resonant frequency of a second-order circuit, at which current variations cause maximum voltage variations and the circuit stores maximum energy, is defined as:

$$f = \frac{1}{2\pi\sqrt{LC}}$$

A typical microprocessor package today may have an on-die d-cap C value on the order of 500 nF (e.g., 320 nF for the Alpha 21264 [13, 7, 16] and 700 nF for the Alpha 21364 [18]) and a parasitic inductance for all the power-distribution solder bumps in parallel on the order of 0.005 nH, giving a resonant frequency around 100 MHz [16].

2.1.2 How much should the frequency of variations be changed?

The effect of resonance is limited not only to the specific resonant frequency. Second-order circuits have a quality factor (Q) that depends on the values of L and R.

$$Q = \frac{2\pi f L}{R} = \frac{2\pi f}{2\pi B}$$

Q determines the size of the *resonance band (B),* or the width of the range of frequencies at which the circuit resonates with more than half the energy than that at the resonant frequency itself. The resonance band is shown in Figure 1(c). If our example circuit has an inductance of 0.005 nH and a resistance of 500 μΩ (because this value is the *series* resistance of the microprocessor power supply, it must be small.), Q is 6.28 and the resonance band is approximately 16 MHz wide. It is reasonable to assume this band is divided evenly about the resonant frequency of 100 MHz; therefore we identify current variations between 92 and 108 MHz as resonant behavior.

Recall from Section 1 that resonance tuning changes the frequency of current variations to a frequency outside the resonance band while damping [14] addresses current variations only at the resonant frequency.

2.1.3 When should the change occur?

Q also affects how much repeated resonant behavior is required to cause noise margin violations because it determines how quickly voltage variations dissipate. $f\pi/Q$ is the damping rate (in nepers/second, not to be confused with damping, the technique, of [14]) of the circuit. A low Q indicates resonant energy dissipates quickly while a high-Q circuit more efficiently stores energy that may build into noise margin violations. In our example, voltage variations dissipate by 40% after each resonant period.

The other factor in determining how many repetitions in the resonance band are required to cause noise margin violations is the size of the current variations. As expected, current variations below a certain threshold will not cause noise-margin violations *even if the variations occur repeatedly in the resonance band.* The variations simply do not have enough energy. Of course, this threshold is less than the maximum chip current variation possible—otherwise, there is no inductive noise problem. We call the threshold as the *resonant current variation threshold.*

Variations beyond the threshold within the resonance band will lead to violations. While the threshold gives a bound below which variations can be ignored, we need to know how many repetitions of variations above the threshold can be tolerated by the power supply before a violation occurs. We call the number of repetitions as the *maximum repetition tolerance.* The larger the variations, the fewer the repetitions. Consequently, we need to know the maximum possible chip current variation within the resonance band to infer the maximum repetition tolerance. Because the processor has a well-defined peak current, minimum current, and maximum rate of change of current, the maximum current variation is not arbitrarily large, even in the resonance band. Hence, the processor's maximum current variation within the resonance band is well-defined, and along with the characteristics of the resonant circuit, determines the maximum repetition tolerance.

Intuitively, the resonant current variation threshold and maximum repetition tolerance can be computed from the energy injected into the resonant circuit by the current variations and the energy dissipated in the resonant circuit in each period (based on Q). However, the values can also be determined through circuit simulations using tools such as Spice or Matlab, as we describe in Section 4. To do so, we need to make one change to the circuit in Figure 1(a). Because we are interested in solving for the effects of changes in the current source, we use the linearity properties of the circuit in Figure 1(a) to eliminate the voltage source, yielding the circuit in Figure 1(b).

We extend our example to give a feel for these values. To obtain the maximum current variation that can be tolerated by the power supply in our example, we simulate the power supply excited with periodic current waveforms at the edges of the resonance band. We determine that the power supply can withstand current variations up to 13 amps peak-to-peak at the edges of the resonance band of 92-108 MHz (Section 2.1.2). Note that the peak-to-peak variation is limited to 13 amps only *near the resonance band*, larger variations are allowed elsewhere (and are absorbed by the power supply). Similarly, we determine that repeated variations larger than 10 amps inside the resonance band causes violation of the noise margin of ±5%, assuming a V_{dd} of 2 V. Thus, 10 amps is the resonant current variation threshold for our example. Next, we simulate the circuit to see how many repetitions of current variations of magnitude 13 amps at the resonant frequency are needed to cause a noise margin violation. This count determines the maximum repetition tolerance, which we count in half waves (i.e., a full period counts as 2). The value is 6 in our example. Note that in a real system, the maximum processor current variation is not determined by the power-supply characteristics, but the other way around.

By establishing how many repetitions of current variations larger than the resonant current variation threshold are allowable, resonance tuning tracks and manages these repetitions without the need for real-time voltage sensing as required by [10], or estimates of current as required by [14].

2.1.4 What should trigger the change?

Although resonant behavior can be detected through either voltage variations as done in [10], or current variations, there are disadvantages to using voltage. Many voltage variations do not cause noise-margin violations because they are caused by current variations outside the resonance band. [10] needlessly reacts to those changes. Furthermore, even in the absence of current variations, supply voltage rings *at exactly the resonant frequency* as an echo of leftover effects of past variations. Even if [10] were to monitor voltage variations only at the resonant frequency, the ringing would trigger unnecessary reactions. Thus, detecting resonance through voltage variations results in unnecessary reactions, causing performance and energy degradation.

In contrast to supply voltage, variations in microprocessor current directly relate to the potential for future noise-margin violations. Microprocessor core current does not echo from past resonant behavior as the core circuitry is *not* part of the resonant loop of the power supply impedance, solder-bump inductance, and the on-die d-caps, as shown in Figure 1(b). This point is subtle in that the processor core current does *vary* according to processor activity, but does not *echo* due to the resonant loop. There is not sufficient capacitance in the core circuitry for this current to echo within the resonance band. (However, the current might echo at frequencies much higher than the resonance band).

Although the d-caps are distributed throughout the die, it is not difficult to sense the current. Because the d-caps are placed *in bulk* in large empty spaces or under busses [7], it is possible to sense the current between the d-cap bulks and core circuitry. We can detect microprocessor core current using a small number of on-die current sensors. Note that we do not wish to *estimate* present or future current as in [14]; we wish to *sense* directly the present current. A coarse sensitivity to within a few amps is adequate because active microprocessor currents are quite large (on the order of a hundred amps) and we need only identify variations larger than the resonant current variation threshold.

Examples of techniques for sensing on-die current are discussed by [19] and [12]. [19] proposes a technique to compute current for chip testing by using a differential transistor pair to measure the voltage drop across supply lines. The measurement is accurate because it depends on only the relative voltage difference across two points, and does not require an accurate, external reference. [12] proposes another technique to implement on-die current sensors with a sensitivity of nanoamps to enable I_{DDQ} testing of quiescent current in high-performance chips. These sensors use Lorentz-force effects and a MAGFET (magnetic field-effect transistor) in a standard CMOS process to sense processor current without placing any resistance in series with the power supply. To achieve the high sensitivity of microamps required for I_{DDQ} testing, many of these sensors must be placed at leaves of the power-supply network and perform tens of thousands of samples compared to a reference current, making them quite slow (in the KHz range) [12]. However, for the coarse sensitivity required for resonance tuning, only a few sensors are needed at the roots of the supply network, and a precise reference current is not needed. For coarse readings, the sensors can use a single sample and run at or near processor clock speed (GHz range).

2.2 Low-frequency Resonance

Microprocessor packages exhibit an additional peak of high impedance at a low frequency due to off-chip inductance and on-chip d-caps. These components (*not* shown in Figure 1(a)) create a similar RLC loop with the power source as the off-chip power supply and the core to the entire microprocessor package and core. Because off-chip inductances and capacitors are quite large, the corresponding resonant frequency is in the range of a few megahertz.

Although the fairly small low-frequency impedance peak

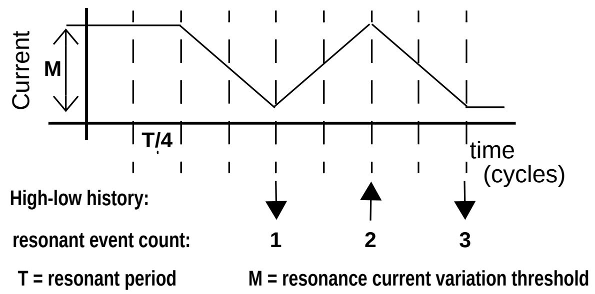

FIGURE 2: High-low history to detect resonant events

in current technology is not as serious a threat as medium-frequency resonance [8, 6], that may not always be the case as more low-resistance, high-current power supplies are developed. Fortunately, resonance tuning can be applied to both medium- and low-frequency resonance.

3 Resonance Tuning

Resonance tuning provides architectural detection of nascent resonant behavior and prevention of that behavior from building into noise-margin-violating resonance by moving current variations away from the resonant frequency. First we discuss detection and then prevention.

3.1 Detection of Resonant Behavior

In this section, we explain how resonance tuning uses the maximum repetition tolerance and resonant current variation threshold defined in Section 2.1.3 and current sensing as discussed in Section 2.1.4 to detect nascent resonant behavior. We wish to classify potentially resonant waveforms as shown in Figure 2. Only those waveforms larger than the resonant current variation threshold with frequencies inside the resonance band are of concern. We begin by explaining how our technique detects resonant behavior for a resonant frequency with a period of T cycles; later we extend our technique to cover the frequency range of the entire resonance band. We explain first how to identify resonant waveforms, and then how to count repetitions of resonant waveforms.

3.1.1 Identifying resonant waveforms

Resonant waveforms are identified by transitions from high current to low current (or vice versa) at the resonant frequency. Such a transition takes a half-period (T/2) of cycles as shown by the resonant waveform in Figure 2 and is indicated by a period of high (or low) current for the first quarter period (T/4 cycles) and a low (or high) current for the second quarter period. We identify the waveform by comparing the sum of the individual-cycle currents in the first quarter period to the sum of the individual-cycle currents for the second quarter period. For instance, the difference between the two sums for a triangle wave of peak-to-peak magnitude X is XT/8 (for a sine wave, this value is XT/2π). We identify a half period with a difference in the quarter-period current sums of MT/8 or more, where M is the resonant current variation threshold, as one *resonant event*.

To perform the identification, we maintain a history of CPU-core current for the last T/2 cycles as reported each cycle by current sensors such as those discussed in Section 2.1.4, in the *current history register*. Each cycle, we also compute a sum of the individual-cycle currents in the most-recent and second-most recent quarter periods. If the difference between the sums exceeds MT/8, we note either a high-low or low-high resonant event depending on the sign of the difference.

3.1.2 Counting repetition of resonant events

Once we can detect resonant events, we count their repetition to identify nascent resonant behavior. Repetition occurs when two or more resonant events of opposite polarity (i.e., high-low followed by low-high or vice versa) occur half-period (T/2 cycles) apart. At any given time, we need to know how many resonant events are affecting the power supply, which we call the *resonant event count*.

The resonant event count depends on the number of repeated resonant events. To track repetition, we maintain a *high-low history register* and a *low-high history register* which contain the history of each polarity of resonant events for enough cycles to cover the maximum repetition tolerance as defined in Section 2.1.3. Each register contains one bit per cycle. Each detected resonant event is noted for that cycle in the appropriate history register. When a new event is detected, we look back into the high-low and low-high histories a half period ago for a resonant event with appropriate polarity indicating nascent resonance, and determine the resonant event count. For example, if we detect a high-low resonant event in this cycle, and there were a low-high event T/2 cycles ago and a high-low event T cycles ago, then we have a resonant event count of three.

As resonant events leave the high-low and low-high history registers, the resonant event count decreases. Recall from Section 2.1.3 that in the absence of additional current variations, voltage variations die down at the damping rate defined in terms of Q. In our example, the circuit loses 40% of its energy after one resonant period. The history registers are long enough to hold as many resonant periods as the maximum repetition tolerance. Therefore, by the time a resonant event leaves the registers the residual effect of the event is low enough that the resonant event count can decrease.

Counting nascent resonant events and taking advantage of the maximum repetition tolerance allows resonance tuning to avoid many unnecessary reactions unlike [10], which responds to all detected variations beyond a threshold.

3.1.3 Extension into resonance band

In this section, we extend our detection scheme to cover all frequencies in the resonance band. First, we extend identifying resonant events to the entire resonance band. Then, we extend counting the repetitions of resonant events to the entire band.

To illustrate, we use our example from Section 2 which has a resonance band from 92 to 108 MHz with the resonant frequency at 100 MHz. Assuming a 5 GHz clock frequency, the resonance band ranges from 46 to 55 cycles, so the half-wave periods range from 23 to 28 cycles. Instead of looking for resonant events only over the half resonant period (25 cycles), we identify resonant events over all the half periods in the band

(23-28 cycles). One simplification that aids in the implementation is that we can use the same current history register for all periods. However, we need to compute separately the quarter-period sum for each period using separate adders.

Once we have detected the resonant events throughout the resonance band, we track them using the same single pair of high-low and low-high history registers. However, instead of looking into the history registers for consecutive events only half resonant periods apart (25 cycles), we look for consecutive events at all half periods in the resonance band (23-28 cycles). Looking into the registers does not need expensive associative searches. Just probing the registers at known, fixed locations half periods away from the current cycle suffices.

It is possible for a single resonant event to be detected at multiple periods in the resonance band. This possibility occurs when there is a large current variation spanning several processor cycles. Such a variation will be recorded in the high-low and low-high registers repeatedly over all those cycles, being detected as several resonant events. However, we should count the variation as only one resonant event. To that end, events of the same polarity occurring in consecutive processor cycles in the high-low or low-high registers, count only once in the resonant event count.

3.2 Prevention

In this section, we explain how we prevent noise-margin violations by changing the frequency of resonant behavior away from the resonance band. If the resonant event count reaches the maximum resonance tolerance, a noise-margin violation may occur. As established in Section 2.1.3, current variations are non-violating as long as the number of repetitions does not exceed the maximum repetition tolerance. Therefore we must take action *before* the resonant event count becomes that high. Fortunately, there is ample time between increase in the resonant event count (half time periods in the band — 23 to 28 cycles in our example), so our response need not be instantaneous.

We use a two-level response system to change the frequency of behavior. The first level tries to steer behavior to a non-resonant frequency with a small impact on performance and energy, while the second forces behavior away from the resonance band, with a much larger impact. The brute force of the second level is necessary to *guarantee* that noise margin violations do not occur.

The first-level response is engaged when a new resonant event occurs and the resonant event count is greater than or equal to the *initial response threshold* (which of course is less than the maximum repetition tolerance). The response is simply to reduce the processor issue width and the number of memory ports available for a specified period of time, called the *initial response time*. Doing so lowers the frequency at which instructions move through the pipeline, lowering the frequency of current variations. (It would be difficult to *raise* the frequency of current variations because that would imply the processor was not initially running at peak performance.)

If the first-level response is ineffective, the resonant event count will continue to climb. In this case, when a new resonant event occurs and the resonant event count reaches one below the maximum repetition tolerance, we must engage the second-level response. The second-level response forcibly lowers the frequency of current variations by stalling processor issue while maintaining a medium level of processor current. The medium level of current is achieved by “issuing” phantom operations similar to those in [10] and [14]. These phantom operations consume current but do no useful work. Because it stalls the processor and consumes extra energy, the secondary response is quite expensive.

It is important *both* to stall the processor *and* to set the processor current to a medium level. If the processor were not stalled, the frequency of current variations might not be reduced, and if the current were not set to a medium level, the act of stalling itself might increase the resonant event count and cause a noise margin violation. To avoid noise-margin violations, the second-level response must remain engaged until the resonant event count reduces by at least one.

Because of the expense of the second-level response, we wish to maximize the effectiveness of the initial response. There is a trade-off between the performance penalty of a longer initial response time and the avoidance of the second-level response.

We conclude this description by analyzing the effect of response delay. Both magnitude-based techniques in [10] and [14] require fast response to prevent noise-margin violations. [10] does not exploit resonant behavior for detection. As such, [10] detects only a few (1-5) cycles before a violation, requiring quick reaction. [14] must make per-instruction, per-cycle decisions at instruction issue to avoid current variations at the resonant frequency. In contrast, the effectiveness of resonance tuning is not affected by delays as long as quarter resonant periods. Because it takes half a resonant period for the resonant event count to increase and a full resonant period for a resonant event to repeat, a delay of even a quarter resonant period allows ample time for a first-level or second-level response. Such delays may decrease the effectiveness of the first-level response because there is less time to take effect, but they will not reduce the brute-force effect of the second-level response.

Scaling trends favor resonance tuning. Because technology scaling leads to larger C while L stays about the same, the resonant frequency (Section 2.1.1) reduces with each technology generation. Combined with rising clock frequencies, the number of processor cycles in a resonant period increases with each generation. This trend implies that resonance tuning has more time to sense, detect, and react in the future. While a quarter of a resonant period is 12 cycles in our example, it will be 50 cycles in a 10 GHz processor with a 50 MHz resonant frequency. In contrast, the allowable response times in [10] and [14] worsen with scaling. [10] does not exploit resonant behavior and must still respond quickly to voltage changes, while slow-scaling wire delays in obtaining data from sensors spread across the die and in sending clock-gate signals upon a threat put more pressure on the sensor response times. [14] must still make per-cycle decisions at instruction issue within an ever-shrinking clock cycle time.

Table 1: System parameters.

Architectural Parameters	
Instruction issue	8, out-of-order
Reorder buffer & LSQ	128 entries
L1 caches	64K 2-way, 2 cycle, 2 ports
L2 cache	2M 8-way, 12 cycles
Memory latency	80 cycles
Fetch	up to 8 instructions/cycle
Int ALU & mult/div	8 & 2
FP ALU & mult/div	4 & 2
Power Distribution and Power Parameters	
Vdd & Clock	1.0 V, 10 GHz
Max & min. current	105 A, 35 A
R, L, C	375 $\mu\Omega$, 1.69 pH, 1500 nF
Resonant frequency	100 MHz
Resonance band	84-119 cycles
Max repetition tolerance	4
Resonant current variation threshold	32

3.3 Implementation Overheads

Finally, we look at the area, performance, and energy overheads of resonance tuning. The current sensors proposed in [12] consume approximately 1000 transistors but do not insert any resistance in series with the power supply. Therefore, sensors do not add much to area or energy. As we need to store and sum current histories accurate only to the nearest ampere, the current-history values and sums can use 7-bit integers. In our example, up to 9 current-history adders are needed to cover all the frequencies in the resonance band. The area of this circuitry is small, and the per-cycle energy of the adders is approximately equivalent to that of one 64-bit adder. The high-low and low-high histories consist of n-bit shift registers, where n is the number of processor cycles in the maximum-repetition tolerance, or about 150 in the example from Section 2.

The performance impact of resonance tuning, however, is limited to the degradations caused by the first -and second-level responses. None of the current or high-low history registers are on the critical path of the processor, and there are no invasive changes which may affect the issue logic timing as in [14]. Tuning coarsely controls the issue width and memory-port availability but does not create specific requirements on the type of instructions that are selected for issue.

4 Methodology

Simulating of inductive-noise techniques requires careful selection of power-supply design-parameters and voltage/current models not normally considered in architectural simulation. First, we discuss our architectural and power simulator, and our extensions to add a power-supply simulator. Next, we discuss our design parameters.

4.1 Simulator

Our base simulator is Wattch [2] extended to include code from SimpleScalar 3.0b [3] to execute the Alpha ISA. The architectural configuration of our simulator is shown in Table 2. Wattch provides a power and clock-gating model, but does not consider processor current or current variation. We determine processor current by dividing power by supply voltage. Because Wattch simulates per-event current (e.g., cache access) rather than per-cycle current (e.g., cache access spread over 2 cycles), we add extensions to spread the current of multi-cycle operations over the appropriate pipeline stages, as was done in [10] and [14].

Current variation levels depend heavily on the clock-gating model—more aggressive gating leads to more variation. We use the aggressive clock-gating model from Wattch except that we do not allow clock-gating of the global clock components (which may be difficult to gate in a real system). [10] further limited clock-gating to include only functional units, the writeback bus, and caches.

The power-supply model is implemented in the simulator using the circuit shown in Figure 1(b). We use the Huen Formula (Improved Euler Formula) [1] to solve the system of equations to simulate supply-voltage variations based on processor core current. While more accurate equation solvers, exist, we found the Huen Formula to be both simple and adequate. While real power supplies incur a voltage drop due to power-supply impedance (IR drop due to the R in Figure 1), the drop is unrelated to inductive noise (which is voltage variation due to di/dt). Therefore, we ignore the IR drop and assume that the power-supply is capable of maintaining a supply voltage of V_{dd} at any constant current level.

We model the current sensors for resonance tuning by assuming that we can determine each cycle's current to the nearest whole-amp. We also model the energy overhead of key components from the resonance tuning detection and prevention hardware as discussed in Section 3.3, though such overhead is small (<1% of processor energy).

4.2 Design parameters

In Section 2, we described a power-supply network with characteristics similar to today's processors. However, we use parameters for an aggressive, future design for our simulations. We choose an aggressive design point such that inductive noise is a problem in some SPEC benchmarks (as will be shown in the next section) but that noise-margin violations do not occur continuously or outside the resonance band (which would characterize an unrealistic, poorly-designed power supply). Our parameters are shown in Table 2. We scale Wattch to assume a 1.0V Vdd and a clock-frequency of 10 GHz with a peak power of 105 W. (The supply voltage, frequency, and power correspond to projections for a 7FO4 pipeline in a 57 nm technology [17].) We allocate 1500 nF of on-die d-caps (approximately 3-4 times larger than those on the 21264 [7]), a parasitic inductance of 1.69 pH, and a power-supply impedance of 375 $\mu\Omega$. A noise margin violation is assumed if the supply voltage deviates more than 0.05 V (5%) from V_{dd}.

These parameters yield a fairly high resonant frequency of 100 MHz, which is conservative for our simulations because resonance tuning has fewer cycles to respond prevent a noise-margin violation, as discussed in Section 3.2. Based on the exact equations from [4], the resonance band extends from 119 MHz to 83.9 MHz, or between 84 and 119 cycles. The remaining parameters are computed as discussed in Section 2.1.

For our simulations, we run all 26 SPEC2K applications with ref inputs. We fastforward 2 billion instructions to skip initialization code and then run 500 million instructions. The base IPC for each application, with no resonance tuning, is shown as part of Table 2.

5 Results

In, Section 5.1 we show resonant behavior in a microprocessor power-supply and show an example of resonance causing a noise-margin violation. Then we classify the SPEC2K applications by their noise-margin violations. In Section 5.2, we show results for resonance tuning. In Section 5.3 we compare resonance tuning to the previously-proposed techniques of [10] and [14].

5.1 Resonant behavior in microprocessors

5.1.1 Voltage variation in the power supply

In this section, we show the voltage response of the simulated power supply when stimulated by known waveforms (as opposed to microprocessor current). We use the circuit from Figure 1(b) with the parameters listed in Table 1. We will show that repeated resonant events lead to noise-margin violations and that resonant energy dissipates quickly from the power supply.

The top graph of Figure 3 depicts supply-voltage variations for our power supply when the processor current is as shown in the bottom graphs. (Recall that by using the circuit in Figure 1(b), we eliminate the supply-voltage source itself and that we subtract out any IR voltage drop (Section 4.2)--so the steady state value of the supply voltage should be 0.) The bottom graph depicts processor current, a 34-amp square wave beginning at cycle 100 and ending in cycle 500. The numbers above the current plot depict resonant events as counted by the resonant event count (Section 3.1.2).

The square current wave is larger than the resonance variation threshold of 32 amps and hence is large enough to cause noise-margin violations. As we see in the top graph, the noise margin is violated once when the resonant event count is equal

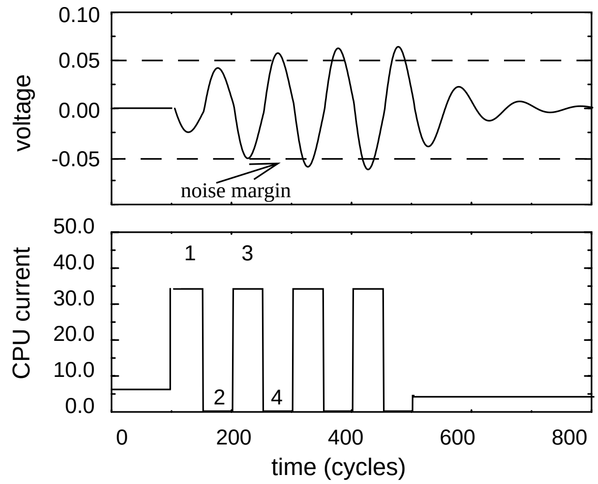

FIGURE 3: Stimulation at resonant frequency.

to four, which is the computed maximum repetition tolerance value shown in Table 1.

The dissipation rate of the voltage variations after the current waveform stops is noteworthy. Voltage variations in this power supply dissipate at a rate of 66% per resonant period (100 cycles). As discussed in Section 2.1.3, this rate depends on Q, which is computed from R, L, and C to be 2.83 for this power supply. (This Q value is lower than that for the example in Section 2, and hence the faster dissipation rate for this power supply.)

5.1.2 Voltage variations in applications

Now we extend our analysis of current and voltage variations to microprocessor current. We use Figure 4 to illustrate typical noise-margin violation and the advance warning provided by the resonant event count. The figure shows processor voltage variation (top graph), processor core current (middle graph), and resonant event count (bottom graph) for a 400-cycle sample of execution in *parser.* As can be seen in the top graph, a noise-margin violation occurs near cycle 300. Current variations occur within the resonance band at roughly 100 cycle intervals (the long, flat current around cycle 250 is due to an L2 miss).

The resonant event count, as reported by the high-low and low-high histories described in Section 3.1, is used to trigger resonance tuning. In the graph, the event count increases to the maximum resonance tolerance of four as we approach the violation. The bottom graph shows that a resonant event count of 2 is reached approximately 150 cycles before the violation, a

Table 2: Classification of SPEC2K applications.

Applications with noise-margin violations													
name	applu	art	bzip	crafty	facerec	gcc	lucas	mcf	mgrid	parser	swim	wupwise		
IPC	1.97	1.49	2.19	2.25	2.60	2.13	0.85	0.38	2.88	1.71	1.99	3.47		
fraction of cycles in violation x 10^{-6}	0.173	3.26	173	4.52	0.047	0.047	5597	0.032	2.61	64.2	2730	0.097		
Applications without noise-margin violations														
name	ammp	apsi	eon	equake	fma3d	galgel	gap	gzip	mesa	perlbmk	sixtrack	twolf	vortex	vpr
IPC	0.44	1.85	2.72	4.00	4.11	3.61	2.84	2.01	3.34	1.34	3.31	1.35	2.40	1.39

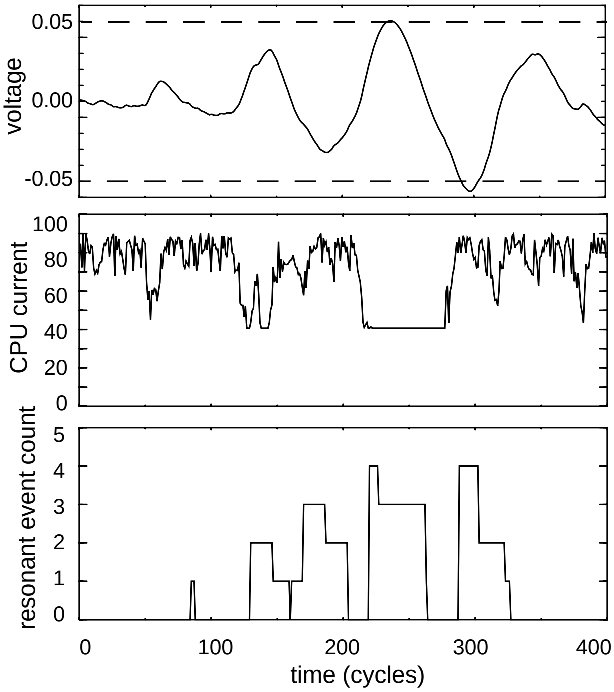

FIGURE 4: Voltage and current variation in *parser*.

resonant event count of 3 is reached just over 100 cycles before the violation, and a resonant event count of 4 is reached about 75 cycles before the violation (and during the violation itself). From these timings it is clear that resonance tuning does not need fast sensors like [10]. We also see that tracking processor current to the precision of whole amps is enough to flag imminent violations, confirming that resonance tuning does not need precise sensors like [10] nor accurate estimates like [14]. Note that the resonant event count falls off for brief periods during which the high-low history does not detect resonant events. This fall-off is unimportant because the prevention technique is engaged as soon as the initial response threshold is reached.

5.1.3 Classification of applications

In this section, we classify each SPEC2K application as either violating or non-violating depending on if our simulated processor exhibits noise-margin violations. Because we wish to simulate an aggressive future processor where inductive noise is problematic, it is important that noise margin violations occur in a number of benchmarks. While a design where noise-margin violations occur often would be unrealistic (indicating a poorly-designed power supply), a design that never exhibits noise-margin violations is not representative of future inductive-noise problems.

Table 2 shows the SPEC2K benchmarks classified by the presence of noise-margin violations, and the benchmarks' IPCs. Twelve applications exhibit violations, and the fraction of cycles (x 10^{-6}) spent in violation by these applications are given. Two characteristics of the applications are noteworthy. The first is that there is no particular correlation between IPC and noise-margin violations. Resonant behavior can occur (or not occur) in either high or low IPC applications. The second is that, as expected, the number of violations is quite low with respect to the total execution time. While this number may be low, the presence of *any* noise-margin violations is unacceptable in a real processor.

5.2 Resonance Tuning

In this section we present performance and energy results for resonance tuning. We expect resonance tuning to prevent noise-margin violations using our two-tiered response system with small performance and energy loss.

Based on the circuit values in Table 1, the resonance-tuning parameters described in Section 3.2 are set as follows. The resonant current variation threshold is 32 A. Because the repetition tolerance is 4 (at which violations can occur), we set the initial response threshold to 2. The second-level response is initiated if the event count reaches 3. Our initial response is to reduce the issue width from 8 to 4 and the number of available cache ports from 2 to 1. We vary the initial-response-time. The second-level response time is set based on the damping rate of the power supply (Section 2.1.3). In our power supply there must be no activity for 32 cycles to ensure variations will dissipate an amount equivalent to reducing the event count by 1. Therefore, we set the second-level response time to 35 cycles.

Table 3 shows results for resonance tuning for initial response times between 75 and 200 cycles over all SPEC2K applications compared to a base processor with no resonance tuning (i.e., one that allows noise-margin violations). We do not separate violating and non-violating applications as we found no substantial differences in their results. The fraction of cycles spent in first-level and second-level responses on average is shown in the second and third columns. Performance-related numbers are shown in the next three columns. Average relative energy-delay is shown in the last column.

The gentle first-level response is effective at reducing the need for the harsh second-level response. The fraction of cycles spent in second-level response (which results in complete stalls) is between 0.0027 and 0.0040. The fraction of cycles spent in first-level response is much higher, between 0.1 and 0.2. This value is quite large compared to the fraction of

Table 3: Resonance tuning.

Initial response time	Fraction of cycles in first-level response	Fraction of cycles in second-level response	Worst relative slowdown	Apps with > 15% slowdown	Avg relative slowdown	Avg relative energy-delay
75 cycles	0.10	0.0040	1.19 (wupwise)	2	1.043	1.052
100 cycles	0.12	0.0038	1.20 (wupwise)	1	1.048	1.057
125 cycles	0.15	0.0032	1.19 (wupwise)	2	1.054	1.076
150 cycles	0.17	0.0031	1.35 (galgel)	4	1.068	1.079
200 cycles	0.20	0.0027	1.27 (galgel)	5	1.075	1.088

cycles spent in noise-margin violation, shown in Table 2. This difference occurs for two reasons. First, there are many instances of nascent resonant behavior that do not build up to violations. Second, it takes many cycles of first-level response to tune out resonant behavior. Despite the high fraction of time in first-level response, performance loss is only between 4% and 8% because the first-level response is gentle.

Increases in the initial response time allow the gentler first-level response to be more effective, reducing the need for the second-level response. However, because the first-level response is applied often, excessively large initial-response times degrade performance. As we increase the initial response time from 75 to 200 cycles, the average fraction of cycles spent in the first-level response doubles, while the fraction of cycles spent in the second-level response goes down by 32%. Although the average slowdowns increase with the initial-response time, the performance loss of the worst application is lowest when the initial response time is 75 cycles, and the number of benchmarks with greater than 15% performance loss is lowest for an initial-response time of 100 cycles.

Energy-delay increases between 5% and 9%. The increase is due both to performance loss from the first- (mostly) and second-level responses, and to the extra energy associated with phantom instructions in the second-level response. For our best-performing initial response times of 75 and 100 cycles, the relative energy-delays are 5.2% and 5.7%, respectively.

We also wish to determine if resonance tuning is affected by delay between sensing nascent resonance and initiating the first-level response. As stated in Section 3.2, delays that are small compared to the resonant period will not diminish the effectiveness of resonance tuning, but they might hurt performance or energy. For an initial response time of 100 cycles and a delay of 5 cycles, performance degradation is 5.8% and relative energy-delay is 6.6% (not shown). These numbers are only 1% and 2% higher, respectively, than those with no delay.

5.3 Comparison to Previous Techniques

In this section, we compare resonance tuning to [10] and to pipeline damping [14]. First we discuss each of those techniques, and then we provide an overall comparison. We expect resonance tuning to have smaller performance and energy degradations due to its use of a gentler response than either previous technique and its ability to avoid responding to non-repetitive resonant events.

5.3.1 Technique of [10]

We implement [10] in our simulator. The best technique of [10] responds by phantom firing the L1 caches and functional units if the voltage is too high or clock-gating the L1 caches and functional units if the voltage is too low. We use their response for high voltages; for low voltages we stop fetch and instruction issue; it may be unrealistic to instantaneously clock-gate the pipeline back-end if there are instructions scheduled in those resources. It is unclear if [10] models the effect of voltage ringing on their response mechanism. Because our voltage simulator is built into Wattch and our threshold is detected directly off that voltage, our results include the effect of ringing.

Our choice of a design where many applications exhibit at least some noise-margin violations is important in evaluating resonance tuning and [10]. While [10] discussed two design points (the “300%” and “400%” target impedance points) that exhibited noise-margin violations in SPEC2K, the performance and energy results in [10] are in terms of *another* design that does not exhibit violations in SPEC2K. While this fact does not question the effectiveness of [10] at eliminating noise-margin violations (theory guarantees that), the performance and energy numbers of our design are different those reported in [10].

We use detection thresholds similar to those determined by [10] to evaluate their performance and energy. To guarantee avoidance of noise-margin violations, we actually need thresholds slightly smaller than those of [10] because our current is higher and the power supply is subject to more variation. However, the values used are conservative (i.e. they favor [10]) and are adequate for evaluating performance and energy.

Table 4 shows results for [10]. The desired detection threshold is in the first column (the high and low threshold are equal in our results because we subtract out IR voltage drop in our simulations as discussed in Section 4.1, making our threshold equivalent to half of [10]’s “safe window”.) Any noise introduced is shown in the second column, with the actual threshold (target minus half of noise) in the third. The remaining columns show the fraction of cycles during which there is a reaction, slowdown, and energy.

The first two data rows of the table show results with no noise and no delay. The performance loss for the 20 mV case is larger than that for the 30 mV case due to non-resonant variations and voltage rings that are between 20 mV and 30 mV. The average performance and energy losses are quite small. The average fraction of cycles that are part of a response for the 30 mV threshold and the 20 mV threshold are lower and higher, respectively, than the average number of second-level responses in resonance tuning. As is the case with resonance tuning, most of these responses are unnecessary. However, the performance and energy impact of *any* response in [10] is sim-

Table 4: Technique of [10]

Target threshold (mV)	Noise (mV peak-to-peak)	Actual threshold (mV)	Sensor/control delay	Fraction of cycles in response	Worst relative slowdown	Avg relative slowdown	Avg relative energy-delay
30	0	30	0	0.002	1.038 (swim)	1.005	1.030
20	0	30	0	0.04	1.180 (fma3d)	1.039	1.047
30	15	22	0	0.05	1.11 (fma3d)	1.031	1.074
20	10	15	5	0.15	1.32 (fma3d)	1.108	1.191
20	15	12	3	0.27	1.68 (galgal)	1.236	1.460

Table 5: Pipeline damping [14].

δ relative to resonant current variation threshold	Worst relative slowdown	Avg relative slowdown	Avg relative energy-delay
1	1.35 (fma3d)	1.10	1.12
0.5	1.60 (fma3d)	1.15	1.17
0.25	2.04 (fma3d)	1.24	1.26

ilar to that of the *second-level* response in resonance tuning, as both stall issue and/or phantom-fire resources. As the threshold lowers, [10] incurs more unnecessary responses. This trend only worsens when sensor noise and delay are introduced.

The ideal design points just discussed may not be realistic as discussed in Section 1. For example, it may not be possible to design accurate voltage sensors to distinguish such small voltage differences. The third row adds 15 mV peak-to-peak of noise to the voltage sensors with a targeted threshold of 30 mV, reducing the actual threshold to 22 mV. Average performance loss increases but is still relatively small at 3.1%.

However, a real system will have some delay between sensing and response to a voltage change as mentioned in [10]. [10] show results for noise and delay separately and not combined. Delay impacts the effectiveness of response and thus requires a lower target threshold. In the fourth row of results, we show a configuration with a delay of 5 cycles, a target threshold of 20 mV (lower than 30 to account for the delay, similar to [10]) *and* noise of 10 mV. Performance degradation jumps to 10.8%, and relative energy-delay increases to 19% If we increase the noise to 15 mV, the actual threshold becomes only 12 mV as shown in the fifth row. Even if we decrease the delay to 3 cycles, there are many non-resonant variations of that magnitude, causing the fraction of cycles in response nearly to double over the previous value. Average performance degradation and energy delay increase to 24% and 1.46.

5.3.2 Damping

We also implement pipeline damping as described in [14]. We apply pipeline damping at the resonant period (which is 100 cycles, making the damping window 50 cycles) and use the a priori relative current estimates given in [14] and their assumption that each unit of estimated current is equivalent to 0.5 A (scaled to our processor configuration). We initially set the worst-case current variation allowed over a resonant period (δ in [14]) to be equal to the resonant current variation threshold of 32 A and we use the "always-on" frontend-damping technique of [14].

Pipeline damping addresses variations only at the resonant frequency and not within the entire resonance band. There are two options to modify damping to account for the entire band. One is to extend the per-cycle decisions to cover the range of frequencies in the band. However, doing so would complicate the issue queue further (Section 1). Therefore, we do not use this option. The other is tightening of δ to reduce variations in the entire band (though it may require substantial tightening of δ to guarantee the noise margins). We use this option to show results for values of δ at one-half and one-quarter of our initial value. In our results, we generously assume that the extensive

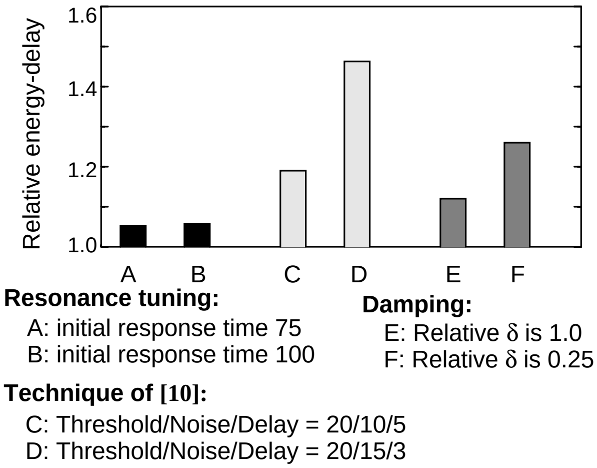

FIGURE 5: Comparison of techniques

issue-queue modifications required for pipeline damping do not impact performance.

Table 5 shows our results for pipeline damping. The performance degradations increase from 10% to 24% and the relative energy delays from 1.11 to 1.26 as we reduce the value of δ relative to the resonant current variation threshold. These values are similar to those reported in [14].

5.3.3 Comparison

We compare the relative energy-delays of resonance tuning, [10], and pipeline damping in Figure 5, showing two design points from the previous sections for each scheme. We note that it is hard to compare to damping because it does not cover the resonance band, and it has issue queue and current estimation problems. To facilitate comparison between resonance tuning and [10], we note that (1) the current sensor precision in resonance tuning is to the whole amp whereas the voltage sensor precision in [10] is 10-15 mV; (2) resonance tuning is not sensitive to the delay of 3 to 5 cycles shown for [10]. We see that resonance tuning outperforms the other two schemes.

6 Related work

While [8], [10], and [14] propose architectural solutions for medium-frequency inductive noise, other proposals focus on high-frequency noise (near processor clock frequencies). [13] proposes to ramp functional unit current slowly to reduce high-frequency noise. [15] proposes architectural techniques to reduce high-frequency noise to reduce the requirements for on-die d-caps and save leakage energy. [5] measured step current using a microarchitectural simulator.

A simultaneous architectural work on di/dt characterization appears in [11]. The authors propose a wavelet-based off-line estimation technique to analyze supply voltage variation and an on-line control technique for current variations that uses simplified wavelet-based convolution to avoid the complexity of full convolution described in [8]. Wavelet-based convolution may be an alternative to using maximum repetition tolerance and resonant current variation threshold (as described in Section 3.1) to detect resonant behavior for resonance tuning.

7 Conclusions

Inductive noise is caused by a few peaks of high impedance at the resonant frequencies of RLC loops in the power-supply network. Current variations in the band of resonant frequencies cause supply-voltage glitches beyond the noise margins. We proposed *resonance tuning* based on the observations that (1) current variations only in the resonance band are problematic whereas other variations are absorbed by the power supply, and (2) only repeated, and not isolated, current variations build up to noise-margin violations. Accordingly, resonance tuning changes the frequency of current variations away from the resonance band to non-resonant frequencies, responding only to repeated current variations and not to individual variations. Upon detecting a build up, resonance tuning uses two-tiered response of minor and major perturbations in the pipeline resource usage (e.g., stalling a few issue widths and complete stall) to tune-out current variations while they are still nascent.

Compared to previous techniques, resonance tuning incurs fewer false alarms by distinguishing between resonant and non-resonant variations, and by sensing current which avoids ringing problems of sensing voltage. Because repetitions of resonant periods are tens of cycles, resonance tuning does not need fast sensors and has ample time to use the simple and gentle first-tier response of minor stalling, unlike previous techniques. Resonance tuning also avoids previous techniques' difficulties of accurate estimates of current, real-time convolution calculations, and invasive changes to the issue queue.

Our simulations using SPEC2000 show that resonance tuning, [10], and [14] incur 5-9%, 19-46%, and 17-26% energy-delay penalty and 4-8%, 11-24%, and 15-24% performance degradation, respectively, over ranges of representative configurations. Technology scaling trends imply increasing number of processor cycles in resonant periods in the future, making resonance tuning's timings more lenient. As supply voltages fall and absolute noise margins shrink, resonance tuning will likely become important for controlling inductive noise.

Acknowledgements

This research is supported in part by NSF under CAREER award 9875960-CCR, NSF Instrumentation grant CCR-9986020, and an NSF Graduate Research Fellowship.

References

[1] W. E. Boyce and R. C. DiPrima. *Elementary Differential Equations and Boundary Value Problems*. John Wiley and Sons, Inc., 1997.

[2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, June 2000.

[3] D. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin–Madison, June 1997.

[4] R. A. DeCarlo and P.-M. Lin. *Linear Circuit Analysis*, volume 2. Prentice Hall, 1995.

[5] W. El-Essawy, D. H. Albonesi, and B. Sinharoy. A microarchitectural-level step-power analysis tool. In *International Symposium on Low Power Electronics and Design*, pages 263–266, Aug. 2002.

[6] B. Garben, M. F. McAllister, W. D. Becker, and R. Frech. Mid-frequency delta-i noise analysis of complex computer system boards with multiprocessor modules and verification by measurements. *IEEE Transactions on Advanced Packaging*, 24(3):294–302, 2001.

[7] M. K. Gowan, L. L. Biro, and D. B. Jackson. Power considerations in the design of the alpha 21264 microprocessor. In *Design Automation Conference (DAC)*, pages 726–731, 1998.

[8] E. Grochowski, D. Ayers, and V. Tiwari. Microarchitectural simulation and control of di/dt-induced power supply voltage variation. In *Eighth International Symposium on High Performance Computer Architecture (HPCA)*, pages 7–16, Feb. 2001.

[9] D. J. Herrell and B. Beker. Modeling of power distribution systems for high-performance microprocessors. *IEEE Transactions on Advanced Packaging*, 22(3):240–248, 1999.

[10] R. Joseph, D. Brooks, and M. Martonosi. Control techniques to eliminate voltage emergencies in high-performance processors. In *Ninth International Symposium on High Performance Computer Architecture (HPCA)*, pages 79–90, Feb. 2003.

[11] R. Joseph, Z. Hu, and M. Martonosi. Wavelet analysis for microprocessor design: Experiences with wavelet-based dI/dt characterization. In *Tenth International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2004.

[12] H. Kim, D. M. H. Walker, and D. Colby. A practical built-in current sensor for IDDQ testing. In *IEEE International Test Conference*, pages 405–414, 2001.

[13] M. D. Pant, P. Pant, and D. S. Wills. On-chip decoupling capacitor optimization using architectural level prediction. *IEEE Transactions on VLSI Systems*, 10(3):319–326, 2002.

[14] M. D. Powell and T. N. Vijaykumar. Pipeline damping: A microarchitectural technique to reduce inductive noise. In *Proceedings of the 30th International Symposium on Computer Architecture (ISCA 30)*, June 2003.

[15] M. D. Powell and T. N. Vijaykumar. Pipeline muffling and a-priori current ramping: Architectural techniques to reduce high-frequency inductive noise. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 223–228, Aug. 2003.

[16] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits*. Prentice Hall, 2002.

[17] SIA. *International Technology Roadmap for Semiconductors (ITRS)*. http://public.itrs.net/, 2003.

[18] M. Tsuk, R. Dame, D. Dvorscak, C. Houghton, and J. S. Laurent. Modeling and measurement of the alpha 21364 package. In *IEEE 10th Topical Meeting on Electrical Performance of Electronic Packaging*, pages 283–286, 2001.

[19] J. P. M. van Lammeren. Iccq: A test method for analogue vlsi based on current monitoring. In *IEEE International Workshop on IDDQ Testing*, pages 24–28, 1997.

[20] B. Wicht, D. Schmitt-Landsiedel, and S. Paul. A simple low voltage current sense amplifier with switchable input transistor. In *27th European Solid-State Circuits Conference*, Sept. 2001.